

**A trust management delegation mechanism for fog
computing systems**

Phongsathon Fongta

**In partial fulfilment of the requirements for the degree of
Doctor of Philosophy**

Computer and Information Sciences

University of Strathclyde

April 2026

Declaration

This thesis is the result of the author's original research. It has been composed by the author and has not been previously submitted for examination which has led to the award of a degree.

The copyright of this thesis belongs to the author under the terms of the United Kingdom Copyright Acts as qualified by the University of Strathclyde Regulation 3.50. Due acknowledgement must always be made of the use of any material contained in, or derived from, this thesis.

Signed: Phongsathon Fongta

Date: 15 April 2026

Published Work

Research in this thesis has been published in the 2024 IEEE International Conference on Cyber Security and Resilience (CSR)

P. Fongta and S. Terzis, "A Trust Management Delegation Protocol for Fog Computing Applications", 2024 IEEE International Conference on Cyber Security and Resilience (CSR), London, United Kingdom, 2024, pp. 1-6, doi: 10.1109/CSR61664.2024.10679386.

Acknowledgements

I would like to express my sincere gratitude to my supervisors, Dr Sotirios Terzis and Dr Jide Edu, for their valuable time, support, and guidance throughout my PhD studies. The completion of this PhD research would not have been possible without their assistance. I would like to thank the Department of Computer Science at Strathclyde University and all the staff members who have offered me considerable support over my PhD over several years.

I am deeply grateful to the members of my advisory committee for their thoughtful suggestions, critical insights, and encouragement during the different stages of this work.

I am extremely grateful to the Thai government for the financial support that provides me with numerous opportunities to pursue a PhD and wonderful life experiences while studying in the UK.

Finally, I would like to say a special thank you to my family and friends, particularly my dad and mom, who gave personal support and encouragement throughout my PhD journey in the UK, especially during the difficult time of the COVID-19 pandemic.

Thank you all for being part of this journey.

Abstract

Fog computing environments consist of a wide range of heterogeneous devices, from high-end servers to resource-constrained IoT sensors. This heterogeneous nature introduces significant challenges for effective trust management. Resource-constrained devices, particularly IoT devices at the edge of the fog architecture, often lack the computational, storage and energy capacities required for effective trust management. Furthermore, existing trust management approaches in fog computing treat all participating devices as homogeneous entities and assume a single trust management architecture for all devices in the system.

This research proposes a trust management delegation protocol that addresses the limitations of resource-constrained devices by enabling them to delegate trust management tasks to more capable nodes in fog computing systems. With this approach, computationally intensive trust management tasks are delegated from limited devices to resource-rich nodes. The protocol also considers the varying capabilities of devices to ensure efficient trust management across diverse environments. The experimental results demonstrate that the proposed delegation approach provides an effective solution for these limited devices.

The contributions of this doctoral research are composed of identifying key issues affecting trust management in fog computing systems, analysing a set of trust management requirements that consider the diverse capacities of participating devices and the varying relationships between devices in different application scenarios, and developing a trust management delegation protocol that enables offloading trust management tasks to more capable nodes within the fog system.

List of Contents

Chapter 1: Introduction	1
1.1 Research motivation.....	1
1.2 Research problem.....	2
1.3 Research questions and thesis statement.....	3
1.4 Research methodology	3
1.5 Research outcome	4
1.6 Contributions to knowledge	5
1.7 Thesis organisation.....	5
Chapter 2: Related work and literature review	7
2.1 Trust management.....	7
2.2 Trust properties	12
2.3 Trust-related attacks	16
2.4 Trust-related misbehaviour	20
2.5 Fog computing	21
2.6 Systematic literature review.....	28
2.7 Trust management in cloud computing and the Internet of Things	31
2.8 Task offloading protocols in fog computing.....	34
2.9 Trust management in fog computing	36
2.10 Research gaps.....	43
2.11 Conclusion	44
Chapter 3: Fog computing scenarios and their trust relationship analysis.....	54
3.1 Scenario selection.....	54
3.2 Smart homes scenario	55

3.3	Smart traffic scenario	63
3.4	E-health scenario	72
3.5	The requirements for trust management architecture.....	80
3.6	Conclusion	82
Chapter 4: Trust management delegation protocol and design.....		83
4.1	Why develop a trust management delegation protocol.....	83
4.2	Device capability classes.....	84
4.3	Trust delegation components	85
4.4	Trust management roles and delegation services.....	86
4.5	Service discovery functions	89
4.6	Trust management delegation functions	96
4.7	Access control for trust management delegation	111
4.8	Recommendation sharing.....	115
4.9	Mobility management	115
4.10	Conclusion	117
Chapter 5: The implementation of trust management delegation protocol.....		118
5.1	Why use Python and CoAP	118
5.2	Protocol message exchange.....	119
5.3	Protocol repository	124
5.4	Trust update.....	130
5.5	Delegation expiration and lease renewal.....	131
5.6	Device identifier.....	132
5.7	Service discovery functions	133
5.8	Trust management delegation functions	139
5.9	Resource access control	159

5.10	Mobility management	161
5.11	Conclusion	163
Chapter 6: Experiments and results.....		164
6.1	Experimental evaluation questions	164
6.2	Experimental methodology and setup.....	165
6.3	Trust metrics and models	167
6.4	Experiment 1: Local trust management and delegation.....	172
6.5	Experiment 2: Resource consumption under different trust models and trust metric configurations	191
6.6	Experiment 3: Resource consumption under different evidence collection and trust model configurations.....	209
6.7	Comparison with traditional trust management systems and task-offloading schemes	234
6.8	Discussion	236
6.9	Conclusion	238
Chapter 7: Conclusion and future work		239
7.1	Research questions	239
7.2	Contributions.....	241
7.3	Limitations and future work.....	241
7.4	Thesis conclusion	243
Bibliography.....		244
Appendix A: Implementation source code.....		264

List of Figures

Figure 2.1: The relationship of trust.....	10
Figure 2.2: The trustor can receive recommendations about the trustee from other nodes	10
Figure 2.3: Fog computing architecture	22
Figure 2.4: PRISMA diagram	30
Figure 3.1: Fog-based smart home architecture	56
Figure 3.2: An architecture of smart traffic system based on fog computing.....	63
Figure 3.3: Fog-based e-health system.....	73
Figure 4.1: The process of delegation service registration	89
Figure 4.2: The process of delegation service deregistration.....	91
Figure 4.3: The process of delegation service registration lease time renewal.....	92
Figure 4.4: The process of delegate(s) lookup	93
Figure 4.5: Pull TSS discovery process	95
Figure 4.6: Push TSS discovery process	95
Figure 4.7: The process of delegating all roles to one delegate	97
Figure 4.8: The process of delegating all roles to multiple delegates.....	100
Figure 4.9: The process of delegating trust evidence collector role to a delegate ...	103
Figure 4.10: The process of delegating trust repositor role to a delegate	104
Figure 4.11: The process of delegating trust computer role to a delegate	106
Figure 4.12: The process of delegating trust computer and repositor roles to one delegate	108
Figure 4.13: The process of delegating trust computer and trust repositor roles to multiple delegates.....	110

Figure 4.14: The process of trust value and trust evidence access permission delegation	113
Figure 4.15: The process of trust value and trust evidence access permission delegation revocation	114
Figure 4.16: The process of trust management delegation handover.....	116
Figure 5.1: CoAP message format	120
Figure 5.2: Delegation service registration	133
Figure 5.3: Delegation service deregistration	134
Figure 5.4: Delegation registration lease renewal.....	135
Figure 5.5: Delegate lookup.....	136
Figure 5.6: Pull TSS discovery	138
Figure 5.7: Push TSS discovery	139
Figure 5.8: Delegating all roles to one delegate.....	140
Figure 5.9: Delegating all roles to multiple delegates.....	143
Figure 5.10: Delegating evidence collector role	148
Figure 5.11: Delegating trust repositor role	149
Figure 5.12: Delegating trust computer role	152
Figure 5.13: Delegating trust computer and repositor roles to one delegate	154
Figure 5.14: Delegating trust computer and repositor roles to multiple delegates ..	156
Figure 5.15: Granting trust value and trust evidence access permission	159
Figure 5.16: Revoking trust value and trust evidence access permission	160
Figure 5.17: Trust delegation handover process	161
Figure 6.1: Experiment 1 setup	172
Figure 6.2: CPU time used by an end node on doing trust locally versus delegating trust.....	174

Figure 6.3: RAM used by an end node on doing trust locally versus delegating trust	176
Figure 6.4: The point at which RAM consumption performing trust locally becomes higher than the RAM consumption of delegating trust.....	177
Figure 6.5: HDD used by an end node on doing trust locally versus delegating trust	177
Figure 6.6: Energy consumption by an end node on doing trust locally versus delegating trust	179
Figure 6.7: CPU time used by a delegate when performing trust using a weighted sum on behalf of 1 to 15 delegators	180
Figure 6.8: RAM used by a delegate when performing trust using a weighted sum on behalf of 1 to 15 delegators.....	181
Figure 6.9: HDD used by a delegate when performing trust using a weighted sum on behalf of 1 to 15 delegators.....	181
Figure 6.10: Energy used by a delegate when performing trust using a weighted sum on behalf of 1 to 15 delegators.....	182
Figure 6.11: CPU time used by a fog node when performing trust management for 1 to 15 trustee end nodes.....	185
Figure 6.12: RAM used by a fog node when performing trust management for 1 to 15 trustee end nodes.....	187
Figure 6.13: HDD used by a fog node when performing trust management for 1 to 15 trustee end nodes	187
Figure 6.14: Energy used by a fog node when performing trust management for 1 to 15 trustee end nodes.....	188
Figure 6.15: Experiment 2 setup for evaluating an end node	191
Figure 6.16: Experiment 2 setup for evaluating a fog node.....	192

Figure 6.17: CPU time used by an end node when performing trust computation using fuzzy logic locally versus delegating trust computation	193
Figure 6.18: RAM used by an end node when performing trust computation using fuzzy logic locally versus delegating trust computation	195
Figure 6.19: HDD used by an end node when performing trust computation using fuzzy logic locally versus delegating trust computation	196
Figure 6.20: Energy used by an end node when performing trust computation using fuzzy logic locally versus delegating trust computation	198
Figure 6.21: CPU time used by a fog node when performing trust computation locally using weighted sum versus fuzzy logic	200
Figure 6.22: RAM used by a fog node when performing trust computation locally using weighted sum versus fuzzy logic	201
Figure 6.23: HDD used by a fog node when performing trust computation locally using weighted sum versus fuzzy logic	202
Figure 6.24: Energy used by a fog node when performing trust computation locally using weighted sum versus fuzzy logic	203
Figure 6.25: CPU time used by a fog node on evaluating trust of end nodes versus evaluating trust of fog nodes with different trust properties	205
Figure 6.26: RAM used by a fog node on evaluating trust of end nodes versus evaluating trust of fog nodes with different trust properties	206
Figure 6.27: HDD used by a fog node on evaluating trust of end nodes versus evaluating trust of fog nodes with different trust properties	207
Figure 6.28: Energy used by a fog node on evaluating trust of end nodes versus evaluating trust of fog nodes with different trust properties	208
Figure 6.29: Experiment 3 setup	209
Figure 6.30: CPU time used by an end node performing trust management delegation with self-evidence collection and delegated evidence collection	211

Figure 6.31: RAM used by an end node performing trust management delegation with self-evidence collection and delegated evidence collection	212
Figure 6.32: HDD used by an end node performing trust management delegation with self-evidence collection and delegated evidence collection	214
Figure 6.33: Energy used by an end node performing trust management delegation with self-evidence collection and delegated evidence collection	215
Figure 6.34: CPU time used by a delegate when performing trust on behalf of 1 to 10 delegators with and without evidence collection	217
Figure 6.35: RAM used by a delegate when performing trust on behalf of 1 to 10 delegators with and without evidence collection	218
Figure 6.36: HDD used by a delegate when performing trust on behalf of 1 to 10 delegators with and without evidence collection	220
Figure 6.37: Energy used by a delegate when performing trust on behalf of 1 to 10 delegators with and without evidence collection	221
Figure 6.38: CPU time used by an end node when performing trust computation locally using weighted sum versus fuzzy logic.....	223
Figure 6.39: RAM used by an end node when performing trust computation locally using weighted sum versus fuzzy logic.....	225
Figure 6.40: HDD used by an end node when performing trust computation locally using weighted sum versus fuzzy logic.....	226
Figure 6.41: Energy used by an end node when performing trust computation locally using weighted sum versus fuzzy logic.....	227
Figure 6.42: CPU time used by a delegate when performing trust on behalf of 1 to 15 delegators using weighted sum versus fuzzy logic	229
Figure 6.43: RAM used by a delegate when performing trust on behalf of 1 to 15 delegators using weighted sum versus fuzzy logic	230
Figure 6.44: HDD used by a delegate when performing trust on behalf of 1 to 15 delegators using weighted sum versus fuzzy logic	232

Figure 6.45: Energy used by a delegate when performing trust on behalf of 1 to 15 delegators using weighted sum versus fuzzy logic	233
---	-----

List of Tables

Table 2.1: Mapping of search strings to SLR research questions	29
Table 2.2: The breakdown of search terms and their results.....	31
Table 2.3: Comparison of trust management approaches in fog computing.....	46
Table 2.4: Comparison of trust models for trust management approaches in fog computing.....	48
Table 2.5: Comparison of trust aspects for trust management approaches in fog computing.....	49
Table 2.6: Comparison of evaluation methods for trust management approaches in fog computing	53
Table 4.1: Classes of devices in fog computing systems	84
Table 4.2: Access permissions to trust value and trust evidence for different roles	112
Table 5.1: Database for each delegation role	125
Table 5.2: Service_Registration table	126
Table 5.3: Delegation_info table.....	127
Table 5.4: Trust_value table.....	128
Table 5.5: Trust_evidence table	128
Table 5.6: Roles table.....	129
Table 5.7: Trust_model table	129
Table 5.8: E_storage.....	129
Table 5.9: E_collector table	130
Table 5.10 Decision_making_method table.....	130
Table 5.11: Access_permission table.....	130
Table 6.1: Distance scores.....	169
Table 6.2: Packet loss rate scores.....	170

Table 6.3: Network latency values.....	171
Table 6.4: CPU time used by an end node on doing trust locally versus delegating trust.....	175
Table 6.5: RAM used by an end node on doing trust locally versus delegating trust	176
Table 6.6: HDD used by an end node on doing trust locally versus delegating trust	178
Table 6.7: CPU time used by a delegate when performing trust using a weighted sum on behalf of 1 to 15 delegators.....	183
Table 6.8: RAM used by a delegate when performing trust using a weighted sum on behalf of 1 to 15 delegators.....	183
Table 6.9: HDD used by a delegate when performing trust using a weighted sum on behalf of 1 to 15 delegators.....	184
Table 6.10: Energy used by a delegate when performing trust using a weighted sum on behalf of 1 to 15 delegators.....	184
Table 6.11: CPU time used by a fog node when performing trust management for 1 to 15 trustee end nodes.....	189
Table 6.12: RAM used by a fog node when performing trust management for 1 to 15 trustee end nodes	189
Table 6.13: HDD used by a fog node when performing trust management for 1 to 15 trustee end nodes	190
Table 6.14: Energy used by a fog node when performing trust management for 1 to 15 trustee end nodes	190
Table 6.15: CPU time used by an end node when performing trust computation using fuzzy logic locally versus delegating trust computation	194
Table 6.16: RAM used by an end node when performing trust computation using fuzzy logic locally versus delegating trust computation.....	196

Table 6.17: HDD used by an end node when performing trust computation using fuzzy logic locally versus delegating trust computation	197
Table 6.18: Energy used by an end node when performing trust computation using fuzzy logic locally versus delegating trust computation	198
Table 6.19: CPU time used by a fog node when performing trust computation locally using weighted sum versus fuzzy logic.....	200
Table 6.20: RAM used by a fog node when performing trust computation locally using weighted sum versus fuzzy logic.....	201
Table 6.21: HDD used by a fog node when performing trust computation locally using weighted sum versus fuzzy logic.....	202
Table 6.22: Energy used by a fog node when performing trust computation locally using weighted sum versus fuzzy logic.....	203
Table 6.23: CPU time used by a fog node on evaluating trust of end nodes versus evaluating trust of fog nodes with different trust properties	205
Table 6.24: RAM used by a fog node on evaluating trust of end nodes versus evaluating trust of fog nodes with different trust properties	206
Table 6.25: HDD used by a fog node on evaluating trust of end nodes versus evaluating trust of fog nodes with different trust properties	208
Table 6.26: Energy used by a fog node on evaluating trust of end nodes versus evaluating trust of fog nodes with different trust properties	209
Table 6.27: CPU time used by an end node performing trust management delegation with self-evidence collection and delegated evidence collection	211
Table 6.28: RAM used by an end node performing trust management delegation with self-evidence collection and delegated evidence collection.....	212
Table 6.29: HDD used by an end node performing trust management delegation with self-evidence collection and delegated evidence collection.....	214
Table 6.30: Energy used by an end node performing trust management delegation with self-evidence collection and delegated evidence collection	215

Table 6.31: CPU time used by a delegate when performing trust on behalf of 1 to 10 delegators with and without evidence collection	217
Table 6.32: RAM used by a delegate when performing trust on behalf of 1 to 10 delegators with and without evidence collection	219
Table 6.33: HDD used by a delegate when performing trust on behalf of 1 to 10 delegators with and without evidence collection	220
Table 6.34: Energy used by a delegate when performing trust on behalf of 1 to 10 delegators with and without evidence collection	221
Table 6.35: CPU time used by an end node when performing trust computation locally using weighted sum versus fuzzy logic.....	223
Table 6.36: RAM used by an end node when performing trust computation locally using weighted sum versus fuzzy logic.....	225
Table 6.37: HDD used by an end node when performing trust computation locally using weighted sum versus fuzzy logic.....	226
Table 6.38: Energy used by an end node when performing trust computation locally using weighted sum versus fuzzy logic.....	227
Table 6.39: CPU time used by a delegate when performing trust on behalf of 1 to 15 delegators using weighted sum versus fuzzy logic	229
Table 6.40: RAM used by a delegate when performing trust on behalf of 1 to 15 delegators using weighted sum versus fuzzy logic	231
Table 6.41: HDD used by a delegate when performing trust on behalf of 1 to 15 delegators using weighted sum versus fuzzy logic	232
Table 6.42: Energy used by a delegate when performing trust on behalf of 1 to 15 delegators using weighted sum versus fuzzy logic	233
Table 6.43: Comparison of the trust delegation protocol against other trust management systems.....	235
Table 6.44: Comparative analysis of different task offloading algorithms.....	236

Chapter 1: Introduction

This chapter introduces the research motivation, defines the research problems and research questions. It then presents the research methodology, summarises the research outcomes and contributions to knowledge, states the thesis statement, and provides an overview of the thesis organisation.

1.1 Research motivation

Smart devices and Internet of Things (IoT) devices generate large volumes of data, which are typically uploaded to cloud data centres for processing and storage. However, due to the distance to the cloud data centres, these devices face many challenges, such as network congestion, high response latency, and location awareness [1]. Fog computing was developed to address these cloud issues. It comprises heterogeneous devices, including base stations, routers, switches, mini-servers, and other networking equipment, that act as fog nodes in close proximity to users at the edge of the network. End devices with limited resources, particularly IoT devices, can offload heavy tasks and data to fog nodes for further processing and storage, rather than directly to cloud data centres. As a result, offloading tasks to fog nodes reduces the amount of data sent to the cloud. Fog computing offers lower network latency and decreased application response times, thereby enabling real-time interactions for time-sensitive applications that cloud computing cannot support [2].

However, concerns about trust arise in fog computing. Can fog nodes be trusted to behave correctly, or might they be malicious, i.e., offering poor quality of service, mishandling data, or failing to protect sensitive data? Similarly, end devices connecting to fog nodes may not always behave well, potentially uploading incorrect or harmful data, which could lead to system disruptions and failures. Hence, trust is an important issue that must be addressed to ensure the security and reliability of fog computing systems. It is essential for any device to avoid interacting with misbehaving and malicious devices and to isolate them from the system. Trust management can assist devices in selecting trustworthy devices that will provide reliable services and

behave appropriately. It monitors and tracks devices within the system and assesses their trustworthiness in specific contexts.

Heterogeneous devices participate in fog computing environments, ranging from high-end servers to tiny IoT devices. These devices possess diverse computational, storage, and network capabilities and operate on different operating systems. Resource-constrained devices, particularly end devices, might lack the capabilities required to perform trust management. Furthermore, their main responsibility is to perform their main tasks. They may lack the resources to execute trust management effectively, failing to identify trustworthy devices. This limitation could make the entire fog computing system vulnerable to malicious devices.

1.2 Research problem

A significant amount of research has been conducted to propose trust management solutions for handling the trust of different entities in fog computing. Some approaches focus on the trust of end nodes toward fog nodes [3], while others emphasise the trust of fog nodes toward end nodes [4]. Trust among end nodes and fog nodes themselves for data sharing or collaborative tasks has also been considered [5], [6]. However, these trust management approaches often assume a single trust management architecture and treat all end devices in a fog system as the same type, without considering their capabilities. In fact, devices vary significantly in computational and storage capabilities, as well as in hardware and software components. Devices in a fog system can include IoT devices, switches, routers, embedded sensors, laptops, smartphones, and more [7].

With the variety of devices in the system, some devices may be capable of performing trust evaluation on their own, while resource-limited devices, particularly IoT devices, cannot because of their limited computational capabilities, insufficient storage, and energy constraints. Some devices with limited capabilities may choose to use other nodes to perform trust computation or rely on their peers. Meanwhile, fog computing devices with high computing power could be willing to handle trust calculations, and devices with large storage capacities could be suitable for storing and managing trust information. Existing research has not taken into account the differences between

devices in terms of their resources. When considering variations in device capabilities, it is not reasonable to have a single trust architecture for all devices. The trust architecture should be flexible enough to accommodate these differences between participating devices.

1.3 Research questions and thesis statement

This study investigates and answers the following research questions (RQs):

- **RQ1:** What are the trust management requirements for managing trust between devices participating in fog computing systems?
- **RQ2:** How can we design an architecture for trust management in fog computing that addresses the identified requirements?
- **RQ3:** How can we show that the architecture is effective in supporting the different trust management scenarios?
 - How can we provide a proof-of-concept implementation?
 - How can we show that the implementation accommodates participating devices to carry out their trust management?

Thesis statement: A trust management delegation approach can accommodate the differences between devices and assist resource-limited devices to take advantage of the resources available in fog computing to support their trust management.

1.4 Research methodology

This research employs the following methodology to review existing research literature, and to design, implement and evaluate a trust management delegation protocol for fog computing environments.

1. Literature review

A systematic literature review of existing literature on trust management in the fog computing domain, relevant systems, and task-offloading schemes was conducted. The literature review identifies existing trust management approaches, summarises trust aspects and models in these approaches, and identifies research gaps related to device heterogeneity and resource constraints.

2. Requirements analysis

Based on the literature review and an analysis of representative fog computing application scenarios, including smart homes, e-health, and smart traffic management, the specific requirements for effective trust management among heterogeneous and resource-constrained devices were established.

3. Protocol design

A trust management delegation protocol was designed to address the identified requirements. The protocol architecture supports delegation of trust management tasks from resource-limited devices to more capable nodes. It supports context-aware trust metrics and ensures lightweight communication using the CoAP protocol.

4. Implementation

The proposed protocol was implemented in Python, utilising the CoAP protocol for efficient message exchange. The implementation supports dynamic trust services discovery, delegation, and trust evaluation among diverse devices in a fog computing environment.

5. Experimental evaluation

Experiments were conducted using simulation to assess the performance and resource efficiency of the implemented protocol. Metrics including computation, memory and storage usage, and energy consumption were measured and compared against other trust management approaches.

1.5 Research outcome

- A comprehensive review of existing research works on trust management in the fog computing domain.
- An identification of trust management requirements required for fog computing application scenarios.
- A design of a trust management delegation protocol that meets the requirements.
- A proof-of-concept implementation of the trust management delegation protocol using Python, utilising the CoAP protocol for message exchange.

- An experimental evaluation using simulation demonstrating the performance of the implemented trust management delegation protocol.

1.6 Contributions to knowledge

The thesis has made the following contributions to knowledge on trust management in fog computing environments:

- The research identified key issues affecting trust management in fog computing systems. Unlike previous studies that treated all devices as homogeneous, this study highlights the heterogeneity of devices operating in fog environments.
- The work proposed a set of trust management requirements that consider the diverse computational, memory, storage, and energy capabilities of participating devices and the varying relationships between devices in different application scenarios.
- This thesis developed a trust management delegation protocol that enables resource-limited devices to offload trust management tasks to more capable nodes within the fog system. The proof-of-concept implementation has been demonstrated to address the identified requirements effectively.

1.7 Thesis organisation

The thesis contains 7 chapters and is organised as follows:

Chapter 2 introduces fog computing, including its architecture and the components in each layer. The chapter also provides a systematic literature review on trust management in traditional IoT and cloud systems, as well as in fog computing systems. Furthermore, existing trust management approaches in the fog computing domain are examined to identify research gaps.

Chapter 3 reviews fog computing applications, including smart homes, smart e-health, and smart traffic management. The participating devices and their trust relationship in each application scenario have been analysed and presented. The chapter also identifies the requirements of trust management architecture for our trust management delegation design and implementation.

Chapter 4 presents a trust management delegation protocol that delegates trust management tasks to a delegate, thereby reducing resource consumption for the delegator. This chapter focuses on the design of the delegation protocol, including service discovery functions, trust management delegation functions, role-based access control for the delegation protocol, and mobility management.

Chapter 5 presents the implementation of the delegation protocol, which was designed in Chapter 4. The implementation leverages the CoAP protocol to make the delegation message exchange lightweight and suitable for different device types in fog computing systems. SQLite is used to store delegation information, trust evidence and value, and other relevant protocol-related information. Each function of the delegation protocol is explained in detail, including the parameters of request and response messages.

Chapter 6 evaluates the proof-of-concept implementation of the delegation protocol. This chapter begins by explaining how aspects of trust used to evaluate the trustworthiness of trustees are collected, as well as the trust models used in the evaluation. The result of the evaluation shows resource consumption in terms of CPU time, RAM usage, HDD usage, and energy consumption when performing the delegation protocol compared to performing trust management locally and other aspects.

Chapter 7 provides the thesis conclusion, discussion, and future work. Additionally, some suggestions for improving the delegation protocol are provided.

Chapter 2: Related work and literature review

The objective of this chapter is to present related work and relevant research to enhance the readers' understanding of trust management in fog computing systems. Initially, the concept of trust and the management of trust relationships between entities will be described. This will be followed by a review of trust management in traditional cloud computing and Internet of Things systems to understand trust management more generally. Subsequently, the background knowledge of fog computing will be introduced, along with fog computing architecture, characteristics, and challenges. Next, the systematic literature review methodology employed to identify the literature is presented. Finally, the discussion of existing research on trust management in fog computing systems will be provided with a classification of trust management approaches with respect to trust relationships, trust composition, trust propagation, trust aggregation and formation, and research evaluation approaches.

2.1 Trust management

2.1.1 Definition of trust

Trust is a complicated concept that relates to the relationship between two participants and the belief of one participant in another. There is no consensus on the definition of trust. Grandison and Sloman [8] defined trust as “the firm belief in the competence of an entity to act dependably, securely and reliably within a specified context”. Cho et.al [9] defined trust as “the willingness of the trustor (evaluator) to take risk based on a subjective belief that a trustee will exhibit reliable behaviour to maximise the trustor’s interest under uncertainty (e.g. ambiguity due to conflicting evidence and/or ignorance caused by complete lack of evidence) of a given situation based on the cognitive assessment of past experience with the trustee”. Therefore, trust is a concept relating to the belief of a trustor towards a trustee in some contexts. Trust is not only related to security but also to many other aspects, such as goodness, strength, reliability, availability, ability, etc., of an entity or service [10].

Trust can be estimated based on personal observations, but we can go beyond that by using recommendations from others [11]. So, a recommendation is third-party

information about the trust of a trustee from an entity that has already interacted with the trustee in the past [11]. Recommendations can be combined with personal observations to evaluate the trustworthiness of a trustee [11].

2.1.2 Trust and reputation

The concept of reputation is closely linked to the concept of trust, but it is different. Reputation and trust are frequently used interchangeably in a lot of research [10]. Trust uses past evidence to predict an entity's behaviour, whereas reputation is not a prediction of the future, but it is knowledge of the past for decision-making [12]. Mousa et al. [13] gave a definition of reputation as "an aggregated opinion of the community members about how much an individual or an entity can be trusted". Therefore, the reputation of a particular entity is an aggregation of personal observations from fellow entities and can be used as a recommendation to determine the amount of trust towards a trustee [13]. Further, Ruhomaa et al. [12] differentiate trust from reputation by highlighting that trust emphasises risk and incentives, whereas reputation focuses on the actions of a party acting in the past regarding its intentions and norms.

The notion of reputation has been extensively employed within society, where each individual has a reputation score derived from their past actions. Two popular instances of reputation-based trust management are Amazon Auctions and eBay [14]. After a successful transaction, both sellers and purchasers have the ability to evaluate one another, and the resulting ratings are recorded for future reference [14]. People can therefore determine the trustworthiness of someone based on their reputation score [14].

2.1.3 Types of trust management

Several trust management approaches have been proposed in distributed systems to manage trust among entities [12]. Reputation-based trust management [14] enables entities to establish and quantify trust based on the historical actions and feedback of other network participants. Each node collects local experiences and recommendations and then computes a trust score for each entity. This approach is suitable for decentralised environments such as peer-to-peer or ad hoc networks. However, it can

be vulnerable to malicious behaviours like false recommendations and collusion attacks [15]. Policy-based trust management [8] relies on explicit rules and credentials, such as digitally signed credentials and organisational policies, for evaluating trust relationships. This method is well-suited for structured environments where access control and compliance are vital. However, it requires credentials or policy management [8].

Behaviour-based trust management [16] derives trust from the direct observation of entity actions, such as packet forwarding or the cooperative behaviour of their neighbour. This trust management is often used in ad hoc and sensor networks to identify malicious nodes. However, it can be resource-intensive and consumes energy on constrained devices [16]. Certificate-based trust management [12] establishes trust through digital certificates, which are issued by centralised authorities (CAs) or distributed mechanisms. Certificates serve as attributes to prove trust relationships. Entities validate others by checking certificate signatures. Although it offers strong trust, certificate management and validation can be computationally expensive for resource-constrained nodes [16].

This research focuses on evidence-based trust management. In this approach, Trust is collected and assessed using quantifiable data, such as Quality of Service (QoS) metrics [17]. This approach is especially relevant in areas like cloud computing and the Internet of Things (IoT). The details of this trust management approach will be detailed in subsequent sections.

2.1.4 How trust management works

In computing systems, a device may cooperate with other devices to perform tasks or require other resource-rich devices to handle resource-intensive tasks [18]. The trustworthiness of a device may change over time, or a device may work in a satisfactory manner from time to time [19]. Therefore, trust management keeps track of devices and allows trust establishment between two parties, called the trustor and the trustee [19]. Trust management assists the trustor in predicting the future behaviour of the trustee [8]. The trustor is a service requester needing to assess the trustworthiness of the behaviour of the trustee. The trustee needs high trust values to

be selected as a provider. With the prediction, it can select a trustee that will provide the best service and be well-behaved. Figure 2.1 shows a trust relationship between the trustor and the trustee.

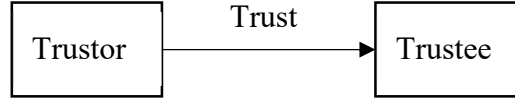


Figure 2.1: The relationship of trust

The trustor can also receive recommendations from others to consider the trustworthiness of the trustee [17], as shown in Figure 2.2. The overall trust value that the trustor has with the trustee might be a combination of its opinion and recommendations from different recommenders [11].

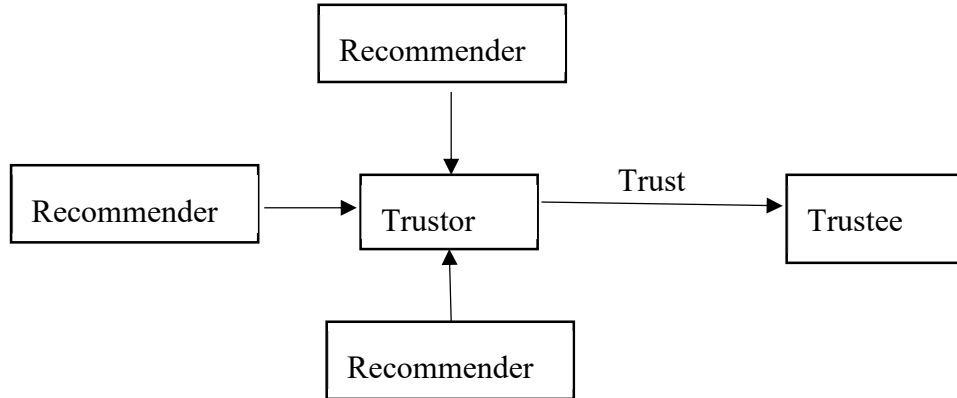


Figure 2.2: The trustor can receive recommendations about the trustee from other nodes

2.1.5 Trust computation methods

To manage trust among devices, Guo et al. [17] provide a classification of trust computation methods. The classification contains 5 dimensions: trust composition, trust propagation, trust aggregation, trust update, and trust formation.

- Trust composition

Trust composition defines the components that are considered in trust computation. Quality of service (QoS) and social trust properties are usually used for trust evaluation in environments such as the Internet of Things [17]. QoS defines the service quality level of a service provider serving a service requester, such as performance,

throughput, and latency. Social trust in devices can derive from relationships between devices or from their owners [9]. For example, two devices can form a social relationship with each other because they have frequent interactions or belong to the same owner. Examples of social trust properties are honesty and cooperativeness [9].

- **Trust propagation**

Trust propagation refers to methods to propagate trust evidence to entities involved in a system [17]. There are three approaches: centralised, decentralised, and distributed [17]. The centralised approach requires a centralised server that might be a node in the system to compute and store trustworthiness data for all devices in the system. All devices can access trust values from the centralised server. The decentralised approach allows all nodes in a system to compute and store trust values themselves. A service requester can get an opinion about a service provider from other devices that they have encountered or interacted with for estimating trust. This is done without a centralised server. In a distributed system, nodes are divided into multiple clusters, with each being managed by a cluster head. The cluster head is responsible for trust calculation and trust data collection, similar to a centralised server.

- **Trust aggregation**

Trust aggregation is the method used to aggregate trust evidence collected by either direct trust or indirect trust from other entities [17]. Examples of trust aggregation methods are weighted sum, belief theory, Bayesian inference, fuzzy logic, and regression analysis [17]. A trustor can aggregate its trust towards a trustee with evidence that can be either direct, if they directly interact with each other, or indirect through recommendation trust towards the trustee from others if they do not have an interaction [17]. The recommendations received from recommenders can be weighed differently depending on the credibility of each recommender or the trustor's trust towards each recommender [17].

- **Trust update**

Trust update defines when trust is updated [17]. There are two approaches: event-driven and time-driven [17]. In the event-driven approach, all trust data is updated after an event occurs. For instance, when a device receives service from a service

provider, it rates the quality of the service and updates trust data in its storage or sends the trust values to a centralised server. In the time-driven approach, trust evidence is collected, and trust is updated periodically, such as every hour or day.

- **Trust formation**

Trust formation describes how multiple trust properties are formed into an overall trust [17]. There are two approaches: single-trust and multi-trust approaches [17]. The single-trust approach considers only one trust property in trust computation. It is usually the most important trust property. However, in cases when the resources of the devices are severely constrained, it may be an alternative due to the reduced need for storage and calculation [9]. The multi-trust approach considers trust as multi-dimensional. It takes into account several trust properties to compute the trustworthiness of an entity. There are two ways to form trust in a multi-trust approach [17]. First, a threshold value is set for every trust property. If the value of one trust property is lower than its threshold, the entity is considered untrustworthy. Second, all trust properties are combined into one value. Each trust property is assigned a weight within a weighted sum. Different weights can be used if a trust property is more important than another.

2.2 Trust properties

Trust properties, also referred to as trust metrics, are measurable attributes used to evaluate the trustworthiness of a device [17], [20]. These properties can be classified into two categories: Quality of Service (QoS) properties and social trust properties [17]. QoS properties reflect the capability and operational performance of a device in delivering services, whereas social trust properties reflect the behaviour of a device. The following subsections provide trust properties from both categories and clarify how they are measured. However, numerous trust metrics are available for evaluating trust.

2.2.1 Task completion rate

Task completion is a QoS trust metric that evaluates the capability of a device to successfully finish a delegated task [6]. The assumption is that the higher the task completion rate, the higher the possibility of the device having the capability to

complete the task. A node maintains a task history log, which records the past transactions of offloading tasks. This task history log will be used to calculate the job completion rate, which is the ratio of the number of successful tasks to the total number of tasks offloaded to a particular node.

2.2.2 Availability

Availability is a QoS metric that indicates whether a service provider is operational and accessible to provide services whenever a service requester requests [21]. It is one of the most fundamental trust properties in distributed and IoT systems. An overloaded node may be unavailable and may reject incoming requests. Availability is estimated as the ratio of the number of accepted tasks to the total number of submitted tasks [21].

2.2.3 Packet loss rate

Packet loss rate is a QoS metric that captures the reliability of data transmission between a sender and receiver [22]. Packet loss causes delays, as lost packets must be retransmitted [22]. A trustor requires reliable data transfer from a trustee. A trustee with a high packet loss rate would receive a lower trust score because the trustor cannot rely on the integrity and completeness of data transfers. Packet loss rate is calculated as the ratio of the number of packets dropped to the total number of packets sent by a sender [22].

2.2.4 Network latency

Network latency is a measure of the delay in data transmission [22]. It is the time taken for a packet to travel from a sender to a recipient across the network. Network latency is therefore a measure of network transmission performance. It is typically measured by the round-trip time for data transmitted from a sender to a recipient and the time of an acknowledgement packet to get back to the sender [22].

2.2.5 Distance

Distance is defined as the number of network hops between a sender and a recipient [23]. Data sent by the sender is transmitted through multiple intermediate devices, like routers, to reach the recipient. Every hop the data passes through causes a delay and increases the risk of packet loss [24]. Moreover, the short distance provides more

reliable and faster communication [24]. Distance is particularly relevant in fog computing, where closer nodes are preferred for latency-sensitive tasks [24]. The trustor can measure distance using a network diagnostic tool such as Traceroute [24].

2.2.6 Response time

Response time represents the total time required for a delegated node to receive, process, and return a result for a request made by a requester [25]. It is measured as the round-trip time for the data to reach a recipient, perform processing, and return a result to the requester [25]. It encompasses both network latency and processing time. This distinguishes response time from network latency, which excludes processing. Response time is a critical QoS indicator for time-sensitive fog applications [25].

2.2.7 Network throughput

Network throughput represents the amount of data that a sender can successfully transfer to a recipient per unit of time [25]. In other words, it shows the rate at which data is successfully transferred between two endpoints (e.g., between an end node and a fog node). Throughput is particularly relevant for trust evaluation in scenarios involving large data uploads, such as video surveillance [22]. Throughput is measured as the total number of bytes transferred from the source to the destination in bits per second [22].

2.2.8 Reputability

Reputability is a social trust property that reflects the belief that a trustee node does not engage in malicious activities, e.g., Denial-of-Service (DoS) and malware attacks, to a trustor node [26]. A trustor tracks the number of times it has observed the trustee behaving maliciously. Once this count exceeds a predefined threshold, the trustee is considered disreputable [26].

2.2.9 Cooperativeness

Cooperativeness is a social trust property that reflects the degree to which a node is willing to collaborate with others in the network [27]. The assumption is that a malicious node discriminatively performs data breaches to non-friends or nodes without strong social ties, while nodes with stronger relationships are more likely to

behave honestly. In other words, a cooperative node has a low possibility of misusing data. In contrast to availability, cooperativeness is a social metric capturing the node's behaviour toward others. A node may be available but uncooperative. Cooperativeness can be computed as the ratio of the number of successful connection requests with the specific node to the total number of requests [28]. Additionally, the degree of cooperativeness can be computed as the ratio of common friends between two devices [29]. In this case, each device maintains a list of its friends that it has interacted with in the past.

2.2.10 Ownership

Ownership is a social trust property that reflects the relationship between devices that belong to the same owner [28]. In the Social Internet of Things (SIoT) literature, the Ownership Object Relationship (OOR) refers to the relationship between objects owned by the same person, such as a mobile phone and a game console owned by the same person, regardless of whether those objects are of the same type [30]. The assumption behind this metric is that devices owned by the same person, household, or organisation are more likely to behave reliably than devices belonging to unknown parties. Roy et al. [31] also mentioned that “devices having a co-owner relationship may connect among themselves with maximum trust and very low risk”. However, ownership is not treated as a guarantee of trustworthy behaviour. Ownership is used only as one factor in trust estimation and is combined with other properties, such as availability and reputability, rather than being used on its own.

However, one limitation of this assumption is that a device may be shared or accessed by an authorised but different user. One example is intimate partner abuse, where a partner misuses legitimate access to a device to behave maliciously [32]. The Android Platform Security Model [32] offers a useful notion of multi-party authorisation to handle the diverse interests of multiple stakeholders. In future work, the proposed protocol may apply this idea to trust management. Ownership should not be treated as belonging entirely to one party. Instead, the trustworthiness of a device could be assessed separately for each stakeholder relevant to it, for example, a household member and an authorised co-user, rather than assuming that every interaction with the device shares the same level of trust. These separate trust levels could then be

combined, or used individually depending on the situation, to evaluate the trustworthiness of the device.

2.3 Trust-related attacks

Trust management plays a key role in ensuring the trustworthiness of devices within a fog computing system [33], [34]. However, trust management systems are vulnerable to various trust-related attacks that can undermine their effectiveness [34]. The following trust-related attacks are presented as background knowledge, and the proposed delegation protocol does not defend against all of the attacks.

- Self-promotion attacks (SPA) occur when a malicious device increases its reputation by providing false positive feedback for itself [34]. This behaviour deceives the trust management system into treating the attacker as highly reputable, allowing it to use that status for further malicious activities. The countermeasure was a trust credibility model in [33] that mitigated the attack by adjusting and discounting inflated self-reported ratings from malicious nodes. Furthermore, SPA can also be addressed by Trust2Vec, proposed by Dhelim et al. [35]. Trust2Vec is a large-scale IoT trust management system based on signed network embeddings. It addresses self-promotion attacks by constructing a signed trust graph and applying a random-walk embedding algorithm to capture relationships among nodes.
- Whitewashing attacks involve attackers with low trust levels leaving their current identities and re-entering the system with new identities [34]. The purpose is to hide their previous negative behaviour and to avoid historical trust tracking. The countermeasures include device fingerprinting and integration of Physical Unclonable Functions (PUFs) to bind identities to unique hardware characteristics, preventing anonymous re-entry [34].
- Sybil attacks refer to a node that generates multiple pseudonymous identities for malicious purposes, such as giving a lot of good or bad feedback [15]. In fog environments, the resource-constrained and heterogeneous nature of devices raises serious concerns about verifying the uniqueness of device identities. Prior works have proposed mitigation through effective authentication and by restricting

devices from obtaining multiple identities to ensure that devices in a system are held accountable for their activities [15].

- Bad-mouthing attacks (BMA) involve a malicious node that attempts to damage a target node's reputation by issuing numerous negative recommendations [36]. As a result, the target node suffers a decline in reputation, and the malicious node has a higher chance of being selected as a service provider [36]. However, BMA can be detected using a detection model based on the deep long short-term memory (LSTM) technique proposed in [37], which achieved an accuracy of 99.28% in identifying untrusted entities.
- Ballot-stuffing attacks (BSA) occur when a malicious node provides false positive recommendations about other malicious nodes to boost their reputation and increase their probability of being selected as service providers [34]. Bad-mouthing attacks aim to degrade the reputation of honest nodes while BSA elevate dishonest ones. Mitigation of BSA requires validating recommendations against direct observations [36]. Alternatively, Chen et al. [38] proposed a trust management protocol with credibility filtering of indirect trust recommendations. In addition, the protocol also employs an adaptive filtering technique to dynamically adjust the weighting of direct and indirect trust.
- Collusion attacks consist of a group of malicious nodes collaborating to increase their reputation by ballot stuffing or collaborating on bad-mouthing other nodes [39]. Mitigating these attacks requires statistical analysis of recommendations from others, such as making a decision based on a majority of them [39]. Additionally, Douss and Abassi [40] introduced a detection approach against collusion attacks in a MANET environment. The mechanism monitors the recommendation behaviour of nodes within each cluster and identifies colluding nodes by detecting statistically abnormal patterns in their trust ratings.
- Opportunistic service attacks (OSA) involve a malicious node that provides good service to other nodes to receive a higher level of trust during periods of low reputation [36]. Once it achieves a high level of trust, it colludes with other malicious nodes on bad-mouthing and ballot-stuffing attacks [36]. To mitigate this, nodes in a system have to record all activity from other nodes. If a malicious node has been known to misbehave in the past, it should be marked. With this

method, it is difficult for the malicious node to recover its trust in the system. A Dirichlet-based detection scheme proposed in [41] is employed to detect opportunistic attackers through statistical Dirichlet distribution modelling of node behavioural patterns.

- On-off attacks (OOA) involve a malicious node that provides good and bad services randomly to avoid being labelled as an untrusted node [42]. A reward and punishment approach can protect against this type of attack [42]. Rewards are given when good service is provided, and punishment is given in case of providing bad service, and the punishment should be twice the weight of the reward. So, this makes a malicious node hard to recover its trust.
- Replay attacks refer to a malicious entity intercepting legitimate data transmissions and sending them again to pose as a trusted node [43]. The replay attack can cause service provider nodes to process out-of-date or forged trust evidence. Strong encryption and time-based authentication methods can mitigate this attack. Additionally, Shaaban et al. [43] proposed an authentication scheme using Elliptic Curve Cryptography (ECC) that resists replay attacks in fog computing.
- Discriminatory attacks (DA) refer to a malicious node that selectively provides unfair levels of service quality to specific nodes, rather than actual service quality [44]. To mitigate this attack, Jafarian et al. [44] introduced a discriminative-aware trust management scheme for the social Internet of Things that incorporates social relationships into the trust calculation process to better detect discriminatory behaviours and minimise service quality bias.
- Conflicting behaviour attack (CBA) refers to a malicious node that performs contradictory or inconsistent behaviour to different requestors simultaneously [45]. Unlike the on-off attack, which operates temporally, it operates conflicting behaviour across concurrent interactions with multiple nodes. To defend against CBA, Ren and Qin [45] proposed a novel multidimensional trust evaluation and fusion mechanism for fog-based IoT environments that collects diverse trust opinions from multiple network entities. Their system employs the Jensen-Shannon divergence measure (JSDM) to detect inconsistencies in behavioural observations, with the analytic hierarchy process (AHP) to assign weights to trust decision factors.

- Flooding attacks are denial-of-service attacks in which a malicious node sends an excessive volume of packets or service requests to a target node in order to exhaust its bandwidth, memory, or processing resources, making the service unavailable to legitimate users [46]. To defend against flooding attacks, Zarei and Fotuhi [46] proposed an approach called LSFA-IoT that protects IoT networks and detects flooding nodes in IoT-fog infrastructure. The LSFA-IoT detection process monitors a weighted average of suspicious activity that exceeds a predefined probabilistic threshold, allowing the system to identify the source of flooding attacks.
- Stealthy attacks are advanced cyber threats in which a malicious node keeps its harmful activities below the detection threshold of trust models by adopting a gradual and low-intensity misbehaviour strategy [47]. The malicious node degrades service quality, for example, by slightly increasing packet drop rates or marginally delaying responses, instead of performing aggressive attacks that would trigger a detection system [47]. To defend against this type of attack, a decentralised trust management framework based on Multi-Agent Reinforcement Learning (MARL) with Independent Deep Q-Networks (I-DQN) was proposed [48]. The approach can adapt its detection policies through dynamic trust evaluation, enabling it to identify subtle, low-intensity misbehaviour patterns associated with stealth attacks.

Traditional cloud-based trust management relies on a centralised cloud server for trust evaluation. The cloud has a large amount of resources to support more complex mechanisms for dealing with trust-related attacks. However, it often suffers from high delay due to its distance from end devices. In contrast, IoT-based trust management usually performs trust management locally on each device. IoT devices have limited resources and cannot support advanced methods for detecting and handling attacks. The proposed protocol overcomes this limitation by leveraging available resources in fog systems to allow resource-constrained devices to delegate trust computation to nearby capable nodes. This improves the efficiency of trust evaluation in fog computing systems while maintaining a lower response time than the cloud-based approach.

2.4 Trust-related misbehaviour

Misbehaviour can be intentional and unintentional [49]. Intentional misbehaviour is divided into malicious and selfish purposes [49]. The malicious behaviour aims to carry out cyber-attacks (e.g., Denial-of-Service (DoS) and malware attacks) or cause data breaches, while the selfish behaviour aims to gain advantages. Unintentional or accidental behaviour happens due to faulty or damaged devices.

2.4.1 Intentional misbehaviour

A malicious node, which may be managed by an attacker, might send many messages to disrupt service, making a targeted node unavailable to provide services [50]. Moreover, it could send malicious code to the targeted node [50]. The attacks can be detected by monitoring the data transmission rate to see whether it exceeds a normal rate [51]. Moreover, it can be detected by considering the number of times a node sends the same data [52]. The same message can be retransmitted by the same node only a limited number of times. A malware attack can be detected using several techniques, including those described in [15]. Besides, the malicious node has a high possibility to misuse data that is stored in its storage or being processed for wrong purposes, e.g., analysing the power consumption of a household can reveal whether customers are at home [52]. This type of behaviour is difficult to detect. However, social trust aspects can decrease the possibility of contacting nodes that perform data breaches [53].

A selfish or fraudulent node that provides false information for its own benefit can be detected by several approaches [39], [54]. First, comparing data provided by a specific node with that of other nodes in a system could verify whether the data is reliable [54]. A selfish node usually provides data that differs from others [54]. However, the selfish node can collude with other nodes in the system to give false information [54]. Second, checking the data against some predefined rules or based on the background knowledge of a trustor can decrease the risk of communicating with a selfish node [39]. For example, the speed of a car should not be higher than the road's maximum speed limit. Third, checking the consistency of data is another solution to detect selfish nodes. The data must be consistent with previous data from the same

source node [39]. For example, the power consumption of a household must not be too high or too low compared with the previous month in the same season.

2.4.2 Unintentional misbehaviour

Unintentional or accidental behaviour occurs due to faulty or damaged devices, connectivity issues, and software malfunctions. It leads to malfunction or incorrect data sensing [55]. Although this kind of misbehaviour does not harm the system, a defective device can send false or misleading information to it. This kind of misbehaviour is very difficult to detect. Also, distinguishing accidental behaviour from intentional misbehaviour is challenging [49]. A faulty node is often considered to be intentionally misbehaving and loses its trust level [49]. However, false data can be detected using the approaches mentioned earlier, such as comparing data with others and checking data consistency [39], [54].

2.5 Fog computing

2.5.1 Introduction to fog computing

The rapid growth of smart devices and the Internet of Things has resulted in an increase in the amount of data created [56]. Resource limitations, including computation capability, memory and storage, are a major challenge for the Internet of Things devices for processing and storing large volumes of data [57]. The development of cloud computing has evolved with different cloud services to support the limitations of the Internet of Things devices [56]. Although cloud computing is able to provide a large amount of resources to process and store data, there are still many challenges because cloud data centres are typically located far from these devices, in a centralised manner [1]. As a result, cloud computing has issues in terms of network congestion, high response latency and location awareness [1].

Fog computing has emerged to address issues in cloud computing. It is a computing paradigm that was introduced by Cisco Systems [7]. Fog computing is presented as an intermediate layer between cloud computing and end devices. It is considered an extension of the cloud that is located closer in proximity to user devices than the cloud. A fog computing system comprises clusters of devices such as switches and routers, which are provided with computation, storage and networking capabilities, and are

placed at closer physical proximity to user devices. Consequently, it aims to offer services closer to the users with better service quality compared to the cloud in many application scenarios [2].

2.5.2 Fog computing architecture

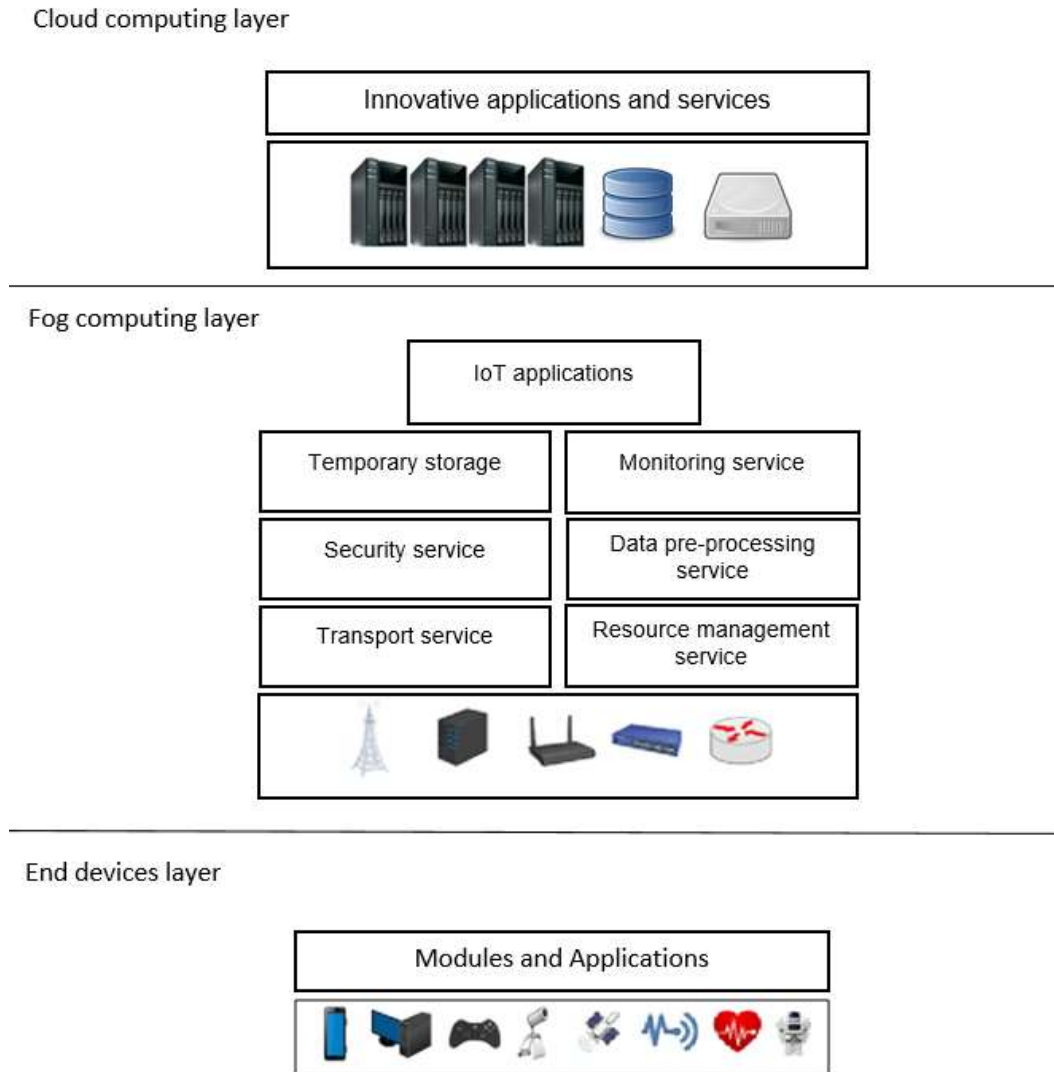


Figure 2.3: Fog computing architecture

The fog computing architecture consists of three layers [7], as shown in Figure 2.3. The bottommost layer contains many kinds of end devices, including smartphones, computers, laptops, and various types of Internet of Things devices. This layer generates a massive amount of data and transmits the data to the upper layer for further processing and storage.

The middle layer is the fog computing layer. It is located above the end device layer. This layer comprises heterogeneous devices, including base stations, routers, switches, mini servers and other kinds of networking components [2]. These devices, called fog nodes, are configured with computing, storage and networking capabilities similar to the cloud. They are placed at closer physical proximity to end devices or where data is created, rather than at a data centre like the cloud [7]. Therefore, fog computing extends cloud computing from the core to the edge of the network. It supports end devices with higher computing power and larger storage.

The fog layer includes the components below [58], [59], [60], [61].

- A monitoring service monitors the performance and status of fog nodes, such as resource usage [58]. It collects this information to support a resource management service.
- A resource management service is responsible for keeping track of the state of available resources provided by the monitoring service. It identifies which cloud or fog node is suitable for hosting an application and allocates fog resources to the application [58]. This service is similar to the resource provisioning service in an architecture proposed by [59].
- A data pre-processing service performs pre-processing, filtering, analysing and trimming of data collected from sensors in the bottommost layer [60].
- Temporary storage provides storage space to store data locally, temporarily from the data pre-processing service or directly from end devices. When the data is uploaded to the cloud, it will be removed from the storage [60].
- A security service provides authentication, authorisation, and cryptography for applications and other services to secure data [59].
- A transport service has the responsibility for uploading the data to the cloud [60]. When a complex task needs more resources, it is offloaded to a cloud data centre by this service. This service may use the security service to secure the data before uploading it to the cloud.

The topmost layer is the cloud computing layer. It is considered a resource-rich layer. It contains a lot of high-end servers, which have powerful CPUs and a large volume of storage [61]. The complex tasks of the fog computing layer can be offloaded for

processing in this layer. Data that requires long-term storage is uploaded to this cloud layer. Furthermore, there are various intelligent applications deployed in this layer to offer to users [61].

2.5.3 Fog computing characteristics

Fog computing provides services including computation and storage near the edge of the network. It has the following characteristics [2], [62], [59], [63].

- **Low latency and real-time interactions**

Fog nodes are deployed near the edge of the network, and they are close to end devices [2]. The distance between end devices and fog nodes is shorter than that to the cloud data centres. So, it can provide low latency and real-time communication that is suitable for latency-sensitive or time-sensitive applications [2]. For example, in AR (Augmented Reality) applications, low network latency is important because users must receive a response in real-time. Because of the location of fog nodes, offloading AR tasks to fog nodes for processing means that results can return to end devices in real-time.

- **Heterogeneity**

In a fog computing environment, there is a variety of types of devices participating as virtual and physical nodes [62]. They comprise high-end servers, edge routers, gateways, access points, and base stations. These devices have various computational and storage capabilities, run different operating systems, and have different software applications [62]. Moreover, they also have various kinds of communication technologies, both wired and wireless, including WIFI, LAN, 4G, Zigbee, LoRa, etc [62]. Fog nodes are able to support the various requirements of IoT applications and work with various IoT devices. For example, a smart city consists of different types of IoT sensors and actuators [64]. Heterogeneous fog nodes can collaborate with them to support many kinds of smart city services, such as smart transportation and smart waste management systems.

- **Reduction of network traffic**

There is a huge number of mobile devices, such as smartphones and IoT devices, sending data over networks to cloud data centres. At the same time, mobile devices also need to download data essential for various applications from the cloud. With fog computing, it is not necessary to send all data to the cloud. Fog computing provides a platform for filtering and analysing the data close to the edge of the network [59]. Fan et al. [65] proposed a web caching system, which is a cache storing web contents for access later, using fog nodes in fog computing as web proxies. The Fog nodes are configured with the capability to provide web cache storage. It helps to save network traffic of an organisation and reduce web service latency. Moreover, the advantage of web cache utilising fog computing is that cache space will increase if there are more devices added to the system.

- **Geographical distribution and location-awareness**

Fog nodes in fog computing can be deployed in a geographical distribution that benefits applications [63]. Based on the distribution, they can support delivering services to end devices in close proximity with low latency and track their current location. For instance, a vehicular ad hoc network (VANET) consists of vehicle nodes connected to other entities by wireless communications [63]. Moving vehicles can exchange information with and offload data to fog nodes that are located at traffic lights, car parking areas, bus stops, etc. Moreover, the fog nodes can determine the physical locations of vehicles in real time for navigation.

2.5.4 Challenges in fog computing

Although fog computing provides benefits to the devices at the edge of the network, it faces several challenges [23], [34], [37], [66], [67], [68], [69], [70] as discussed below.

- **Resource provisioning**

The number of fog nodes in the fog computing environment changes dynamically because fog nodes join and leave the system all the time [66]. The unpredictable change affects the quality of service offered by fog nodes that vary according to the

amount of available resources. Trust management helps an end node to select a fog node which can satisfy its requirements and provide the best quality service [23]. Trust management also helps a fog node to select other available fog nodes for collaborative working [34]. However, trust management is also a task that needs extra resources to be processed [67]. Some devices in the end device layer do not have enough resources to handle a trust management task. They may need other trusted nodes to do the task on their behalf.

- **Task offloading**

In a scenario in which an end device does not have enough resources to complete a task, the task needs to be offloaded to fog nodes [70]. A targeted fog node to outsource the task to might not be well-behaved and may lack the resources and capabilities to meet the task requirements. In the same way, a fog node might be overwhelmed and need to offload some tasks to other fog nodes. Then, it also needs to make sure that the tasks offloaded are handled by trustworthy fog nodes. Trust management provides both the end node and fog node with assistance to decide where they should offload their tasks [68].

- **Data security**

Data generated on end devices is frequently offloaded to fog nodes for processing and storage [70]. A key challenge is ensuring that offloaded data is handled by trustworthy nodes, as data shared with unreliable nodes may be misused. Trust management assists devices in selecting nodes with demonstrated reliability for data sharing and processing. In the same way, the data might be shared with other nodes to work collaboratively or outsourced by a particular node [34]. Cryptography can protect the confidentiality of data in transit, but cannot guarantee that any device holding the data will protect its confidentiality. So, it is necessary to consider which device the data should be shared with. Data shared with untrusted nodes might be disclosed to the public or misused for malicious purposes [34].

- **Developing applications on diverse platforms**

Devices in fog computing can be sensors, routers, and network switches [62]. They certainly have different CPU, memory and storage architectures, and run different

operating systems. The heterogeneous devices in fog computing require building applications to be compatible with those devices [37]. The development of trust management systems also faces the same problem [37]. The diversity of fog platforms requires different trust management versions to be compatible with. Software developers not only have to develop multiple trust management systems but also have to make these versions work synchronously and consistently in their functionality. Moreover, platform diversity also increases the difficulty of maintenance.

- **Authentication**

Authentication is the process of verifying the identity of a user or device [67]. It cannot verify who is trustworthy or behaves well. A user who has been authenticated might access a system for malicious purposes. Authentication works well to keep a system secure in an environment where authenticated users accessing the system are trustworthy and well-behaved [67]. In fog computing systems, fog devices can be devices that belong to different providers [30]. End devices owned by different people might behave differently and dynamically. Fog and end devices may not have come across each other before. Therefore, authentication might not be enough to protect the fog system from malicious devices. Trust management can help to dynamically identify who should be authenticated, ensure the trustworthiness of devices, detect malicious devices, and isolate misbehaving devices from the fog system.

- **Secure communication**

Data in transit between end devices and fog nodes may be subject to eavesdropping or interception [69]. An attacker can eavesdrop on communications between devices over both wired and wireless networks. Trust management cannot help to prevent attackers from listening in. However, in scenarios where data needs to pass through intermediate nodes to get to its destination, trust management can help to decide which intermediate nodes to transfer the data [69]. In an ad-hoc network that consists of devices communicating with each other directly, trust management can help to decide which device can be trusted to share data with.

2.6 Systematic literature review

This section describes the systematic review methodology used to analyse existing literature on trust management in fog computing systems. The review follows the systematic literature review (SLR) guidelines [71].

2.6.1 Research questions

The following four research questions were formulated to guide the literature search.

- **SLRQ1:** What trust management approaches have been proposed for fog computing and IoT environments, and how are they evaluated?
- **SLRQ2:** What trust relationships, trust aspects and trust computation models are considered in existing fog computing trust management approaches?
- **SLRQ3:** What trust management requirements can be derived from fog computing application scenarios?
- **SLRQ4:** What limitations of existing task offloading approaches in fog computing?

2.6.2 Search strategy

The search strategy was developed according to systematic review guidelines [71] for identifying all relevant literature. The search strings were derived from the four research questions and the main concepts of this study. Key terms include trust management, fog computing, Internet of Things, task offloading, smart home, smart traffic, and e-health. Synonyms and alternative terms were also added, such as “IoT” for “Internet of Things” and “edge computing” for “fog computing”. These terms were combined using Boolean operators, with OR for similar terms and AND to join different concepts. The resulting search strings are as follows:

- "trust management" AND ("Internet of Things" OR "IoT")
- "trust management" AND ("edge computing" OR "fog computing")
- "fog computing" OR "fog-based computing" AND ("smart traffic" OR "smart home" OR "home automation" OR "e-health" OR "smart healthcare")
- "fog computing" AND "task offloading"

Table 2.1 shows how each search string maps to the SLR research questions it is intended to address.

Table 2.1: Mapping of search strings to SLR research questions

Search string	SLRQ1	SLRQ2	SLRQ3	SLRQ4
trust management AND (IoT OR Internet of Things)	✓	✓	✓	
trust management AND (fog computing OR edge computing)	✓	✓	✓	
(fog computing OR fog-based computing) AND (smart traffic OR smart home OR home automation OR e-health OR smart healthcare)		✓	✓	
fog computing AND task offloading				✓

The search terms were conducted across six electronic research databases to provide broad coverage of the computer science, engineering, and information technology domains, including IEEE Xplore, ACM Digital Library, ScienceDirect, SpringerLink, Google Scholar and arXiv. The search was restricted to publications from 2000 to 2025 in order to focus on current developments in this research area. However, some older publications were included to provide background on traditional cloud computing and fog computing systems.

2.6.3 Inclusion and exclusion criteria

Inclusion and exclusion criteria were defined to ensure that only relevant and high-quality studies were included in the analysis.

Inclusion criteria include:

- 1) published in peer-reviewed journals or conference proceedings between 2000 and 2025.
- 2) focused on fog computing, edge computing, or Internet of Things systems.
- 3) focused on task offloading in fog computing.
- 4) focused on the utilisation of fog computing in application domains including smart traffic management, e-health, and smart home environments.
- 5) written in English language.

Exclusion criteria include:

- 1) published before 2000 or non-peer-reviewed publications such as magazines or website articles.
- 2) focused only on cloud computing without fog or edge layer considerations, or addressed only high-resource devices like servers without resource constraints.
- 3) covered general security mechanisms, authentication protocols, or other systems without trust management components involved.
- 4) Presented purely theoretical models without an implementation context.

2.6.4 Study selection

The study selection process was carried out in four sequential stages. The PRISMA diagram illustrating the study selection process is presented in Figure 2.4.

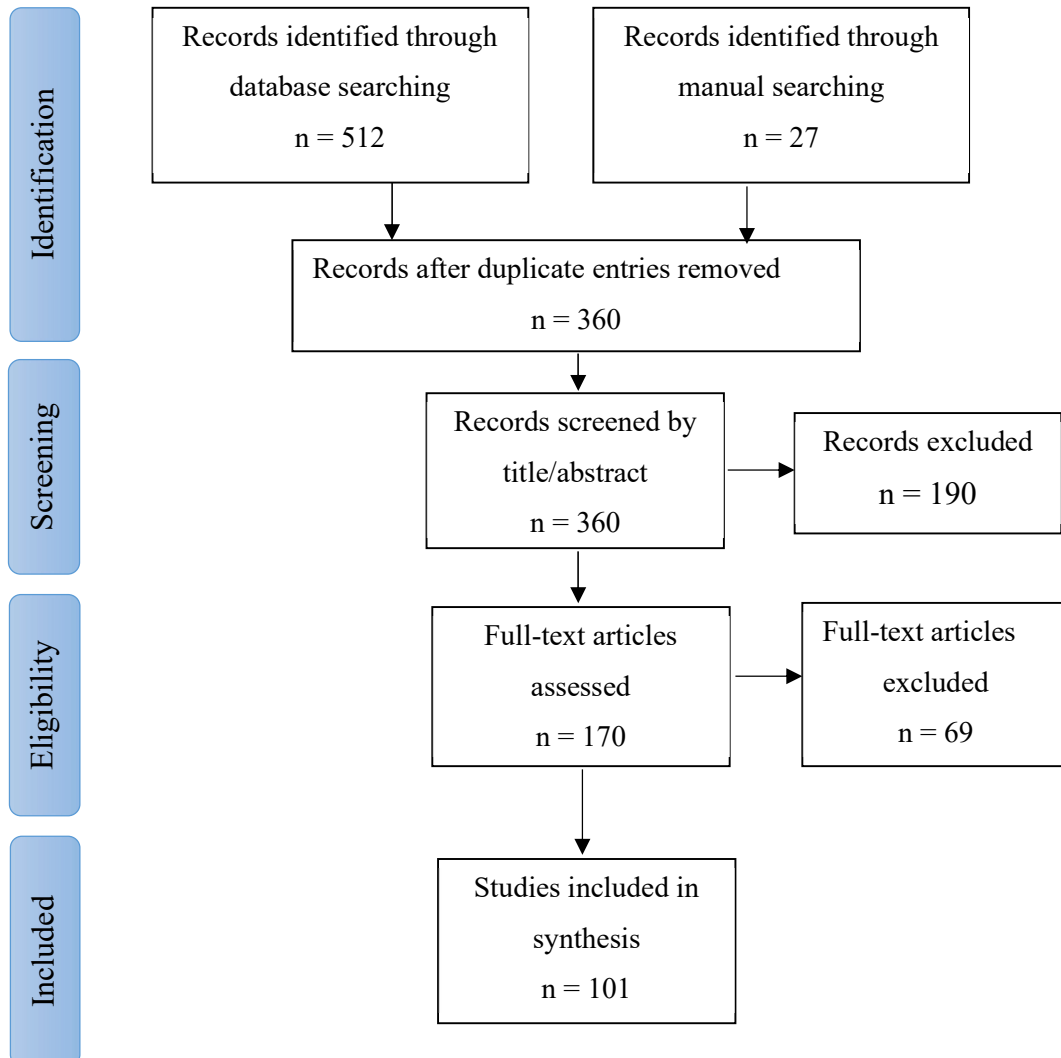


Figure 2.4: PRISMA diagram

Table 2.2: The breakdown of search terms and their results

Search Term	No of Papers
"trust management" AND ("Internet of Things" OR "IoT")	313
"trust management" AND ("edge computing" OR "fog computing")	40
"fog computing" AND ("smart traffic" OR "smart home" OR "home automation" OR "e-health" OR "smart healthcare")	71
"trust management" AND "task offloading"	88

The initial database search across the six databases provided a total of 512 records, distributed across the four search strings as shown in Table 2.2. An additional 27 were identified through manual searching, by tracking citations and checking the reference list. After 179 duplicate entries were removed, a total of 360 articles were screened by title and abstract against the inclusion and exclusion criteria. This screening process eliminated 190 articles. The full texts of 170 remaining studies were then retrieved and assessed in detail against all criteria. Studies that did not fall within the defined research scope were excluded at this stage. The final studies included in this literature review comprised 101 articles.

2.7 Trust management in cloud computing and the Internet of Things

Cloud computing offers customers a wide range of services, particularly data storage and processing power [72]. Because the services are delivered online, there are several concerns associated with cloud computing, including privacy, security, availability, and data leakage. [73]. Confidentiality is at risk due to the outsourcing of the data. Even though encryption is used to maintain confidentiality, it remains challenging to detect when data has been altered in transit. Cloud computing also lacks guarantees for latency and bandwidth [72]. A major reason for low bandwidth and high latency is the physical distance between cloud data centres and consumer devices. Consumers are also particularly concerned about the availability of cloud services and resources. [72]. Availability refers not only to the cloud service's reachability but also to the transaction success rate. Even though most cloud service providers guarantee 99.99%

server availability, multiple outage situations have already been reported [72]. Trust and reputation systems can be utilised to assist users in selecting reliable cloud providers. Several trust and reputation methodologies have been proposed to evaluate cloud service providers based on the quality of their services and security measures [74], [75], [76].

Trust management helps in selecting a trusted cloud service provider in several contexts, such as availability and service quality [77]. CloudArmor [74] is a reputation-based trust management framework that deals with the trust of cloud services and allows cloud users to effectively distinguish trustworthy cloud services. A key strength of CloudArmor is its credibility model, which uses several factors to detect collusive and misleading feedback to enhance the reliability of reputation assessment. It also includes an availability model that considers several criteria, such as operating power, to assist cloud instance selection. However, its effectiveness depends on the quality of the collected feedback, and its reputation calculations may become less reliable in highly dynamic or large-scale cloud environments because feedback can vary widely. Furthermore, the credibility model depends on sufficient historical feedback data, making it less effective for newly registered cloud providers with limited interaction records.

Trust has also been incorporated into access control decisions to decide whether to allow access to particular resources [75]. Ghafoorian et al. [75] proposed a trust and reputation-based access control (RBAC) model for secure data storage in the cloud. The model addresses a key limitation of conventional RBAC that only verifies user identities and cannot detect malicious or abnormal behaviour. It dynamically adjusts user access privileges based on their trust level. Users whose trust level drops below a defined threshold are automatically revoked their access rights, reducing the risk of insider threats and unauthorised data manipulation. Yan et al. [76] also proposed a flexible data access control scheme for cloud storage based on trust and reputation with ciphertext policy attribute-based encryption (CP-ABE). The scheme allows data owners to define fine-grained access policies based on user attributes and trust scores. This approach is more flexible than the earlier RBAC-based model because the RBAC-based model assigns permissions based on predefined roles and cannot

differentiate between two users sharing the same role. However, CP-ABE incurs a high computational cost, particularly when a large number of attributes are involved.

With respect to trust in the Internet of Things, sensor nodes are typically not tamper-resistant, making them highly vulnerable to compromise [16]. Trust management and reputation systems can help avoid communication with malicious devices and differentiate trusted sensor nodes. It can also be used to estimate how much credibility each node has for the purpose of collaboration [78]. Ganeriwal et al. [78] developed a reputation-based framework for sensor network systems. Each sensor node in a sensor network system maintains the reputation of other nodes by monitoring their behaviour and assigning them a cooperative or non-cooperative rating. This approach provides simplicity and a distributed architecture for maintaining reputation records without requiring a central authority. The reputation values are also used to predict future behaviour of nodes, specifically to detect other nodes that are behaving inappropriately and to assess the trustworthiness of the information they offer. Boustanifar et al. [18] proposed a trust management system for mobile machine-to-machine (M2M) communications in IoT to prevent communication with malicious or selfish nodes. Similar to the previous approach, this system leverages past interactions to estimate the trustworthiness of other nodes. Trust values are used to decide whether to execute main tasks locally or delegate them to neighbouring entities. However, both approaches are vulnerable to trust-related attacks such as bad-mouthing and ballot-stuffing. Malicious nodes may attempt to inflate or deflate the reputation of others to manipulate the system. Furthermore, reputation is calculated based on direct observations and past interactions. Newly joined nodes with little interaction history cannot be accurately evaluated.

Blockchain technology has been applied to manage data trustworthiness in the Internet of Things domain [79], [80]. The reliability of the system depends not only on how trust is computed but also on how trust-related information is stored, shared, and protected from manipulation [79]. Blockchain is relevant to trust management because its decentralised and immutable nature helps preserve the integrity, transparency, and authenticity of trust records in IoT environments. Moreover, it offers a promising approach to data storage by addressing critical security and integrity challenges with

traditional centralised systems [79]. By utilising a decentralised immutable ledger, it provides a secure environment that enhances trust and authenticity [79]. The authors in [79] leveraged blockchain technology to secure trust management information in their trust management mechanism in IoT environments. Trust among IoT devices is assessed using federated learning. Reputation ratings results are stored on an immutable blockchain to ensure data authenticity and accuracy. Federated learning and blockchain create significant computational and communication overhead, which may not be well-suited to IoT devices with limited processing power and battery capacity for maintaining the blockchain. Arul et al. [80] proposed a blockchain-based framework for the Internet of Medical Things (IoMT) system to improve the secure sharing of healthcare data. The framework protects patient privacy and supports secure data access. It allows only authenticated healthcare providers to receive alerts and patient information through a healthcare application network. A main disadvantage, similar to the previous approach, is that blockchain-based healthcare systems require high computing power and implementation costs. It may be difficult to integrate with existing healthcare systems.

2.8 Task offloading protocols in fog computing

Task offloading in fog computing refers to the transfer of computationally intensive tasks from resource-constrained devices to more capable nodes [81]. The primary objectives are to reduce task completion latency, improve processing response time, and lower energy consumption at the network edge. Shahryari et al. [81] proposed an energy-efficient and delay-aware computation offloading scheme for IoT-fog-cloud networks. The motivation is that resource-constrained IoT devices struggle to handle computation-intensive and delay-sensitive tasks locally. These tasks can be executed locally, offloaded to nearby fog nodes, or sent to the cloud, depending on task requirements and conditions. Their scheme minimises energy consumption under delay constraints on end nodes and reduces task waiting times through a distributed task-sharing algorithm among fog nodes. However, the model assumes each IoT device handles only one task at a time, which does not reflect the nature of real IoT systems, where devices often process multiple concurrent tasks simultaneously.

Singh and Singh [82] proposed an algorithm based on Levy-flight moth flame optimisation (LMFO) to determine the optimal allocation of tasks across local IoT devices, fog servers, and the cloud. The work demonstrates significant reductions in energy consumption and task processing time compared to baseline schemes. However, the offloading decision is made by a centralised algorithm, which can lead to a single point of failure. Aleteemat and Shurman [83] proposed an energy-efficient task offloading strategy (EETOS) to allocate tasks among local IoT devices, fog servers, and the cloud for reducing task completion time and processing cost. Computation-intensive IoT tasks are offloaded to fog nodes and assigned through a fog broker using a hybrid genetic algorithm. An advantage of the approach is that it balances energy efficiency and delay performance, which makes it suitable for battery-powered IoT sensors. However, the approach relies on a central fog broker for task scheduling, which introduces a single point of failure and raises scalability concerns when the number of IoT devices increases.

Yadav and Sangwan [84] proposed a dynamic offloading framework in which a fuzzy logic engine makes offloading decisions based on CPU usage, delay sensitivity, residual energy, task size, and available bandwidth. Time-sensitive tasks are offloaded to nearby fog nodes, while resource-intensive tasks are directed to the cloud. The key strength of this approach is that offloading decisions are made dynamically based on current resource and network conditions, rather than relying on a static configuration. Alenizi and Rana [85] proposed an offloading scheme for delay-sensitive IoT tasks in fog environments. The scheme determines whether a task should be processed locally or offloaded to neighbouring fog nodes. Two algorithms, dynamic task scheduling (DTS) and dynamic energy control (DEC), are employed to dynamically adjust the offloading threshold, thereby reducing overall delay, improving throughput, and minimising energy consumption. However, the threshold adjustment is managed by a single fog controller rather than allowing individual IoT devices to discover and select the most appropriate delegate nodes in a distributed manner. This centralised design can create a communication bottleneck and increase latency, which is problematic for latency-sensitive IoT sensor applications. Vakilian et al. [86] proposed a distributed workload-offloading scheme for fog computing. Fog

nodes can collaborate with neighbouring fog nodes to offload tasks among themselves and to the cloud in order to minimise response time, energy cost, and cloud rental cost. The scheme employs a multi-objective optimisation technique using the distributed alternating direction method of multipliers (ADMM) algorithm to guide offloading decisions across fog nodes. However, this advanced technique may be less suitable for highly dynamic environments because the optimisation process becomes computationally expensive when workload conditions or fog node availability change frequently, leading to high delays in offloading decisions.

The reviewed studies indicate that task-offloading schemes in fog computing mainly focus on reducing delay, improving energy efficiency, and resource utilisation by offloading computation from IoT devices to fog or cloud resources. The reviewed studies differ in that some use optimisation algorithms, others adopt dynamic threshold or fuzzy-logic control for offloading decisions. Cooperative offloading approaches were also introduced to improve resource utilisation by sharing workloads across fog nodes. Overall, the main advantages of task offloading protocols in fog computing are reduced delay and lower energy consumption at constrained devices. However, their main weaknesses include the use of centralised control in some schemes, which introduces a single point of failure, and limited consideration of the trustworthiness of destination nodes to which tasks are offloaded.

2.9 Trust management in fog computing

Several trust management approaches have been proposed for fog computing environments. This section analyses and classifies existing trust management works in the area of fog computing according to trust relationships, trust composition, trust propagation, trust aggregation and formation, and their research evaluation approaches, as shown in Table 2.3, Table 2.4, Table 2.5 and Table 2.6.

2.9.1 Trust relationships

The objective of trust management in fog computing is to identify bad fog nodes, end nodes and cloud nodes [34]. Existing trust management approaches take into account the trust relationships among fog nodes, end nodes, and cloud data centres, as shown in Table 2.3. In [3], [23], [91], [92], [93], they considered the trust of end nodes

towards fog nodes. An end device with limited resources offloads its tasks to fog nodes based on their trustworthiness. A bad fog node may provide poor quality of service or behave maliciously [23]. These approaches help the end node to select a fog node with the highest trust value to outsource tasks. The authors [4], [26], [89], [90], [91] proposed trust management for dealing with the trust of fog nodes towards end nodes. In this scenario, once end nodes get connected to the fog nodes for services, they may perform different types of attacks, such as Denial-of-Service (DoS) or hide malicious scripts or harmful code in offloaded data with damaging effects on the hardware and software of fog nodes [4]. Trust management verifies the trustworthiness of end nodes to prevent malicious access to fog resources. In contrast, the work of [28], [33] pointed out that trust should be considered in a two-way direction, end nodes towards fog nodes and vice versa, and they presented trust management that allows end nodes and fog nodes to estimate the trustworthiness of each other in fog computing systems.

Trust among end nodes has also been considered in [5], [92], [93], [94], [95]. An end node sharing data or working collaboratively with other end nodes needs to ensure that those nodes are trustworthy. In the same way, fog nodes may share their resources with each other and work collaboratively. Offloading tasks from one fog node to another fog node is necessary when the fog node is overwhelmed with tasks. The authors in [34], [6], [96], [97], [98], [99], [100] proposed trust management to allow a fog node to select another trusted fog node to collaborate with and handle task overload. However, Ramamurthy et al. [97] considered a bi-directional trust relationship between a fog client and a fog server. The fog client expects a trustworthy service and reliable data sharing from the fog server, while the fog server needs to provide the service and share data with a trustworthy fog client. Moreover, Wang et al. [95] have considered trust between cloud nodes and end nodes to ensure that cloud nodes meet the requirements of end nodes and that end nodes are well-behaved before they access cloud services. In summary, existing trust management approaches have considered multiple trust relationships across all three layers of fog computing systems, which cover trust management in different relationships within fog systems.

2.9.2 Trust composition

Various studies have used different trust aspects and combined multiple trust metrics to improve the effectiveness of trust estimation in trust models, as shown in Table 2.5. The authors [87] and [3] used reputation score and rater's credibility as trust metrics in their proposed reputation-based trust management system. The authors [5], [6], [21], [25], [91], [94] considered only QoS metrics such as throughput, response time, etc, while Alemneh et al. [28], Almas et al. [98] and Ramamurthy et al. [97] relied on QoS and social trust metrics. For example, in the [28] approach, QoS metrics, namely latency and Packet Delivery Ratio (PDR), were used to evaluate the trustworthiness of fog nodes while social trust metrics, including friendship, honesty and ownership, were used to reflect the trustworthiness of end nodes. The research works [10], [21], [30], [34] utilised QoS with other specific metrics that were defined depending on their perspectives. For instance, the authors in [23] and [88] considered the distance between an end node and fog nodes together with QoS to evaluate trust. The assumption is that closer fog nodes are more desirable for the end node. Junejo et al. [30] took into account QoS with device specification, resource usage and energy consumption in their trust model, while Wang et al. [95] used their own specific metrics with QoS. The authors of [93], [55], [34] relied on their own specific metrics to estimate trust. Several specific metrics were introduced. For example, Experience Satisfaction (ES), introduced by [34], is computed based on QoS and Quality of Protection (QoP). In this research, QoS was defined as the ability to deal with requests with maximum bandwidth and minimal latency, while QoP was the degree to which a node protects data during processing and transferring to others. Liang et al. [93] introduced service quality ratings as a factor of trust calculation that derives from packet size, forwarding delay and route success rate, while Soleymani et al. [78] proposed specific methods to measure trust metrics, including experience level, plausibility level and accuracy level for their trust model.

Various studies have combined multiple trust properties to improve the effectiveness of trust evaluation in their trust models. However, the existing literature remains inconsistent in how these metrics are selected and used. Many models are also specifically designed for a specific context and are difficult to apply in other

environments. Furthermore, using multiple metrics can improve trust evaluation accuracy, but many studies do not provide a clear justification for why those particular metrics are the most representative of trustworthiness. Another limitation is that many existing approaches pay little attention to the environmental and contextual factors that can affect node behaviour.

2.9.3 Trust propagation

The architecture of trust propagation found in the literature is centralised, decentralised, and distributed, as shown in Table 2.3. A centralised model was used in [5], [21], [88], [91]. These methods have a fog node acting as a centralised node responsible for information gathering and trust evaluation on behalf of a trustor. The centralised fog node may monitor trust information itself [21], [88] or receive trust information from other nodes [5]. The centralised fog node then computes and stores the trust values of all nodes in a system. A device that needs the trust value of a particular node can access trust values from the centralised node. Moreover, another centralised model in [91] relied on trust agents, which act as an intermediary node communicating with both a trustor and trustee for trust management. Trust management in a decentralised manner was proposed in [28], [29], [38], [42], [43], [44], [39], [96], [97], [98], [99]. These approaches allow every node to maintain trust information and perform trust calculation locally. A node can rely on only direct trust [23], [6] or a combination of direct trust and indirect trust [11], [27], [29], [98]. Besides, Wang et al. [95] used decentralised and centralised architectures for trust management between end nodes and between cloud nodes and end nodes, respectively.

In addition to the first two methods, a distributed architecture for trust management has also been proposed in existing research. The authors [55], [93], [94] divided end nodes into clusters, and each end node maintains trust information and computes the direct trust locally, similar to the decentralised architecture. A fog node in each cluster acts as a centralised server responsible for accumulating feedback from end nodes and generating a reputation value for a specific node. Finally, each end node performs an overall trust by combining direct trust and the reputation value from the fog node in each cluster. The authors [4], [89] introduced a cluster head within a fog cluster to

handle trust calculation and filter end nodes based on their trust values before they access any fog node in the cluster. Besides, Junejo et al. [33] proposed a two-way direction of trust management between fog nodes and end nodes, divided fog nodes and end nodes into multiple clusters, and each cluster has a centralised fog node acting as an intermediate node to manage trust between them. The centralised fog node monitors trust information, performs trust calculation, and stores the trust values of each fog node and end node in its database. Any node can access the centralised fog node with trust values of other nodes.

Blockchain technology has been adopted for trust management in fog computing systems. Blockchain allows nodes participating in the blockchain system to maintain a chain of data that is consistent across all participants. Thus, it provides transparency and traceability to the data. Fan et al. [92] used a blockchain hosted on fog nodes to store trust values of end nodes. When an end node needs the trust value of a particular node, it contacts a nearby fog node to obtain it from the blockchain. Each end node collects information and calculates trust when it interacts with another node, and then sends the trust value to the blockchain. However, blockchain requires fog nodes to have sufficient computing power to run it. Therefore, the authors in [87], [25], [3] utilised an Ethereum blockchain that is deployed on cloud computing to hold trust information and handle trust computation. This eases the capability requirements of fog nodes to deploy a blockchain system. However, high latency could arise due to the distance to the cloud.

Trust propagation architectures in fog computing systems, centralised, decentralised, distributed, or blockchain-based, provide distinct advantages and limitations for trust management. Centralised approaches are straightforward to manage, but they can become bottlenecks, suffer from limited scalability, and introduce a single point of failure. Decentralised approaches remove the limitations of a central node and are more suitable for dynamic fog environments, but they place additional computational, storage, and energy burdens on specific nodes. Distributed approaches address this concern by allowing each node to maintain trust locally and share it with others. However, this may introduce inconsistent trust observations and increase computational overhead on constrained nodes. Cluster-based approaches that place the

workload on cluster heads may suffer additional overhead for cooperation between the cluster heads. Blockchain-based approaches provide transparency, traceability, and tamper resistance, but they require significant computation and communication costs.

2.9.4 Trust aggregation and formation

As shown in Table 2.3, a couple of research works [6], [23] take into account only direct trust in their trust models, while [3], [21], [25], [33], [87], [88], [97] consider only indirect trust, and some research [55] [28], [34], [91], [92], [93], [94], [95], [98] combine direct and indirect trust. With respect to their trust model in Table 2.4, the authors of [6], [21], [34], [91], [93], [94], [95], [99] proposed specific mathematical models to be used for trust evaluation. The trust models in [55], [88] used fuzzy logic with a trust level of each trust aspect combined based on fuzzy logic. A threshold is assigned to each of the trust metrics for trust formation in [23], [88], while [55] used a centroid defuzzification technique, which is a method to determine a single number from the output of a fuzzy set, to formulate an overall trust value. The trust models in [87], [3] use reputation scores to compute the trust level of a trustee based on the history of interactions, and use credibility as the weight to aggregate reputation scores provided by raters.

Furthermore, some research uses artificial intelligence to calculate trust. Authors in [5] adopted multiple linear regression in their approach by formulating trust evaluation as a multiple linear regression problem to establish a trust model. The idea of subjective logic, which is a kind of probabilistic logic for subjective opinions that takes into account the uncertainty and reliability of sources, was adopted in [28] to aggregate recommendations from multiple recommenders for indirect trust. The research [97] utilised logistic regression and subjective logic to aggregate the recommendations of other fog nodes in proximity in a bi-directional trust management system to evaluate the trustworthiness of a fog server and a fog client, respectively. Lastly, random forest regression, a machine learning method for classification and regression, was used in [33] to predict the trust of a fog node based on the QoS parameters. Kochovski [25] ranked fog nodes by a specific method called a Markov model-based ranking mechanism for fog node selection. In [4], [26], [89], [90], [92],

[96] approaches did not propose trust models for trust evaluation, but they focused on how trust values are managed in a system.

Existing research has utilised several techniques for trust aggregation and formation methods. Mathematical trust models provide low memory and processing requirements for IoT and fog nodes with limited CPU and RAM. However, fixed mathematical models lack adaptability in highly dynamic fog environments. Fuzzy logic-based trust models can effectively evaluate uncertain and imprecise trust information. However, the models need to configure membership functions and trust decision thresholds manually, leading to biases and inconsistencies in trust management evaluations. Machine learning and artificial intelligence models provide high trust evaluation accuracy, but they require significant computational resources. Consequently, future research should focus on developing trust models that balance computational accuracy and resource efficiency to operate in diverse and dynamic fog computing environments.

2.9.5 Evaluation approach

Most researchers evaluated their approaches by simulation, as shown in Table 2.6. The authors in [23], [5], [34], [88], [94], [95] did not implement the entire trust management architecture, but they focused on proposing mathematical trust models. So, they prefer to evaluate their models in MATLAB because it provides essential tools for evaluating trust models. The research in [91] used MATLAB and other tools for experimental simulation. Some research used well-known simulators, namely Network Simulator 2 or 3 (ns2 or ns3) [55], [96], [99], FogNetSim++ [97], Cooja [98], and iFogsim [4], [33], [6], [100], to simulate fog computing environments. However, these simulators may require extensive modifications to support trust management research as Network Simulator 2 or 3 was designed specifically for research in computer communication networks, and iFogsim was usually used to simulate resource management in fog computing systems.

The authors [28], [93] used JAVA and [21] used System C to build simulators themselves. Although they have to create their own simulators from scratch, it may take less time than modifying ns2 and iFogsim to support fog computing research.

Neither was designed to represent the complexity of trust management behaviour, such as the heterogeneity of device capabilities, energy constraints, and dynamic trust interactions. As a result, significant modifications are required to simulate trust management scenarios, and it is not always clear whether these modifications faithfully represent real-world environments. Moreover, the research that has utilised blockchain [3], [25], [87] implemented and evaluated their proposed mechanism with a real blockchain hosted on the cloud.

Most existing evaluation methods rely on simulation rather than real-world deployment. This raises questions about how accurately the simulations reflect actual fog computing environments. Well-known simulators, such as ns3 and iFogSim, were not designed for trust management simulation and therefore require modifications to represent trust management interactions. Similarly, evaluating trust management approaches using mathematical tools such as MATLAB does not represent the operational characteristics of real fog environments. Although simulators such as iFogSim and FogNetSim++ provide useful support for evaluating resource management and network performance in fog environments, they were not designed to simulate trust management interactions. Both simulation tools require modification to simulate trust evidence collection, trust computation, trust decision, and node behaviour.

2.10 Research gaps

Despite extensive research on trust management in fog computing systems, there remains a significant gap in adequately addressing the heterogeneity and resource constraints of participating entities. Existing trust management frameworks in fog computing assume that all devices are homogeneous with similar computational, memory, storage, and networking capabilities. They typically implement a single architecture for all devices, regardless of the consideration of their individual capabilities. However, real-world fog computing environments comprise highly diverse devices, ranging from high-end servers to resource-constrained IoT devices, each with different computational power, memory and storage capacity, and energy supply [2]. Alwakeel et al. [101] also emphasise that edge and fog computing settings contain a wide range of heterogeneous devices, all varying in their resources.

Additionally, there is limited focused research on how resource-constrained devices can effectively participate in trust management when they lack the resources to perform complex trust evaluations. Existing approaches overlook scenarios where many IoT devices have resource limitations and are unable to execute sophisticated trust algorithms locally [102]. The complexity of trust management approaches exacerbates this problem. Many trust management approaches employ computationally intensive mathematical models and advanced techniques, such as fuzzy logic, multiple linear regression, and subjective logic, to perform trust evaluations in fog computing [5], [28], [23]. These sophisticated trust models require significant computing and memory resources, which may not be suitable for resource-limited devices. For instance, Wang et al. [5] demonstrated that multiple linear regression techniques pose significant computation challenges for resource-limited IoT devices.

Furthermore, current trust architectures lack the flexibility to make use of available resources in fog systems, resulting in inefficient resource utilisation across heterogeneous devices and failing to leverage high-capability devices within the system for handling trust management tasks. High-capability nodes are underused, while constrained devices may be overwhelmed. These approaches not only underutilise available resources but also create potential risks to the whole fog system because resource-limited devices are vulnerable to compromise. While task-offloading schemes in fog computing in the previous section demonstrate that transferring computation from resource-constrained IoT devices to capable fog nodes is both feasible and beneficial, these works do not address the specific requirements that arise when the offloaded workload is trust management. These gaps motivate the design of the trust management delegation protocol.

2.11 Conclusion

This chapter summarises the background knowledge of fog computing and the concept of trust. Existing trust management research in the area of fog computing was discussed, focusing on several aspects. Based on an analysis of existing research, most studies concentrate on proposing trust models that effectively estimate the trustworthiness of a device. These models utilise complex mathematical formulas and

advanced techniques such as multiple linear regression and subjective logic. However, these approaches require significant computing power to handle the complexity of the models. Furthermore, several trust properties, such as QoS and social trust metrics, have been proposed to reflect the device's capability and behaviour.

Table 2.3: Comparison of trust management approaches in fog computing

Research	Trustor & Trustee	Trust context	Direct or indirect	Architecture	Trust calculation
[3], [87], [25]	End nodes towards fog nodes	Task offloading	Indirect	Decentralised	Blockchain
[23]	End nodes towards fog nodes	Task offloading	Direct	Decentralised	End nodes
[88], [21]	End nodes towards fog nodes	Task offloading	Indirect	Centralised	A fog node
[92]	Between end nodes	Collaborative working	N/A	Decentralised	End nodes
[93], [94], [55]	Between end nodes	Collaborative working	Direct and indirect	Distributed	End nodes, but fog nodes compute indirect trust
[5]	Between end nodes	Collaborative working	Indirect	Centralised	A fog node
[95]	Between end nodes	Collaborative working	Direct and indirect	Decentralised	End nodes
	Between cloud nodes and end nodes	Task offloading	Indirect	Centralised	A fog node
[4], [89]	Fog nodes towards end nodes	Malicious & misbehaved node detection	N/A	Distributed	Fog nodes
[90], [26]	Fog nodes towards end nodes	Malicious & misbehaved node detection	N/A	Decentralised	Fog nodes
[91]	Fog nodes towards end nodes	Malicious & misbehaved node detection	Direct and indirect	Decentralised	An intermediary trust agent

Research	Trustor & Trustee	Trust context	Direct or indirect	Architecture	Trust calculation
[34], [97], [99], [100]	Between fog nodes	Collaborative working and task offloading	Direct and indirect	Decentralised	Fog nodes
[40]	Two-way direction between fog nodes	Trust of a fog service provider for trustworthy service and secure data sharing	Indirect	Decentralised	Fog nodes
		Trust of a fog service requester for trustworthy interaction			
[6]	Between fog nodes	Collaborative working and task offloading	Direct	Decentralised	Fog nodes
[96]	Between fog nodes	Collaborative working and task offloading	N/A	Decentralised	Fog nodes
[33]	Two-way direction between fog nodes and end nodes	Trust of fog nodes for task offloading	Indirect	Distributed	Fog nodes
		Trust of end nodes to prevent malicious end nodes			
[28]	Two-way direction between fog nodes and end nodes	Trust of fog nodes for task offloading	Direct and indirect	Decentralised	End nodes
		Trust of end nodes to prevent malicious end nodes			Fog nodes

Table 2.4: Comparison of trust models for trust management approaches in fog computing

Research	Trust model	Description
[3], [87]	Reputation model	Trust value is computed based on reputation scores and rater's credibility
[23], [88], [55]	Fuzzy logic	Trust value is computed based on fuzzy logic
[34], [21], [93], [94], [95], [98]	Specific formula with weighted sum	An overall trust value is computed by combining trust metrics with weighting factors
[6], [91], [99]	Specific formula	Trust metrics are combined by specific formulas
[28]	Specific formula and subjective logic	Direct trust is calculated by combining trust metrics with weighting factors. Indirect trust is done by subjective logic. They are combined with weights to be an overall trust value.
[97], [100]	Logistic regression and subjective logic	The recommendations of other nearby nodes are aggregated by logistic regression and subjective logic to evaluate trust towards a fog service provider and a fog service requester respectively.
[33]	Random forest regression and specific formulas	Trust of fog nodes is predicted based on Random Forest regression. An overall trust value is computed by combining trust metrics with weighting factors.
[5]	Multiple linear regression	A trust value is computed based on a multiple linear regression model
[25]	Specific method	Fog nodes are ranked by using a Markov model-based ranking mechanism.
[4], [89], [92], [96], [90], [26]	Not provided	-

Table 2.5: Comparison of trust aspects for trust management approaches in fog computing

Research	Trust aspects	Trust metrics	Determination of the trust metrics
[3], [87]	Reputation score	Reputation scores and rater's credibility	Reputation scores: 1 = "satisfactory transaction" and 0 = "non-satisfactory" Rater's credibility: comparing rating value with the majority
[6]	QoS	Job completion rate and job failure rate	Job completion rate: the number of successful jobs divided by the total number of jobs Job failure rate: the number of failed jobs divided by the total number of jobs
[94]	QoS	Packet loss rate, route failure rate and processing delay	Packet loss rate: the ratio of the number of packets lost to the total data packets Route failure rate: the ratio of the number of routing packets discarded by recipients to the total routing packets sent by senders Processing delay: the time interval from receiving data to forwarding data for relay nodes
[25]	QoS	Availability, response time and throughput	Availability: the probability of the node's correct functioning at a specific moment in time Response time: round-trip time for the package to reach a destination and return to a source Throughput: the amount of data that can be successfully transferred per unit of time
[5]	QoS	Package delivery ratio, response time and average energy consumption	Not provided

Research	Trust aspects	Trust metrics	Determination of the trust metrics
[21]	QoS	Availability, reliability, data integrity, and turnaround efficiency.	Availability: the number of tasks accepted by a fog node divided by the total number of tasks submitted to that node Reliability: the number of successful tasks divided by the total number of tasks Data integrity: the number of tasks that a fog node successfully preserves data integrity, divided by the total number of successful tasks. Turnaround efficiency: the promised turnaround time for the completion of a task divided by the actual turnaround time for the task
[91]	QoS	availability, honesty, and cooperativeness	Cooperativeness is the ratio of prompt responses to the total number of all responses. The authors did not provide the determination of honesty and cooperativeness.
[28]	QoS and social trust	latency, Packet Delivery Ratio (PDR), friendship, honesty and ownership	Friendship: the ratio of the number of successful connection requests of a client to the maximum number of connection requests. Honesty: is calculated by counting the number of suspicious dishonest experiences of a trustee as observed by a trustor node during a time period Ownership: if a node encounters another node owned by the same person, the trust value is set to 1; otherwise, it is 0. Latency: time required by a fog node to provide a service to a fog client Packet Delivery Ratio (PDR): the ratio of packets successfully received to the total packets sent

Research	Trust aspects	Trust metrics	Determination of the trust metrics
[98]	QoS and social trust	Response time and similarity	Response time is the time taken to send a request and receive its response. Similarity is calculated based on server similarity, social contact similarity, and service Similarity.
[23]	QoS and other trust metrics	Distance, Latency, Reliability	Distance: distance between the user and each of the fog nodes in meters Latency: time interval between stimulation and response in milliseconds Reliability: is measured based on the load of the fog nodes
[88]	QoS and other trust metrics	Distance, Latency, Reliability, security, and feedback	Distance and Latency: the same as the previous work Reliability: is computed based on the number of users who experienced a failure in a time interval and promised mean time to failure Security: is computed based on confidentiality and authentication by fuzzy rules. The value ranges from high, medium to low Feedback: is ratings from users upon the completion of service. The value ranges from high, medium to low
[95]	QoS and other trust metrics	packet loss rate, route failure rate, processing delay and other specific metrics	Packet loss rate: the ratio of the number of packets lost to the total data packets Route failure rate: the ratio of the number of routing packets discarded by recipients to the total routing packets sent by senders Processing delay: the time interval from receiving data to forwarding data for relay nodes $Trust_{service}$: is computed from the negotiation of the service parameter. If it is within a defined range, the value is 1; otherwise, 0. $Trust_{WSNs}$: trust value of a sensor node $Trust_{CSPs}$: trust value of a cloud node

Research	Trust aspects	Trust metrics	Determination of the trust metrics
[33]	QoS and other trust metrics	CPU frequency, memory size, hard disk capacity, current CPU utilisation rate, current memory utilisation rate, current hard disk utilisation rate, average response time, task success ratio, Energy consumption, response time and bandwidth	Not provided
[55]	Specific trust metrics	Experience level, plausibility level and accuracy level	Experience level: the quality level of interactions between two entities. Plausibility level: is measured by the correctness of the location information Accuracy level: the accuracy level of the location of an event
[34]	Specific trust metrics	Experience Satisfaction (ES)	Each fog node scores its collaboration experiences against the partner fog node based on QoS and QoP. 1 for a satisfied experience or 0 for an unsatisfied experience QoS: This research defines as the ability of a fog node to achieve maximum bandwidth and deal with requests with minimal latency. QoP: the degree of which a fog node protects data during processing and transferring to others
[93]	Specific trust metrics	Packet size, forwarding delay, and route success rate	Packet size, forwarding delay, and route success rate are normalised to be service quality ratings, and each attribute ranges from 0 to 1.
[4], [89], [92], [96], [90], [26], [97]	Not provided	-	-

Table 2.6: Comparison of evaluation methods for trust management approaches in fog computing

Evaluation approach	Tools	Research
Simulation	Java	[28], [93]
	System C	[21]
	MATLAB	[34], [94], [5], [95], [23], [88]
	Network simulator version 2 or 3	[55], [96], [99]
	iFogsim and other supporting tools	[4], [33], [6], [100]
	FogNetSim++	[97]
	MATLAB, MuPAD and Simulink tool	[91]
	Cooja	[98]
Real system	Blockchain on cloud	[87], [25], [3]
No evaluation provided		[90], [26], [89], [92]

Chapter 3: Fog computing scenarios and their trust relationship analysis

Fog computing, as described in Chapter 2, is a computing paradigm that provides local computation, storage, and networking services in various scenarios. This chapter analyses three representative fog computing application scenarios, including smart homes, smart traffic, and e-health. Understanding those scenarios concretely will aid in identifying the requirements to design the trust management architecture in the following chapter. In each scenario, we first explore several device types that participate in different layers of the fog computing paradigm. Next, we present fog-based applications to better understand device interactions. The trust relationships among the devices will be analysed, and the key trust aspects that should be considered in trust management will be discussed.

3.1 Scenario selection

The three scenarios presented in this chapter were chosen to elicit concrete requirements for designing the proposed trust management delegation protocol in fog computing environments. These scenarios are widely studied application domains in fog computing research, provide a clear role for the fog layer, and involve explicit trust management requirements.

The smart home scenario is included because it is one of the most widely studied application domains in fog computing research. The benefits of the fog layer in this context are clear, particularly in terms of local data processing and storage. It also involves a wide range of heterogeneous devices, such as home sensors and actuators. Many of these devices are resource-constrained and unsuitable for performing trust management locally. This scenario highlights the requirements for a trust management architecture that must support diverse device capabilities.

The e-health scenario is considered because fog computing has been leveraged for many healthcare applications that require real-time processing, low-latency response, and secure management of sensitive medical data. It also highlights the need for trust management to maintain privacy, integrity, and reliability in health information. For

this reason, the scenario is useful for eliciting the trust relationships and trust properties that the delegation protocol must support across different entities.

The smart traffic scenario is selected because the role of the fog layer is especially clear in time-sensitive applications, such as road monitoring, safety warnings, and traffic control, where data processing must occur near the data source. In addition, this scenario introduces a mobility property that is less explicit in the other scenarios. Vehicles and other mobile nodes continuously move across fog coverage areas, making trust management more challenging and requiring the transfer of trust management across fog domains with minimal delay.

3.2 Smart home scenario

Smart home systems have become a popular emerging technology that provides homeowners and occupants with more comfort and convenience in their homes. Furthermore, smart home systems can automatically collect real-time data from multiple sensors and intelligently control various home appliances and other devices, without human intervention, to manage home functions.

3.2.1 Fog-based smart homes architecture

To make homes smart and intelligent, smart home applications need to process and store data collected from several devices, analyse the data and compare it with user-preference values, and issue commands to control home devices [103]. These tasks could not be done by the home appliances themselves because of the limitations of their resources and capabilities. The smart home applications can use traditional cloud computing. However, connecting smart home devices directly to the cloud might face several challenges. With the distance to the cloud, the home devices suffer from high latency, making them unsuitable for latency-sensitive applications, especially home safety and security systems that require low-latency network connections to ensure real-time notifications and alerts [52]. Moreover, accessing the stored data in the cloud raises security and privacy concerns for sensitive data, as the data can be intercepted while being transmitted [104]. As a result, fog computing can provide processing and storage capabilities in close proximity to smart home systems while also addressing

the challenges mentioned previously. The architecture of a fog-based smart home scenario can be shown in Figure 3.1.

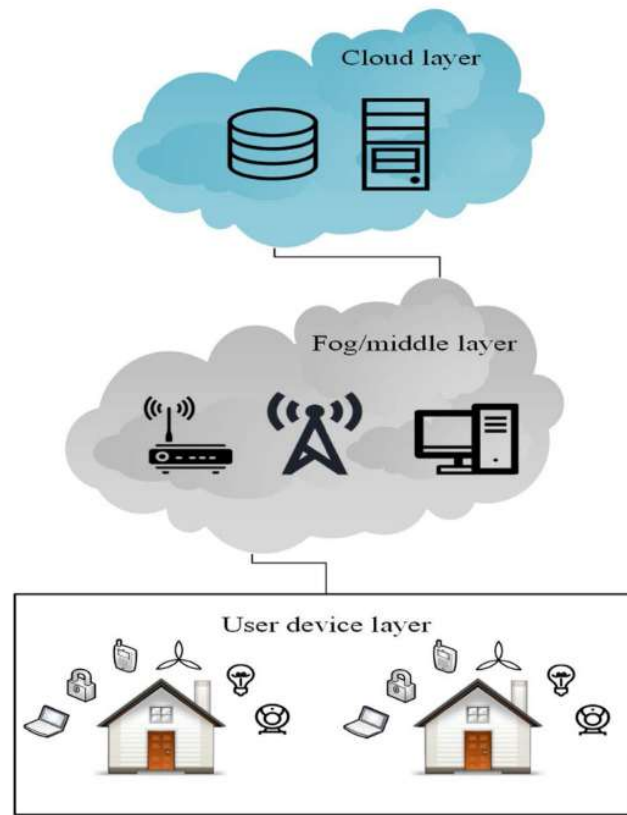


Figure 3.1: Fog-based smart home architecture [3]

The components of these layers are as follows.

1. End layer

The end layer of smart home architecture is primarily responsible for interfacing directly with users and devices, enabling real-time monitoring, management, and automation of household functions. This layer supports the execution of automated actions based on user preferences, sensor data, or predefined rules. End devices in this scenario comprise smart appliances such as smart refrigerators, smart televisions, smart washing machines, and smart door locks, which can be remotely monitored and controlled.

In addition to home appliances, the end layer includes a wide range of smart sensors for detecting motion, humidity, smoke, carbon monoxide, water leaks, temperature,

and ambient light [103]. These sensors continuously generate volumes of data and need to be transferred to the upper layer, the fog layer, for further processing and storage. Moreover, actuators such as intelligent light bulbs, smart switches, and a smart thermostat are also in the end layer [105]. These actuators are responsible for executing actions, such as adjusting the thermostat, dimming the lights, or activating home appliances, based on commands received from a control system operating at the fog layer. Smartphones and tablets are also part of this layer, providing users with interfaces to monitor, configure, and control home devices from anywhere [106]. These mobile devices enable users to remotely access real-time home data and adjust home settings. Some research has leveraged IoT devices to enhance the intelligence of home appliances and energy management systems. For instance, Pawar et al. [107] developed a smart socket module that measures the energy consumption of connected home appliances and transmits this data to a central gateway. The module can also remotely switch the appliances on or off in response to commands from the gateway. Similarly, the study in [105] used different kinds of sensors, such as movement and temperature sensors, as end devices to monitor environmental parameters within the home to support automated smart home functions.

2. Fog layer

The fog layer includes a home server, which could be a home router or gateway [103], a personal computer (PC) [108] or a smart home hub such as Samsung SmartThings and Google Nest or Google Home Hub, all of which can act as fog nodes. The fog node provides additional resources, including computational and storage services, to support various smart home applications. It gathers home environment data from end devices and performs several functions, including data preprocessing, compression, and storage. Furthermore, it could run several smart home applications, such as home automation, smart energy management, and home security systems [52]. The fog node that is located closer to the end devices can support local data processing and real-time response. It ensures that the applications take less time to process and respond to events than the cloud servers. Besides, the fog node can be an embedded device, e.g., Raspberry Pi 1 [109]. The authors in [105] proposed a distributed fog computing home automation system (HAS) that uses a Raspberry Pi to act as a home controller. The

Raspberry Pi was used to deploy home controller software, process data from several sensors and send prompt responses to actuator nodes. Moreover, the authors in [110] also used a Raspberry Pi 3B to gather data from smart meters and perform energy management and home appliance scheduling.

3. Cloud layer

In the smart home scenario, the cloud data centre with high-performance computing offers advanced data analytics, device integration, and remote access. It carries out computation-intensive applications, such as voice assistants, including Google Assistant and Amazon Alexa Voice Service (AVS). It also enables homeowners to monitor and control their smart devices via the Internet from anywhere.

3.2.2 Smart home applications

This section presents a use case in which energy is managed in a smart home environment to enhance energy efficiency.

1. Smart home energy management

Electricity prices vary depending on the time of use. To support price prediction, a nearby local fog server can efficiently preprocess energy consumption data and estimate electricity prices without transmitting the data to the cloud for processing. The authors in [52] employ a fog node to perform data cleaning and preprocessing on household appliance usage records. The cloud then performs clustering analysis of home appliance usage to represent the energy consumption behaviour of occupants. The result can help identify energy consumption behaviour and provide recommendations to reduce energy costs in households. The local fog server can also schedule and remotely control smart home appliances according to electricity prices received from a provider without affecting the comfort of residents [111]. The aim is to reduce or shift the high electricity demand of customers from peak to non-peak hours. Moreover, the local fog server can provide resilience to the system when there is no Internet connectivity to the cloud. A fog node deployed in a house can interact with smart home appliances to make an effective appliance scheduling strategy via the local network. In [112], the authors proposed an energy management system in a smart home based on fog computing. A fog node, a Raspberry Pi 4, gathers energy

consumption data from appliances and uses a Markov-chain model to forecast the future power consumption in different time slots. The fog node schedules appliance operations to reduce overuse during peak hours by shifting them from peak hours to off-peak hours. The system also considers environmental factors such as current temperature, luminance and humidity in the prediction to avoid affecting the occupant comfort. In the proposed approach of [113], a fog node deployed in a home controller schedules the charging time of an electric vehicle in order to reduce the electricity cost while still meeting the departure time of a user. The fog node processes information, including total battery capacity, current battery status, maximum charge of a charger, off-peak times, and user-specified departure time and then schedules the charging operation over the off-peak period. If more energy is needed, the fog node calculates and adjusts the electricity current rate to charge the battery based on the current time. The authors of [107] conducted a smart energy management system. A smart socket measures the power consumption of an appliance and sends data to a gateway via wireless communications. The gateway gathers the power consumption data of all home appliances and performs an algorithm to switch the appliances on/off based on their priority set by a user when reaching the maximum demand limit. Moreover, the gateway sends a warning message to a user about high-power-consuming appliances during peak hours to avoid high tariff charges.

Besides, effective energy management can save electricity costs for customers. It allows users to remotely switch smart home appliances, such as smart light bulbs, washing machines, and heaters, on and off according to the customer's need [114]. For example, customers can remotely turn off the lights by using their smartphones when they are outside and forget to turn them off before leaving. The customer may wish to turn on the heating for half an hour before they arrive home from work. There have been many products, such as smart light bulbs and smart plugs. Smart light bulbs allow customers to control them remotely through the Internet, anytime and anywhere. Moreover, some smart light bulbs, e.g., Philip Hue, can use GPS in occupants' smartphones to pinpoint customers' home locations and can automatically switch the lights on and off when they get home or leave the house, respectively. In addition, smart plugs such as Circle plug and WeMo Insight Switch mentioned earlier also allow

customers to control home appliances or whatever is plugged into it through smartphones or tablets.

2. Smart home automation systems

Smart home automation systems are designed for convenience and comfort for residents. The end devices collect data and transfer it to a fog node, a home server, for processing and analysis. After processing and analysing the data, the home server automatically adjusts home environmental factors as needed by controlling home appliances and devices. A framework in [115] leverages fog computing to monitor and control lights in a smart home. Various sensors and actuators (e.g., motion sensors, illuminance sensors, light actuators) are attached to Intel Galileo boards and deployed in different rooms of the smart home (e.g., bedroom, kitchen, bathroom, living room). A Raspberry Pi acts as a fog node, communicating with the sensors via IEEE 802.11g wireless. The fog node processes data from the sensors and automatically switches the lights in each room on or off according to people's presence and movements, and adjusts brightness according to the current illuminance level.

Besides, a home automation system in [103] can automatically monitor and control home devices. The proposed system consists of various kinds of sensors, such as light sensors, temperature, passive infrared (PIR), and smoke detection sensors to sense environmental measurements, and a set of relay modules to control home appliances such as air conditioning and light bulbs. The data from these sensors is transmitted to a Raspberry Pi, which acts as the main controller to process data and take actions. For example, the proposed system can automatically open or close doors when detecting the presence of humans by the PIR sensors. Air conditioners and fans can be turned on and off according to the temperature data received from temperature sensors. Furthermore, the Raspberry Pi controller can receive the weather forecast from the Internet and automatically control the air conditioners and fans, and open or close windows and curtains according to the weather conditions.

3. Smart home safety and security systems

Smart home security systems aim to provide protection against robbery, intrusion, and crime in a home environment. Security systems use video surveillance and various

sensors, such as PIR and acceleration sensors, to detect suspicious incidents. A fog node can perform data analysis to identify the incidents in proximity. As a result, it can respond to an event and report to the relevant authorities in near real time. The cloud data centre is used to store data in the long-term for statistical analysis and training datasets used to teach a machine learning model to process security information. A fog-based surveillance system for real-time crime detection in a smart home environment was proposed in [116]. In this research, a motion sensor detects movement, and a camera then captures motion-detected images. The images are transmitted to a fog node for processing. If the fog node detects and identifies the possibility of a crime event, the presence of a human and a weapon, using a deep neural network (DNN), it sends short message service (SMS) alerts to police officers or a crime prevention unit. The location of the fog node close to the edge nodes ensures real-time crime detection before a crime occurs. Furthermore, the proposed system enables the cloud server to store crime event data for statistical analysis and crime event data mining.

The research in [117] conducted a security management system in a smart building. PIR sensors are placed in a room to detect and predict occupancy in the room. Tri-axis acceleration sensors are used to monitor the status of the room doors, whether they are open or closed. The sensors are also attached to the window to detect window sliding and glass breaking. These PIR and acceleration sensors work together to collect data and transmit it to a fog node. The fog node is responsible for processing data to identify an intruder attempting to open the window of an unoccupied room and to detect glass breaking. If these events happen, the fog node triggers security alarms.

Furthermore, a system to detect and prevent an electrical fire caused by power overloads and short-circuits is proposed in [108]. A hall-effect current sensor measures the power consumed by home appliances and sends the data to a controller. If the power is overloaded above a threshold, it then reacts to a relay to cut off the power supply to home appliances to prevent fires and protect the appliances from damage.

3.2.3 Trust relationship analysis

Trust relationships in this scenario are between end nodes, between fog nodes, an end node and a fog node, and a fog node to the cloud.

1. Between end nodes

An end device communicates with another end device in smart homes for the purpose of home appliance control [114]. For example, a smartphone sends a command to turn a smart appliance, such as a smart light bulb, on and off. In this case, the smart light bulb looks for commands sent by the smartphones of the house members. A stranger may send a fake command (e.g., sending a command to turn on the light bulb when it is not needed). So, ownership of the two devices could be used to estimate the trustworthiness of the smartphone. The assumption is that a command sent from devices within families, or from devices owned by the same owner and flatmates, is trustworthy.

2. Between an end node and a fog node

End nodes in a smart home scenario are considered resource-limited devices. They need to offload their tasks and data to fog nodes in the fog layer. An end node must evaluate the trustworthiness of a delegated fog node in the context of providing services before offloading smart home tasks. First, the end node needs a delegated fog node that is able to complete tasks, especially resource-intensive tasks. For example, a fog node has the processing power to handle a machine learning algorithm to detect and identify a crime event. This can be estimated by the task completion rate. In smart home security systems, the fog node is expected to provide high availability. It must be available to receive and process data from home security cameras for an urgent event such as a crime. Furthermore, to respond immediately to the event, the end node expects the fog node to process the data it sends and take action on the incident as soon as possible. Response time could be used to estimate the trustworthiness of the fog node in this case.

Similarly, the fog node needs to assess the trustworthiness of the end node. A faulty or damaged sensor may send incorrect data. Besides, malfunctioning actuators could

perform unintended actions or fail to operate. To mitigate these risks, the fog node should evaluate the reputability and reliability of the sensors and actuators.

3. A fog node and a cloud data centre

Cloud data centres store data such as monthly or annual energy consumption. It also trains a dataset to teach the ML models how to identify a crime. The cloud resources needed to handle these tasks are not a problem for the resource-rich cloud. However, cloud servers should maintain the security and privacy of the data. On the other side, the cloud is concerned about data offloaded by broken or faulty fog nodes. Evaluating the reputability of the fog nodes can protect against this.

3.3 Smart traffic scenario

Smart traffic systems collect data from various sensors and analyse it to enable real-time monitoring and management of road traffic. The smart traffic system can adjust traffic light timing to reduce road congestion. Additionally, it also enhances road safety by detecting accidents or hazards and supporting prompt responses.

3.3.1 Fog-based smart traffic architecture

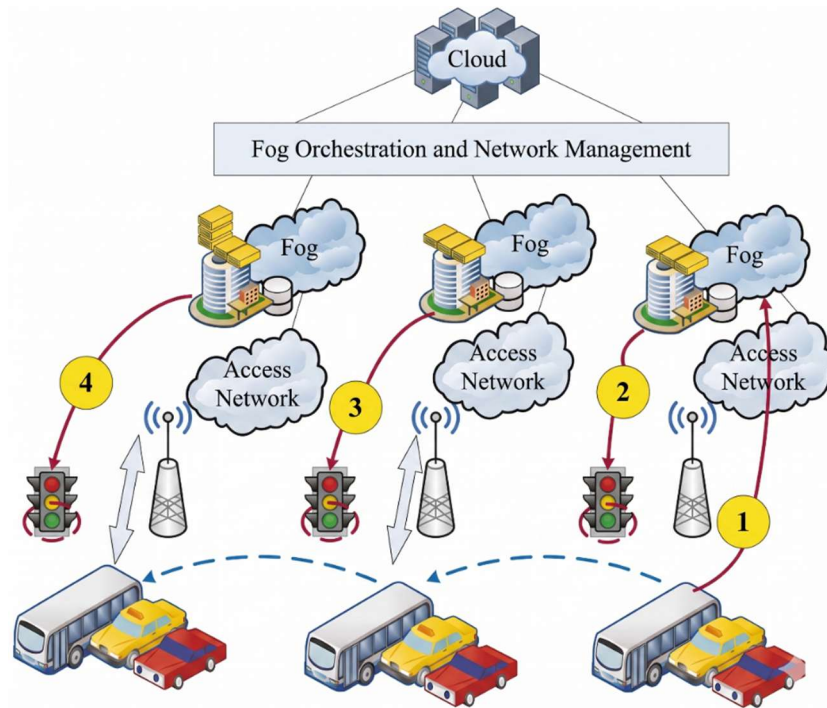


Figure 3.2: An architecture of smart traffic systems based on fog computing [111]

Fog computing plays a vital role in smart traffic systems. It significantly reduces network latency and response time when compared to traditional cloud-based systems. It brings data processing and decision-making tasks closer in proximity to where traffic events occur, for example, at intersections or along roadways, enabling real-time data processing and responses that are essential for effective road traffic management. The architecture of the smart traffic systems based on fog computing is shown in Figure 3.2 can be explained as follows

1. End layer

This layer is responsible for collecting data from local sensors, cameras, and other IoT devices at intersections or along roadways. It consists of vehicles equipped with several sensing devices (e.g., video cameras, radars, GPS, etc.), traffic lights, roadside cameras, etc [118]. An on-board unit (OBU) [119] is widely used to provide a vehicle with computation and communication capabilities. It usually consists of a processor and memory for processing and storing data, a wireless network device for communication based on IEEE 802.11p technology to communicate with Roadside Units (RSUs) or other on-board units. Some on-board units may have additional wireless communication technologies such as IEEE 802.11a/b/g/n or WIFI [120].

Petracca et al. [120] developed an OBU for linking vehicles to roadside networks in Intelligent Transport Systems (ITS). The device has a 32-bit Atmel ARM processor with a 400 MHz clock speed, an embedded GPS, a 2GB SD card, and a Ralink WUBR-506N card supporting IEEE802.11a/b/g/n standards. An OBU developed by Yang et al. [121] includes a Motorola MPC8313 processor at 333MHz, a 256Mb DDR SDRAM chip, an 8Mb NOR flash chip, an ET-318 GPS module, and a DCMA-86P2 module for communication via DSRC protocol. Moreover, Xu et al. [122] used a tablet and a PC deployed in vehicles to act as end nodes in their experiments.

2. Fog layer

This layer primarily performs local data processing and provides real-time responses to smart traffic systems. It also performs data preprocessing to reduce the high volume of data generated by IoT sensors and cameras in urban areas before being sent to the cloud. Roadside units (RSUs), which are generally deployed along roadsides,

pedestrian areas, or in dedicated locations such as at junctions and car parks in a city, act as fog nodes [118]. An RSU is equipped with computational and communication capabilities similar to the OBU but with higher performance than the OBU. One example of the RSUs is developed by Siemens. It has a Dual-core CPU at 800 MHz, 1 GB of RAM, an IEEE 802.11p 5.9 GHz DSRC module, 802.11 b/g/n and Bluetooth 4.0 modules, and a power-over-Ethernet module for power supply. Moreover, computing nodes co-located with cellular base stations, Wi-Fi access points, and other network infrastructure devices can also act as fog nodes [123]. Furthermore, fog nodes can be parked and moving vehicles that are attached to more powerful OBUs. Hou et al. [124] proposed the concept of vehicular fog computing (VFC) that utilises parked and slow-moving vehicles in an urban area near RSUs to form a group of fog nodes. It leverages vehicles' resources to provide computational and storage capabilities in fog computing systems. Zhu et al. [125] utilised commercial vehicles such as buses and taxis as fog nodes to gather and process crowdsourced video for several applications, e.g., parking navigation and road construction detection.

3. Cloud layer

Cloud servers are responsible for storing data for further processing and performing data analytics to provide city-level decision-making. For example, the cloud performs an intelligent traffic light control algorithm to predict traffic conditions by considering relative information, such as historical traffic records, to offer a navigation service for smart vehicles to help control traffic flow in a city [126]. Moreover, by using a smart vehicle's current location and destination, the cloud can compute the best route to the destination based on traffic conditions. In [127], cloud servers is used to process a global view of car parking spaces in a city and assist drivers in car parking based on their preferences.

3.3.2 Smart traffic applications

The following are the traffic management applications that benefit from fog computing.

1. Dynamic traffic light control

Dynamic traffic light systems aim to minimise traffic congestion at a traffic light junction by using traffic condition information to compute the time duration of traffic lights and change their colour according to real-time traffic conditions. Therefore, this application requires real-time data collection and computation, and low-latency services. With fog nodes deployed in close proximity to the junction where traffic information is produced, they can make real-time data collection and processing, and immediately control the traffic lights. In this system, a vehicle communicates with the nearest fog node in a fog layer through wireless communications to send relevant data such as current location, destination, speed, and additional information (e.g., road traffic congestion, accidents, broken down vehicles) [118], [128]. Moreover, cameras located at an intersection capture traffic conditions as video files [128]. The video is then sent to a nearby fog node for processing, which can be operated by local storage servers such as network video recorders (NVRs) or digital video recorders (DVRs). Because of a short distance between the two nodes, they can communicate with each other with high bandwidth wireless communication and within a few hops when compared to the cloud that needs to send data via the Internet. This ensures the video files reach the fog node for processing in a short time. Using the received data, the fog node extracts information about intersection conditions, such as vehicle queue lengths, using computer vision (CV) technologies and adjusts traffic lights in order to manage traffic flow and minimise traffic congestion. For example, if traffic is congested on one side of an intersection and the other side is free, then the fog node turns the traffic lights green for the congested side and red for the other side.

A fog node also communicates with other fog nodes to get traffic conditions of other intersections that are outside its coverage area to perform traffic light management [128]. This is because every intersection interplays with the others. The cause of traffic congestion at an intersection may be influenced by the control of another intersection. Furthermore, the fog node can deliver specific information to a specific vehicle, e.g., the traffic condition at an intersection ahead, to assist the vehicle in changing its route to avoid a traffic jam [126]. Besides, the fog node aggregates and performs preliminary processing of the received data to produce statistical information (e.g., number of

vehicles, and average vehicle speed) and reports it to the cloud for further processing [118].

2. Road safety and approaching emergency vehicle warning

A road safety information service delivers helpful information such as road work warnings, speed limits, traffic diversions and car accidents ahead to drivers during their journey [129]. Besides, information about weather and traffic conditions, nearest petrol and restaurant location, tourist information, and car parking can be delivered to make their trip more comfortable and pleasant [119]. Fog computing can provide location-aware services based on the user's device location and time. Therefore, drivers can obtain travel advice and additional information that is specific to their geographical location. The authors in [129] designed a real-time traffic management scheme based on fog computing architecture to enable drivers to acquire information about road conditions and incidents. Vehicles sense event information along their travel routes, such as traffic jams, car accidents, and road surface damage, and upload it to a nearby roadside unit. The information is then processed by fog nodes that are formed by parked and moving vehicles near RSUs. After processing, the information is distributed by roadside units to vehicles within their communication range.

The approaching emergency vehicle warning aims to send notification messages to drivers when emergency vehicles such as ambulances, fire engines, police or other emergency vehicles are approaching in order to clear the road to allow them to reach their destination as soon as possible. Moreover, traffic lights along the emergency vehicle route must be set to green upon arrival at each intersection. In this system, roadside cameras and sensors detect emergency vehicles approaching an intersection by using light detectors, sensors to detect the vehicle's siren sound, and computer vision technologies [57]. The data is aggregated and sent to fog nodes in close proximity, each responsible for a couple of intersections, instead of sending them together to cloud data centres. Therefore, the fog nodes can process the data and then change the colour of traffic lights from red to green, allowing the emergency vehicle to go through an intersection without delay. The fog nodes also immediately send warning messages to other vehicles approaching. A warning message contains

information about emergency vehicle velocity, direction, lane information, and a path to guide giving way to the emergency vehicle [119]. In addition, a fog node also cooperates with other fog nodes to turn green the other traffic lights along the emergency vehicle route. Because fog nodes can communicate with one another within the fog layer with low latency, traffic lights along an emergency vehicle's route can be controlled with minimal delay.

An emergency vehicle in the work of [63] is responsible for collecting data and distributing warning messages to other vehicles in the vicinity instead of sending it to a fog node. An emergency vehicle prepares relevant information, such as its current position, speed, heading and driving route. Other vehicles that are close to the emergency vehicle route receive the warning message and then display it to drivers. Fog nodes or roadside units that are close to the emergency vehicle also receive the message and take action, e.g., turning traffic lights along the emergency vehicle route from red to green for the approaching emergency vehicle. Moreover, a smart road sign displays the type of emergency vehicle, the estimated time of arrival, and other information.

3. Vehicle, pedestrian and cyclist collision warning

A collision warning system was developed to improve driving safety by detecting potential collisions and sending warning messages to drivers, pedestrians, and cyclists. The collision warning system gathers useful information about all the vehicles in the vicinity, such as location, velocity, and direction of travel, from onboard units attached to the vehicle to predict potential collisions. The system then sends warning messages to drivers and therefore prevents collisions. Moreover, the system can protect pedestrians crossing a road from possible collision and injury [119]. With the benefits of fog computing, a collision warning system is able to collect real-time vehicular information, predict the possibility of collision, and alert drivers to potential collisions with low response time.

Xu et al. [122] proposed a fog-based collision warning system that utilises fog and cloud computing to predict the possibility of collision at an intersection. A vehicle collision could occur when multiple vehicles approach an intersection and drivers are

not aware of the traffic lights. A vehicle is equipped with several sensors and on-board units and periodically sends real-time status, including position, speed, driving direction and acceleration, to a fog node via DSRC protocol. Roadside units, Wi-Fi access points and parked vehicles act as fog nodes. They are responsible for collecting real-time information of surrounding vehicles, detecting and predicting potential vehicle collisions, and sending warning messages to corresponding vehicles. As a result, drivers can respond to the warning message, such as by stopping or slowing down their vehicle, to mitigate the risk of a collision.

Furthermore, cloud servers could be used to mitigate the risk of collisions. By using historical data (e.g., traffic status, timing, vehicle speed, vehicle position and road surface), the cloud can forecast potential collisions and send warning messages to drivers. The authors of [130] proposed a collision warning system to avoid rear-end collisions when a vehicle in the front is slowing down. A vehicle frequently broadcasts its state information, e.g., position and speed, to other vehicles in its proximity via the DSRC protocol. Each vehicle that receives the information estimates the collision probability and alerts the driver.

A sensor or camera installed near the sidewalk collects data of pedestrians crossing the road and transmits it to the system. The data is sent to a fog node in its vicinity, e.g., at bus stations, supermarkets, and roadside locations, via low-latency wireless communication. The fog node then executes the data to predict a collision risk. When a potential collision is detected, the fog node dispatches warning messages that can rapidly reach pedestrians and vehicles to prevent accidents. In addition to that, the fog node also measures the speed of approaching people and bicycles and changes traffic lights or pedestrian traffic signals to give convenience to the pedestrians and bicycles [62]. Furthermore, the fog node can send alerts to drivers of the corresponding vehicles to take action, such as slowing down, if there is a possibility of a collision. The work in [131] developed a fog-based architecture for protecting vulnerable road users. It utilises pedestrians' and drivers' smartphones acting as end nodes to sense data, namely their longitude, latitude, speed, and direction.

3.3.3 Trust relationship analysis

Interactions in this scenario occur between end nodes themselves, between fog nodes, between an end node and a fog node, and between a fog node and the cloud.

1. Between end nodes

End nodes communicate with each other to exchange information. First, data exchange between end nodes needs to be sent and processed with low latency. For example, a vehicle that broadcasts its state information to other vehicles to avoid rear-end collisions must send data with minimal delay. If there is a delay for a second, the vehicle will crash into the one in front of it. Network latency can be a metric to estimate trustworthiness. Second, an end node should avoid communicating with, or ignore, information from a fraudulent node. A fraudulent vehicle has the potential of providing fake data to deceive other vehicles to make wrong decisions, such as sudden braking, causing serious traffic problems and accidents. Reputability can be used to estimate the risk of communicating with the fraudulent node. Checking the data against some predefined rules, or based on background knowledge, or an approach in [132] helps to filter out fraudulent messages. Third, some end nodes might not be willing to provide their information due to battery constraints or might provide incomplete or defective information. To avoid misbehaving end nodes, cooperativeness could be used.

2. Between an end node and a fog node

An end node has a trust relationship with a fog node because the fog node provides computational and storage services to perform various smart traffic applications. First, the end node expects to offload data to the fog node that has the capability to process traffic condition information and provide accurate control of traffic lights, especially computation and storage capabilities to extract information about intersection conditions from video files using computer vision technologies. The end node can use the task completion rate to reflect the capability of the fog node. Second, although the fog node is capable of completing a task, it should process data and return results as quickly as possible. For example, in a dynamic traffic light control system, the fog node must process traffic condition data from vehicles and roadside cameras and must send commands to immediately alter corresponding traffic lights. Similarly, in

pedestrian and cyclist collision warning systems, the fog node is responsible for detecting pedestrians or cyclists crossing the road and delivering timely alerts to the smartphones of pedestrians and drivers. Response time can be used as a trust metric to estimate the fog node. Third, network performance is important in this scenario to transfer large video files to a fog node. It can be evaluated using network throughput, which is the rate at which data is transferred between the end node and the fog node. Fourth, the fog node is expected to be available when an end node requests. Smart traffic applications, particularly those related to safety, such as warning about a vehicle collision, require fog nodes to be available to process data and send alerts to a vehicle before a car crash happens.

On the other hand, fog node needs a way to identify end nodes that engage in malicious or untrustworthy behaviour because interacting with such end nodes can result in the fog node and other vehicles receiving unreliable and harmful data. In this scenario, end nodes may belong to different owners and could be vulnerable to malicious activities. A misbehaving end node might send fake data for selfish purposes. For instance, a vehicle may send false information to a fog node in order to clear the road for itself. It might also perform various attacks, such as DoS, to degrade the performance of the fog system. Accidental behaviour, such as sending false data, may occur because of faulty sensors, e.g., flashing lights and siren sound detectors to detect emergency vehicles. This misbehaviour is difficult to distinguish from selfish purpose, but the fog node can ignore false data from faulty sensors by using the approaches mentioned in the previous section. The fog node could use reputability to evaluate the misbehaving end node. Moreover, network latency is an important factor in transferring time-sensitive data to protect pedestrians crossing a road from possible collision. Nevertheless, as mentioned earlier, some electric vehicles need to conserve their batteries, so they might not distribute their information to others. So, cooperativeness could be used to evaluate the trust of these kinds of vehicles.

3. Between fog nodes

In this scenario, fog nodes communicate with one another to exchange data. The fog nodes in the dynamic traffic light control system exchange traffic condition data within

their coverage areas to effectively control traffic lights. In an emergency vehicle warning system, the fog node cooperates with another fog node to change traffic lights along the route of emergency vehicles. Therefore, network latency is important for transmitting information between fog nodes. Moreover, the target fog nodes should be available to cooperate with.

4. Between a fog node and the cloud

Cloud servers in this scenario are responsible for long-term data storage and big data analytics to provide a city view of decision-making on traffic management. As in other scenarios, cloud data centres can offer large amounts of resources. The ability of cloud servers is not a problem. However, cloud server availability and reliability are also important to ensure that the cloud is available when required by the fog node and operates continuously without failures. In the reverse, the cloud is concerned about the behaviour of fog nodes as they are responsible for data pre-processing and transmitting information to the cloud. RSU deployed at the roadside may be compromised, potentially acting maliciously or altering data for various reasons. The trustworthiness of a fog node can be assessed by evaluating its reputability. Additionally, the reliability of data sent by the fog node can be verified by cross-checking it with information from other fog nodes.

3.4 E-health scenario

E-Health, or electronic health, aims to improve the efficiency, accessibility, and quality of a healthcare system by enabling a wide range of digital healthcare applications, such as electronic health records and telemedicine.

3.4.1 Fog-based e-health architecture

A fog-based e-health system leverages fog computing into healthcare systems to address the limitations of traditional cloud-based models. e-health requires real-time processing and a low-latency response. It is important to analyse time-sensitive data and respond to circumstances in seconds because a slight delay costs a patient's life, e.g., monitoring the vital signs of patients with acute ischaemic stroke. Therefore, e-health applications cannot rely on traditional cloud architecture to collect and process

vital signs and bio-signals of patients in real-time. Fog computing is the most efficient approach for tackling the issue. A fog-based e-health system consists of components, as shown in Figure 3.3.

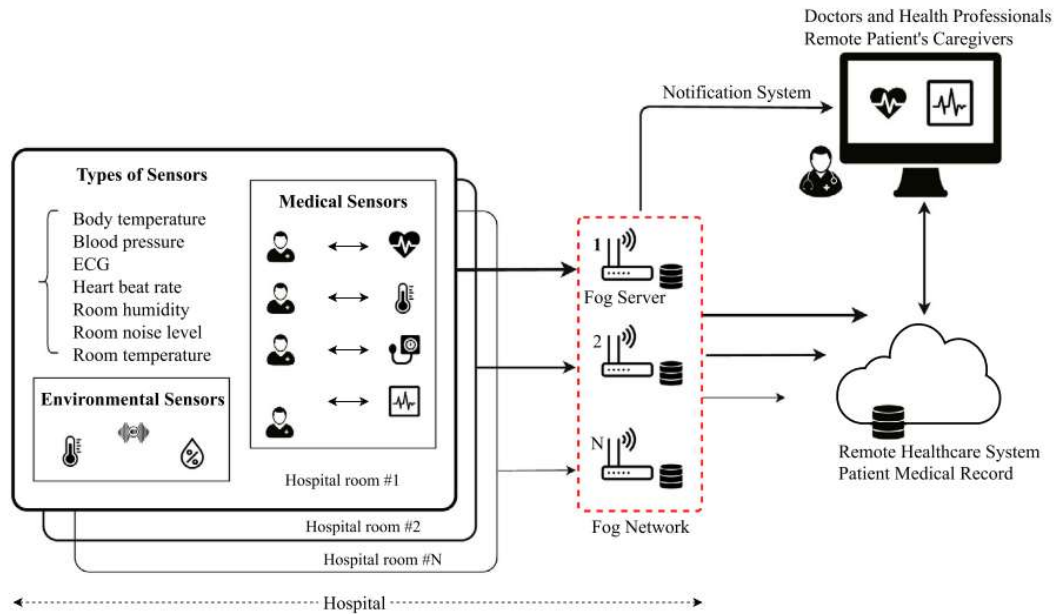


Figure 3.3: Fog-based e-health system [134]

1. End layer

The end layer comprises various medical sensors and actuators, smartwatches, smartphones, etc. The end devices generate data and send it to a fog layer for processing and storage. They communicate with the fog layer via various communication protocols, such as Bluetooth, Wi-Fi, ZigBee or 6LoWPAN. The IoT-based medical devices in this layer collect patients' vital signs and transfer them to the fog layer [133]. It can be divided into three distinct groups. First, implantable and wearable medical devices, such as cardioverter defibrillators and cochlear implants, support the functions of specific organs or tissues and monitor physiological activities (e.g., respiration rate, body temperature, blood pressure, blood oxygen level, and ECG). Second, medical sensors (e.g., heart rate, blood pressure, temperature, oxygen saturation, glucose levels, and ECG) and actuators (e.g., insulin pumps and pacemakers) to monitor vital signs and implement actions based on sensor data, respectively. Third, environmental sensors for capturing light, temperature, humidity, etc. Because some IoT-based medical sensors and actuators are very limited in

resources and capabilities, they are usually plugged into a wireless module or a microcontroller in order to transmit sensory data to a fog layer, such as Arduino boards, which are equipped with Bluetooth, 6LoWPAN, and WIFI [134], [135], or ESP8266 for Wi-Fi communication [136].

2. Fog layer

A large volume of data generated by many medical sensors is transferred to a nearby fog node for real-time healthcare data processing, notification, storage and preprocessing. A fog layer in e-health contains one or several fog nodes, and/or sink nodes [137]. The purpose of the sink nodes is to enable the network capability to connect different sensors via different communication protocols. Each fog node is connected to a group of end devices that typically covers a small area (a room in a hospital, a home, etc.). It collects data from end devices and performs data processing and storage for several e-health applications. With the fog node located at the edge of the network closest to the end devices, it can analyse the time-sensitive data and make a decision with minimal delay. Besides, the fog node periodically sends aggregated data to the cloud for further advanced data analysis and long-term storage.

There are several types of devices acting as fog nodes in e-health systems. Vilela et al. [137] used a Raspberry Pi Zero board as a fog node communicating with medical sensor nodes through Wi-Fi. The Raspberry Pi Zero is equipped with a 1GHz single-core CPU, 512MB of RAM, 802.11 b/g/n WIFI, Bluetooth 4.1, and Bluetooth Low Energy (BLE). Besides, Craciunescu et al. [138] use a home PC as a fog node. The work [56] proposed a smart e-health gateway that has several communication protocols with sensor nodes. A Panda board acts as a fog node, and a SmartRF06 board integrated with a CC2538 module acts as a sink node for 6LoWPAN. The main functionalities of fog nodes in e-health are:

- Local data processing: Data such as an ECG signal from a sensor contains P wave, QRS complex, T wave, U wave, and a lot of noise. It needs to be pre-processed to reduce noise, extract features (i.e., P and T waves), and calculate heart rate [135]. Axis acceleration data from a posture sensor attached to the patient's body is sent very frequently, every 50 ms, to a fog

node for fall protection [138]. With fog node location, it can process the high-rate incoming data and respond to the fall delegation in real time.

- Data compression: Data compression helps to reduce data size before sending to the cloud or storing in a fog node. For example, Rahmani et al. [135] used a tar method, a command on Linux, to compress a file in a fog gateway before sending it to the cloud.
- Local data storage: Raw data from medical sensors in the end layer, as well as processed data, can be compressed and stored locally in a fog node without sending to the cloud. However, the fog node can upload the data to a cloud server and then remove old and duplicate data from the local storage.
- Data security: The data stored on a fog node can be encrypted with a public key by an asymmetric encryption method using the Crypto library in Python [135]. Transmitting data to the cloud is vulnerable to interception.
- Data fusion: Collecting data from multiple sources can improve the accuracy of the data in case of sensor failure and noise interference. A fog gateway can apply data fusion on the sensory data from diverse sources. For example, the gateway combines heart rate values from the monitored person using two different sensors [135].
- Real-time notification: A fog gateway informs devices in the end layer, such as smartphones or tablets, about abnormal situations with low latency. The notification might be a heart rate that is outside the normal range, a notification about not receiving any data from a specific sensor during a period of time, or a body temperature of a patient that is above a threshold. Moreover, the fog node can send a quick notification to caregivers and family members when it detects a fall of a patient [135], [138].
- Graphical user interface: Data stored in a fog node can be accessed by end users such as caregivers, medical doctors and system administrators. The fog node also provides visualising data and other related data with low latency and response time compared to the cloud [133].

3. Cloud layer

A cloud layer consists of cloud servers that provide high computing power and large amounts of storage capacity. The cloud aggregates patient data from the fog layer and can perform data processing to diagnose diseases that require a more powerful computational capacity, such as diagnosing ischemic heart disease and myocardial infarction. Furthermore, it can provide data access for patients, caregivers, and medical experts via a mobile application or a web interface [135].

3.4.2 E-health applications

By leveraging fog computing, wearable devices and sensors can continuously collect, analyse, and monitor patient health effectively, making it possible to timely identify early signs of health problems and instantly send notifications to healthcare providers, caregivers, or family members. The following are the E-health applications that benefit from fog computing.

1. Real-time fall and stroke detection

Fall detection systems utilise accelerometer and gyroscope sensors in wearable devices attached to a patient's body to monitor movement patterns and detect falls. These sensors collect relevant data and send it to a fog node for fall detection processing. When a fall is detected, the fog node immediately triggers alerts to caregivers, family members, and emergency services, resulting in significantly reduced response times when compared to traditional cloud-based approaches. The authors in [139] present a fog-based fall detection system to enable real-time fall recognition and notification. Smartphone accelerometers capture movement data and transmit it to a local fog node. The fog node performs local preprocessing and makes immediate fall detection decisions. The cloud is responsible for retraining the global fall-detection model using machine learning with new data. This architecture enables the fog node to notify nearby caregivers over local area networks even when the Internet connection is unavailable.

The paper in [140] presented a fog-based healthcare system for real-time prediction and monitoring of stroke disease. Traditional stroke prediction and monitoring systems faced latency, network congestion, and privacy concerns because all data is

transmitted to cloud servers. The proposed system utilises fog computing as an intermediate layer between sensor devices and cloud servers. Sensors collect various types of patients' health data, including health symptoms, the surrounding environment and personal information. The data is sent in real time to a fog computing gateway to be processed and analysed using Random Forest with Boosting (RFB) to predict stroke risk. The fog gateway also delivers real-time alerts to patients based on the prediction results. Furthermore, the processed patient data is stored on a cloud layer, facilitating data sharing with authorised users such as healthcare providers and family members, and supporting historical data analysis and feedback mechanisms

2. Real-time healthcare monitoring system

Fog computing significantly enhances the efficiency of healthcare monitoring systems. The authors in [141] leverage fog computing to provide real-time notifications for patient monitoring. Sensors attached to a wearable device continuously measure the patient's vital signs, including body temperature, heart rate, and blood pressure. The sensed data is processed locally by fog nodes using machine learning algorithms to detect abnormal vital signs relative to normal threshold values. This approach reduces latency and response time for abnormal vital-sign detection compared to traditional systems. When abnormal readings are detected, the fog node immediately notifies doctors or caregivers. Additionally, all health data are transmitted to the cloud for long-term storage, comprehensive analysis and future references by hospitals and researchers.

The authors in [142] proposed a framework that integrates fog computing to support elderly healthcare monitoring. In this system, a MySignals HW V2 platform, equipped with wearable biomedical sensors, collects data such as blood oxygen levels, body temperature, heart rate, blood pressure, patient position, and other vital signs from elderly individuals. These data are transmitted to a fog server, which allows elderly individuals and their families to monitor their health status. It can also facilitate communication with health care providers for recommendations. The fog server also performs local data processing and analysis to deliver real-time notifications to caregivers or practitioners when abnormal health parameters are detected. Additionally, it handles local storage and data filtering before transmitting the pre-

processed data to the cloud for permanent storage and further analysis by healthcare professionals.

During surgical procedures, anaesthesia is vital to ensure patient comfort and improve the safety of operations. The authors in [143] proposed a fog-based anaesthesia monitoring system to monitor patient status during surgery. The aim was to improve patient safety and operating room efficiency. Traditional anaesthesia management carries the risk of underdosing or overdosing, which could lead to complications. The proposed approach utilised fog computing for data processing and decision support for anaesthesiologists. Sensors equipped with electroencephalography (EEG) tools collect and stream real-time brain activity data from patients during surgery to a fog node. The fog node processes EEG data to assess anaesthetic depth and detects abnormal patterns that may be potential risks. Additionally, it can deliver critical alerts to anaesthesiologists, enabling them to closely monitor patient status and respond promptly to any issues.

3. Medical diagnostics and chronic disease management

Fog computing revolutionises medical imaging by enabling real-time processing and analysis of large volumes of medical images, such as CT scans, MRIs, and X-rays. The authors proposed a healthcare framework for image analysis that leverages fog computing to detect COVID-19 symptoms from chest X-ray (CXR) images. In this architecture, diverse data from different devices, such as medical devices, sensors, and electronic health records, is collected and transmitted to fog nodes. The fog nodes perform COVID-19 case detection from image processing of CXR by using a modified Convolutional Neural Network (CNN). By conducting computationally intensive analysis at the fog layer, the system reduces latency, bandwidth usage, and privacy risks compared to traditional approaches. Only processed results or selected data are transmitted to the cloud for further storage, advanced analytics, or long-term decision support.

Rincon et al. [144] proposed an IoT and fog computing-based system for remote, real-time monitoring of cardiovascular patients, with a focus on the automatic detection of arrhythmias such as atrial fibrillation. Patients wear an ECG sensor device and bio

signal sensors that collect data, including photoplethysmography (PPG), oxygen saturation (SpO₂), phonocardiography (PCG) and temperature. The ECG signals and other sensed data are transmitted via the LoRa protocol to a group of fog nodes, where they are analysed locally using deep learning algorithms. If a cardiac anomaly is detected, the fog nodes automatically send a diagnostic report, along with the acquired data, to cloud services for in-depth analysis. The automatic identification of cardiac anomalies assists physicians by providing a clearer picture of the patient's condition and supporting better diagnostic decisions about treatment.

3.4.3 Trust relationship analysis

1. End nodes and fog nodes

An end node has a trust relationship with a fog node for data processing and storage. First, the fog node must have sufficient processing power and storage capacity to perform e-health tasks and store data, respectively. For example, a large amount of storage capacity is required because an ECG sensor generates a gigabyte of raw time-series data within a day. The capability of fog nodes can be measured by the task completion rate. Second, many healthcare applications require handling data quickly and responding to situations as fast as possible, e.g., in a fall detection system. Therefore, the sensor expects data to be outsourced to the fog node, which can provide low data processing time and deliver results on time. So, response time could be used in this case. Third, low latency for data transmission is also important. Actuators in this scenario need fog nodes transmitting data with low latency, such as sending a notification to a caregiver, as well as adjusting the insulin pump, etc. Fourth, the availability of the fog node is an important attribute. Offloading a task or data to an overwhelmed fog node causes a delay. The fog node is expected to be available when required. Regarding network bandwidth, high bandwidth is not necessary to send some data in this scenario. For example, blood pressure requires 1.44 kbit/s, whereas an electrocardiogram (ECG) typically requires between 36 and 216 kbit/s [137]. However, sending electroencephalogram (EEG) data, which measures electrical activities in the brain, requires a higher bandwidth, up to 921.6 kbit/s [137]. Therefore, the network throughput of fog nodes should be evaluated. Besides, the end node needs to ensure that the fog node is well-behaved. A misbehaving fog node might be installed

with malicious software that sends commands to adjust the insulin release rate, causing a serious problem for a patient. Sensitive vital signs and the patient's data stored in the fog node might be disclosed and misused. To avoid misbehaving fog nodes, reputability can be used to estimate the behaviour. Nevertheless, multiple fog nodes might be deployed in a hospital. The end node needs to communicate with the nearest fog node for better data transmission. So, the number of hops is a factor in measuring the distance between them.

On the other hand, a fog node trusts an end node regarding its behaviour. The fog node needs to communicate with a well-behaved end node. A malicious end node can engage in malicious activities. Reputability can be used to avoid interacting with the malicious end device. Besides, sending data with a high sample rate might result in a packet loss. Axis acceleration data from a posture sensor is sent very frequently at every 50 ms. The data might be lost and does not reach the fog node. So, the fog node can measure the packet loss rate of data sent by the end nodes. Furthermore, the response time of an actuator, such as an insulin pump, must be evaluated. The fog node requires the insulin pump to execute commands and respond to insulin level adjustments in real time. Response time is one of the vital factors that the fog node must evaluate towards the end node.

2. Between fog nodes and cloud servers

A fog node needs to trust cloud servers because they receive data from fog nodes for long-term storage and further processing, such as statistical analysis and advanced analytics. Cloud server capability is not a problem because it is considered a resource-rich layer in fog computing architecture, which contains many high-end servers. However, a fog node is concerned about the cloud server maintaining the security and privacy of user data. Regarding trust in cloud servers towards fog nodes, a cloud server is concerned about the behaviour of fog nodes. The data may be altered by the fog node.

3.5 The requirements for trust management architecture

Based on the analysis of these scenarios, there is a wide range of devices (end nodes, fog nodes, cloud servers) with varying computational, memory, storage, and energy

capacities. Many end devices, such as implantable medical devices, ESP8266 microcontrollers, and Arduino boards, have extremely limited resources. Due to these constraints, such devices are unable to perform any trust management tasks locally and must rely entirely on more capable devices for evidence collection, trust computation, decision-making and repository management. Some smart home appliances can only collect trust evidence but lack sufficient computational power or storage capacity for trust computation and storage. Devices such as smart home hubs or routers deployed in hospitals may have sufficient computational resources for trust calculation and efficient data collection. However, they often lack storage capacity for maintaining extensive long-term historical trust data. Additionally, battery-powered devices, such as smartphones or tablets operating over cellular networks, typically have adequate computational and storage resources for trust computation. However, frequent or periodic evidence collection can significantly drain their batteries. Therefore, they are unsuitable for continuous evidence gathering in practice. Some trust evaluation schemes employ sophisticated algorithms such as fuzzy logic and machine learning approaches [55]. These approaches require more processing power and memory than these devices can provide [97]. In large-scale environments, such as smart traffic systems, a vast number of devices generate volumes of trust-related data [119]. This can create significant storage challenges for devices like On-Board Units (OBUs) and Roadside Units (RSUs). It is especially important to ensure that resource-constrained devices are able to participate effectively in trust management. Therefore, the trust management architecture must accommodate all device types within the system, taking into account varying capabilities, hardware configurations and operating systems.

In contrast, devices operating at the fog layer are typically equipped with greater computational power, memory and storage capacity, and energy resources [2]. Devices with high computational resources, such as personal computers, can handle complex trust computation models. A local storage server in surveillance systems with high storage capacity can serve as a dedicated node for trust data storage. Therefore, the trust management architecture should leverage these powerful nodes to efficiently manage trust management tasks in fog computing systems.

Trust relationships are established among all participants in a fog computing system. The architecture must be designed to maintain trust between every pair of interacting entities, including not only end nodes and fog nodes but also between end nodes and fog nodes themselves. Furthermore, trust relationships are dependent on specific application contexts and require the consideration of multiple trust aspects to effectively manage trust among devices. Therefore, trust management should support a wide range of trust dimensions and should evaluate trust based on multiple criteria, such as availability, response time, reliability, reputation, and more, rather than being restricted to a single dimension. In large-scale scenarios such as smart traffic management, Fog environments are dynamic, with devices frequently joining and leaving [119]. They may move from one place to another place. Therefore, the trust architecture must support dynamic fog environments with mobility management.

3.6 Conclusion

In this chapter, we have presented detailed information about three scenarios that benefit from fog computing, namely smart traffic, smart grids and e-health, along with their applications and functionalities. The details of device types participating in each scenario were explored. Furthermore, a discussion of trust relationships between entities and the trust aspects needed has also been analysed. From the study in this chapter, it can be seen that each scenario consists of many device types that have different capabilities. Some devices, especially end devices in the end layer, have limited resources, while fog devices in the fog layer have a larger amount of resources. The resource-limited devices need to offload their tasks to other devices. Although resource-rich devices have a lot of resources, they may be overwhelmed by their main tasks.

Chapter 4: Trust management delegation protocol and design

This chapter presents the design of a trust management delegation protocol that enables delegators to delegate trust management responsibilities to delegates.

4.1 Why develop a trust management delegation protocol

After analysing the requirements of trust architecture from the previous chapter, traditional task offloading approaches in fog computing systems, previously discussed in Section 2.8, cannot fully fulfil the requirements. There are four reasons why a new delegation protocol is needed. First, the trust management delegation protocol is designed to accommodate a diverse range of devices, so that even resource-limited end devices can run it to participate in trust management. Moreover, the delegation protocol provides a way to assign trust management roles, including evidence collection, computation, decision-making, and repository, to other devices based on their individual capabilities. This approach allows devices to participate in trust management by delegating only those tasks that they are unable to perform themselves, and allows delegates to provide only the needed outsourced services, which isn't feasible in traditional task offloading approaches, such as code offloading [145]. As a result, the protocol enables effective utilisation of available resources throughout the fog systems. Second, the protocol employs lightweight technologies such as CoAP and SQLite (as described in the following chapter). It imposes minimal overhead in communication, computation, memory, and storage, making it highly suitable for resource-constrained devices. Third, the delegation protocol recognises that each fog application may assess trust across multiple dimensions and that trust relationships must be managed not only between end nodes and fog nodes but also among all participating entities. Lastly, the delegation protocol incorporates a handover mechanism to transfer trust management roles from a previous delegate to a new nearby delegate without additional burden on the constrained delegator.

4.2 Device capability classes

Constrained devices can be classified based on multiple facets that often apply in combination, for example, limitations in ROM/Flash, RAM, processing power, available power, or user interface [146]. Bormann et al. [146] proposed a device classification for constrained-node networks, but it does not adequately apply to devices in fog systems. This research therefore adapts their model to categorise fog computing devices into four device classes, shown in Table 4.1, with resource thresholds derived from the representative devices identified in the analysis in Chapter 3. Delegation decisions in the proposed trust management protocol are then justified with respect to these classes.

Table 4.1: Classes of devices in fog computing systems

Attribute /Class	Class 0: Highly constrained	Class 1: Constrained	Class 2: Less constrained	Class 3: Resource-rich
CPU	< 400 MHz	400 MHz – 1 GHz	1 GHz – 1.5 GHz	> 1.5 GHz
RAM	< 1 MB	1 MB – 512 MB	512 MB – 2 GB	> 2 GB
ROM/Flash	< 4 MB	4 – 1024 MB	1 GB – 4 GB	> 4 GB
Energy	battery-powered, less than 1000 mAh	battery-powered 1000–5000 mAh	mains powered	mains powered

1. Class 0: Highly constrained devices

- Hardware: CPU < 400 MHz; RAM < 1 MB; ROM < 4 MB.
- Energy: Battery-powered with capacity < 1000 mAh,
- Capabilities and delegation: Cannot perform any trust management and must delegate all trust management roles to more capable nodes
- Example device: Small embedded sensors and simple actuators in smart homes and e-health, e.g., lightbulbs, implantable sensors.

2. Class 1: Constrained devices

- Hardware: CPU 400 MHz – 1 GHz single-core; 1MB – 512 MB; ROM 4 MB – 1024 MB.

- Energy: Battery-powered with 1000–5000 mAh capacity
- Capabilities and delegation: Can collect trust evidence locally, but must delegate trust computation, trust decision-making and trust evidence storage to delegates.
- Example device: More capable IoT nodes in smart traffic and e-health scenarios, such as on-board units in vehicles or wearable devices

3. Class 2: Less constrained devices

- Hardware: CPU 1 GHz – 1.5 MHz single-core or multi-core; RAM 512 MB – 2 GB; Storage 1 GB – 4 GB.
- Energy: Mains powered
- Capabilities and delegation: Can maintain local databases of trust evidence and values. Execute simple trust models such as weighted-sum methods. Serves as a delegate for partial delegation, trust storage, trust decision-making and trust computation of lightweight trust models. However, resource-intensive trust models (e.g., fuzzy logic and machine learning) or large-scale evidence storage must be delegated to more capable delegates.
- Example device: home hubs, smart gateways, and roadside units, which are more capable but still have limited resources.

4. Class 3: Resource-rich devices

- Hardware: CPU > 1.5 GHz multi core; RAM > 2 GB; persistent storage > 4 GB.
- Energy: Mains powered
- Capabilities and delegation: Can perform all trust management tasks, including large trust data repositories, complex trust models (fuzzy-logic and machine learning). Serves as a delegate for all trust management roles on behalf of Class 0–2 devices.
- Example device: powerful routers or gateways and mini servers.

4.3 Trust delegation components

The trust management delegation protocol consists of four main components that enable an entity to delegate trust management to others.

1. A delegate is a node that provides trust management services to resource-constrained nodes.
2. A delegator is a node that requires trust management tasks to be delegated to a delegate.
3. A trust service server (TSS) is a node responsible for managing trust management service registration and facilitating queries for trust delegation service descriptions.
4. A delegation protocol is a set of functions that allow entities in the trust system to communicate with others to delegate trust management.

4.4 Trust management roles and delegation services

The objective of trust management is to establish trust with other entities. The trust management process consists of 4 main roles:

1. Trust computer

The trust computer is a designated component responsible for computing and updating trust values within the system. It processes collected trust information based on trust models to calculate trust values on behalf of a delegator. The mechanisms for trust computation and update are discussed in Section 2.9.4.

2. Trust Repositor

The trust repositior is responsible for storing and maintaining trust evidence and trust values, serving as the data storage component that supports the operations of both the trust computer and the trust decision maker on behalf of a delegator.

3. Trust evidence collector

The trust evidence collector is a component responsible for collecting and gathering trust evidence on behalf of a delegator. It observes and records trust evidence from trustees, including both QoS and social trust properties, and supplies this evidence to the trust computer for trust value computation.

4. Trust decision maker

The trust decision maker is responsible for making trust decisions on behalf of a delegator by using trust values computed by the trust computer. It either determines

whether a trustee meets a required trust threshold or selects the most trustworthy node from a set of candidates, as follows:

- Trust Decision (TD): Enables a trustor to make a decision based on a defined threshold.
- Trust-Based Decision (TBD): Enables a trustor to select the most appropriate trustee from a set of candidate trustees based on their trust values.

Entities can perform all these roles themselves or choose to delegate specific responsibilities to others. However, the trust decision maker typically needs less resources compared to other roles. So, it is reasonable to work with the trust computer role, minimising additional communication and resource overhead for making trust decisions.

In this work, delegates are initially pre-trusted at the bootstrapping stage. However, this trust can be earned through delegate evaluation performed by delegators, either through direct observation or recommendations from other delegators. This evaluation may be based on factors such as computation accuracy and data storage success. Each delegator maintains local trust records for its delegates and updates them over time. Furthermore, the similar trust management approaches used to evaluate the trustworthiness of trustee nodes in the system can be applied to evaluate delegates, either across all their roles or for a specific role. If corruption or misbehaviour is detected, the delegator may immediately revoke the delegate and select an alternative one from the delegates provided by the TSS.

The delegation patterns included in the protocol are designed to cover delegation requirements identified through trust management requirements analysis in various fog computing scenarios. By limiting delegation options to these patterns, the protocol reduces overall complexity and eases complicated operations among heterogeneous devices. The delegation patterns are as follows:

1. Delegating all trust management roles, except for some types of evidence collection, to one or multiple delegates.

This pattern is necessary for highly constrained devices that cannot perform any trust management tasks locally. However, some evidence based on direct interaction cannot be delegated and must be collected by the delegator itself during its own interaction with the trustee.

2. Delegating evidence collection, except for some types of evidence, to a single delegate.

This option is useful when evidence collection becomes resource-demanding, for example, continuous behaviour monitoring, or collecting evidence with multiple trustees. This option is also needed for battery-powered devices because evidence collection can consume significant energy and communication resources.

3. Delegating trust computer to a single delegate.

This pattern is appropriate when a device can collect evidence locally but lacks sufficient processing power or memory to execute trust computation. Lightweight trust computation models, such as weighted sum, may still be feasible on moderately capable devices, but more resource-intensive models, such as fuzzy logic, multiple linear regression, and machine learning-based approaches, are better delegated to a more capable node.

4. Delegating trust computer and trust repository to one or multiple delegates.

This pattern is intended when both trust computation and trust storage cannot be performed within the resource capacity of the device. Moreover, this is especially relevant for trust management approaches that are based on complex multi-dimensional trust evaluations that use large volumes of evidence or historical interaction records, which must be retained and processed over time.

5. Delegating trust repository to a single delegate

This option applies when a device can perform trust computation locally but cannot maintain large amounts of trust evidence. Storage demands arise particularly in large-scale scenarios, where devices store large volumes of trust-related data.

4.5 Service discovery functions

4.5.1 The need for TSS-based discovery

Several service discovery protocols for IoT systems have been proposed, most commonly CoAP-based discovery [147], DNS-based discovery [148], Service Location Protocol (SLP) [149] and Universal Plug and Play (UPnP) [150]. However, the Internet community is currently considering CoAP and DNS-based service discovery as the primary protocols for resource-constrained networks [55]. Both approaches are well-suited for basic service discovery, but they are not sufficient to support the complex delegation requirements of the proposed protocol. The delegation protocol must discover not only the existence of services, but also whether candidate delegates satisfy complex trust delegation requirements, such as supported delegation roles, trust models, decision-making methods and evidence types. Moreover, DNS-based discovery, such as multicast DNS (mDNS) and DNS Service Discovery (DNS-SD), creates high overhead compared to other approaches [151]. Therefore, a custom discovery protocol is needed to support the specific requirements of trust delegation using the advantages of CoAP. A TSS is introduced as a directory lookup node that allows delegates to register their delegation services and enables delegators to perform lookups based on their requirements, as described in the next sections.

4.5.2 Delegation service registration

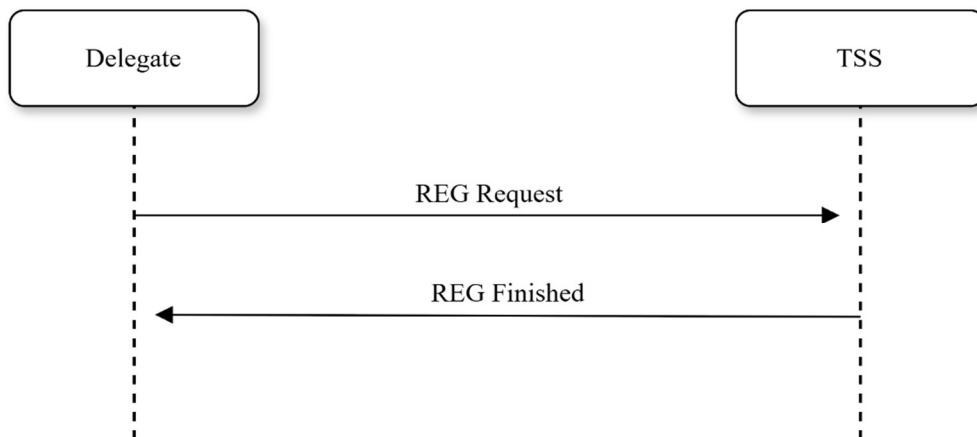


Figure 4.1: The process of delegation service registration

A delegate that intends to offer trust management services to others must register a delegation description with the TSS. This delegation service description will be stored on the TSS and made available for lookup by delegators. The process of how a delegate performs registration with the TSS is shown in Figure 4.1. The delegation service registration proceeds as follows.

1. The delegate sends a *REG Request* message with relevant parameters to the TSS to register the trust management services that the delegate offers. The message includes the following parameters:

- **Delegate ID:** A delegate identifier.
- **Lease time:** The duration of providing services in seconds.
- **Delegation roles:** A list of trust management roles provided by the delegate. If this parameter is blank, it indicates all roles.

The roles are defined as the following service codes:

- TC = Trust Computer
 - TR = Trust Repositor
 - EC = Evidence Collector
 - DM = Trust Decision Maker
- **Trust model:** A list of trust models supported by the delegate.
 - **Evidence storage:** A list of trust evidence types supported by the delegate for the trust repositore role. If this parameter is blank, the delegate does not support trust value and trust evidence storage.
 - **Evidence collector:** A list of evidence types supported by the delegate for the evidence collector role. If this parameter is blank, the delegate does not support the evidence collector role.
 - **Decision maker:** A list of decision-making types supported by the delegate. If this parameter is blank, the delegate does not provide a trust decision-making service.

It is important to note that a blank value means all values in that parameter. Using this approach in parameters reduces communication costs. This approach is beneficial in resource-constrained environments.

2. The TSS responds to the delegate with a confirmation message indicating the success of the registration. This message includes a registration reference number. It will be used to deregister services and renew the lease in the future. After this step, the registration is completed, and the delegate can start offering trust management services to others.

4.5.3 Delegation service deregistration

A delegate may stop providing some delegation services for various reasons, such as running low on resources. The deregistration process shown in Figure 4.2 enables the delegate to deregister its trust management delegation services that it no longer offers from the TSS. Once the services are deregistered, the TSS withdraws them from its database, making them unavailable for discovery. However, the delegate continues providing delegation services to existing delegators who requested them prior to the withdrawal until their lease durations expire.

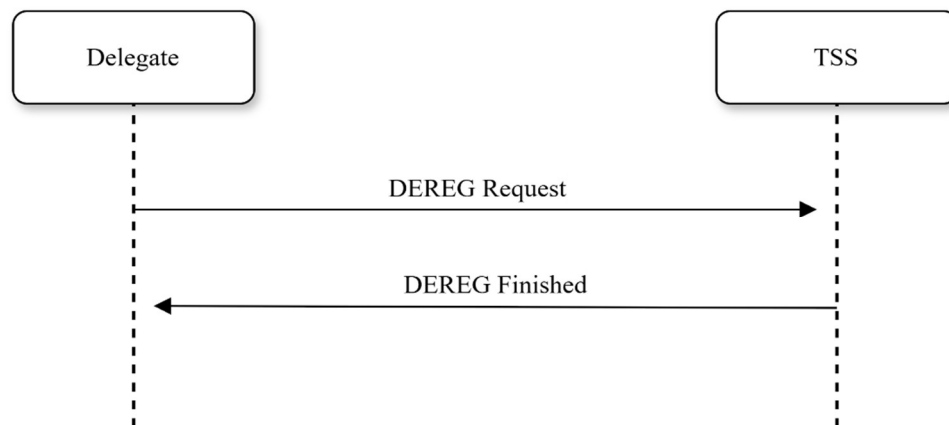


Figure 4.2: The process of delegation service deregistration

The deregistration process operates as follows.

1. The delegate sends a *DEREG REQUEST* message to the TSS. The message includes a registration reference and a list of trust management delegation services that the delegate wishes to deregister. If the list of services is blank, it indicates that the delegate is deregistering all delegation roles. The message includes parameters as follows.

- **Registration reference number:** The registration reference number received from the TSS during the registration process.
 - **Delegation roles:** A list of trust management delegation roles that the delegate needs to withdraw from the TSS. If this field is blank, it indicates that the delegate is deregistering all delegation roles.
2. The TSS removes the specified delegation roles from its database and sends a response message to confirm that the delegation roles have been successfully withdrawn.

4.5.4 Registration expiration and lease renewal

Lease time refers to the total time that delegates are willing to offer their delegation services to others. This mechanism is designed to prevent delegates from remaining in the system indefinitely and to minimise delegation downtime. When a delegate registers its delegation services with the TSS, it specifies a lease time, which indicates the duration of its delegation services. The lease must be renewed before its expiry. Otherwise, the delegate must re-register its delegation services with the TSS.

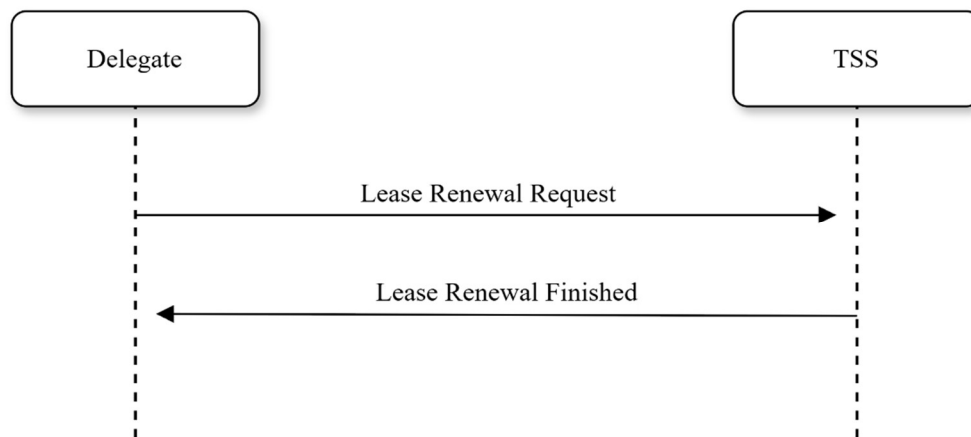


Figure 4.3: The process of delegation service registration lease time renewal

The lease renewal process, shown in Figure 4.3, allowing a delegate to renew or update its delegation service time with the TSS. The following are the steps of lease renewal.

1. The delegate sends a *Lease Renewal Request* message to the TSS to request a renewal of its lease duration. The message contains essential information,

including a unique reference number from the registration and the duration of the new lease, as follows.

- **Registration reference number:** A registration reference number
 - **New lease time:** A new lease time in seconds.
2. The TSS updates the lease information in its database and replies with a message to indicate that the lease renewal has been successfully completed.

4.5.5 Delegate lookup

After delegates have registered their delegation services with a TSS, these delegation services are available on the TSS for lookup by delegators. A delegator that needs to delegate trust management can search for delegate(s) that meet its delegation requirements from the TSS. After that, the delegator can directly communicate with them. Moreover, the delegator can look up multiple delegates for delegating multiple trust management roles at once. The process of looking up is shown in Figure 4.4.

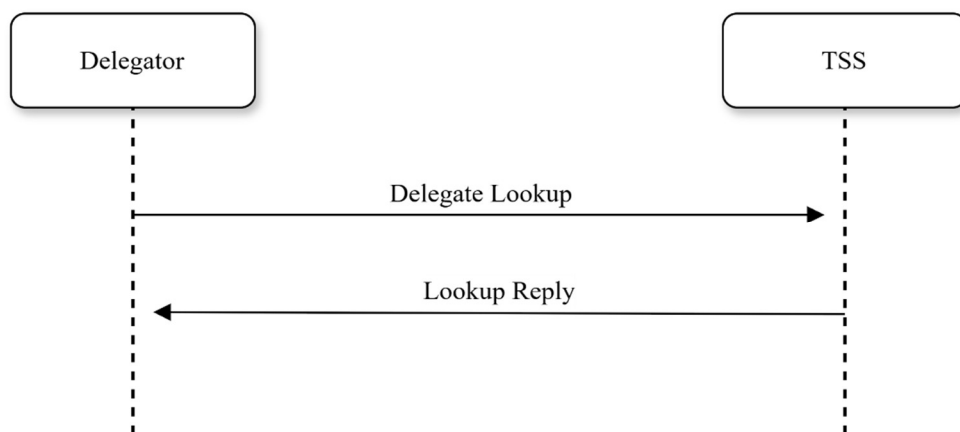


Figure 4.4: The process of delegate(s) lookup

The following describes how the delegate lookup works.

1. The delegator sends a *Delegate Lookup* message to a TSS with the requirements of the delegate specification to the TSS. The message includes the following information:
 - **Delegation roles:** A list of trust management roles that a delegator needs to delegate.

- **Trust model:** A trust model required by the delegator for trust calculation. This field is blank if the delegator does not delegate the trust computer role.
 - **Evidence storage:** A list of trust evidence required for the trust repositor role. This parameter is blank if the delegator does not delegate the trust repositor role.
 - **Evidence collector:** A list of trust evidence required for the evidence collector role. If this parameter is blank, the delegator does not delegate the evidence collector role.
 - **Decision maker:** The type of decision-making required for the trust decision-maker role. If this parameter is blank, the delegator does not delegate the trust decision-making role.
2. The TSS searches its database for delegates that meet the specified requirements. Once suitable delegates are identified, the TSS sends a reply message to the delegator. This message contains a list of matching delegates. After this step, the delegator and the delegates can directly communicate with each other.

4.5.6 TSS discovery

TSS discovery is a process used by a delegate or a delegator to find a TSS when no service is known, the existing service has expired, or the known service doesn't respond. There are two kinds of TSS discovery: pull and push approaches. In the pull approach, the delegate and delegator broadcast a message to find a TSS, while the push approach allows the TSS to broadcast its information to the delegate and delegator. The push method provides real-time updates, but it generates high network traffic. The pull method lacks real-time updates but incurs lower communication overhead [151]. In our protocol, the TSS offers both push and pull approaches simultaneously. The delegate and delegator can choose the approach, as they need to find a TSS.

1. Pull approach

In the pull TSS discovery approach, the delegator and delegate broadcast a message within a network to search for a TSS. When the TSS receives the message, it responds with a reply containing its information to the original sender, as shown in Figure 4.5.

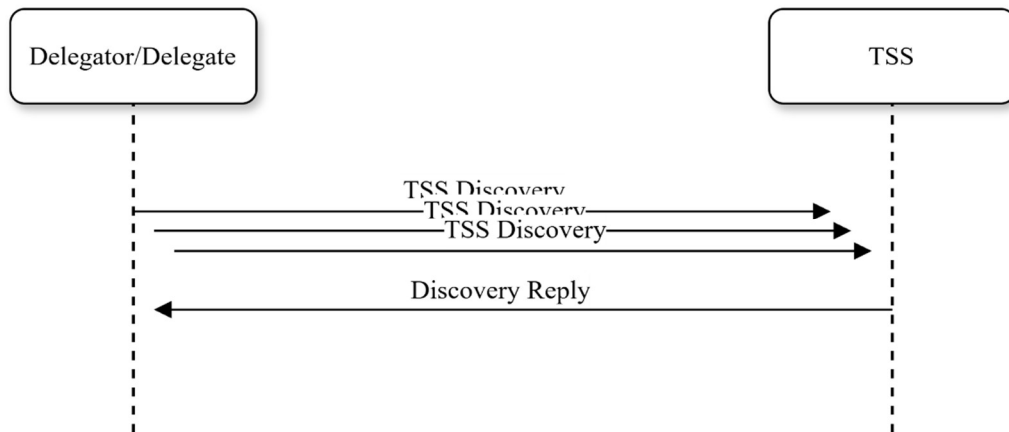


Figure 4.5: Pull TSS discovery process

The process of the pull TSS discovery is explained below.

1. The delegate or delegator broadcasts a *TSS Discovery* message to all devices within a network.
2. Upon receiving the *TSS Discovery* message, the TSS replies to the sender with a TSS Reply message containing its IP address. Then, the delegate or delegator can request service discovery functions provided by the TSS. The delegate or delegator might receive multiple reply messages from multiple TSS. The delegate or delegator accepts only the first reply and ignores subsequent messages.

2. Push Approach

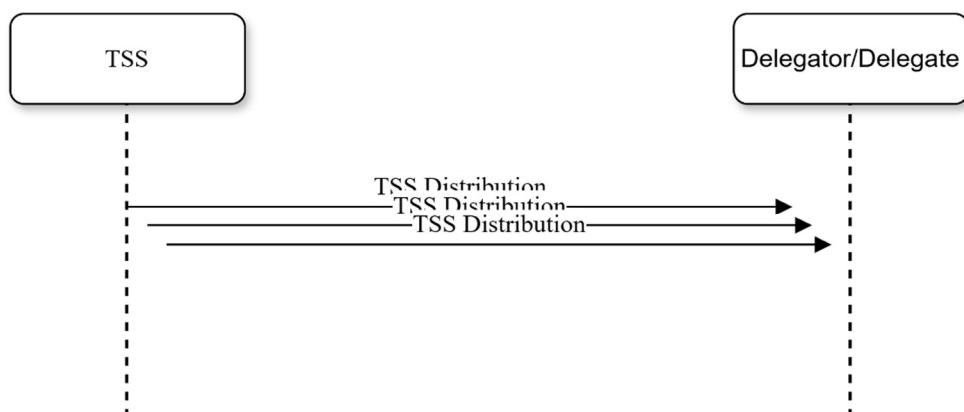


Figure 4.6: Push TSS discovery process

In the push method, the TSS actively broadcasts its information across the network rather than waiting for a *TSS Discovery* message from delegates or delegators. The primary advantage of this approach is that delegates or delegators can begin using the service discovery functions of the TSS immediately after receiving its broadcast information. The process of the push approach is shown in Figure 4.6.

The process of the push TSS discovery is explained as follows:

1. The TSS broadcasts a *TSS Distribution* message to every device within the network.
2. Any delegate or delegator that needs to request service discovery functions from the TSS accepts this message. Devices that do not require the service drop the message.

4.6 Trust management delegation functions

4.6.1 Delegate selection

A delegator assesses whether it has sufficient local resources to perform each trust management role. If some roles can be handled locally, it delegates only the unsupported roles. If none of the roles can be supported locally, they delegate all trust management roles. The delegator first discovers candidate delegates through the TSS and then selects one or more delegates from the registered delegates that offer the required delegation services. A single delegate is selected when it can support all required trust management roles. If no single delegate is capable, multiple delegates are selected to distribute the roles across different nodes. Delegate selection is performed according to the specific trust management role being delegated. A trust evidence collector is selected based on its support for the required evidence types, a trust computer based on the trust models it can execute, a trust decision-maker according to its support for the required decision-making method, and a trust repositior based on its ability to store the required trust evidence and trust values. This process ensures that delegation roles are delegated to the most appropriate registered delegate or set of delegates, rather than simply to any node with greater resources.

4.6.2 Delegating all trust management roles to one or multiple delegates.

1. Delegating all trust management roles to one delegate

When looking up delegates from a TSS, the delegator receives a list of delegate nodes along with their available delegation services. It can delegate all roles to a single delegate if a delegate providing all trust management roles is available. The delegate is responsible for trust computation, decision-making, evidence collection, and evidence storage on behalf of the delegator. The process of delegating all trust management roles to one delegate can be shown in Figure 4.7.

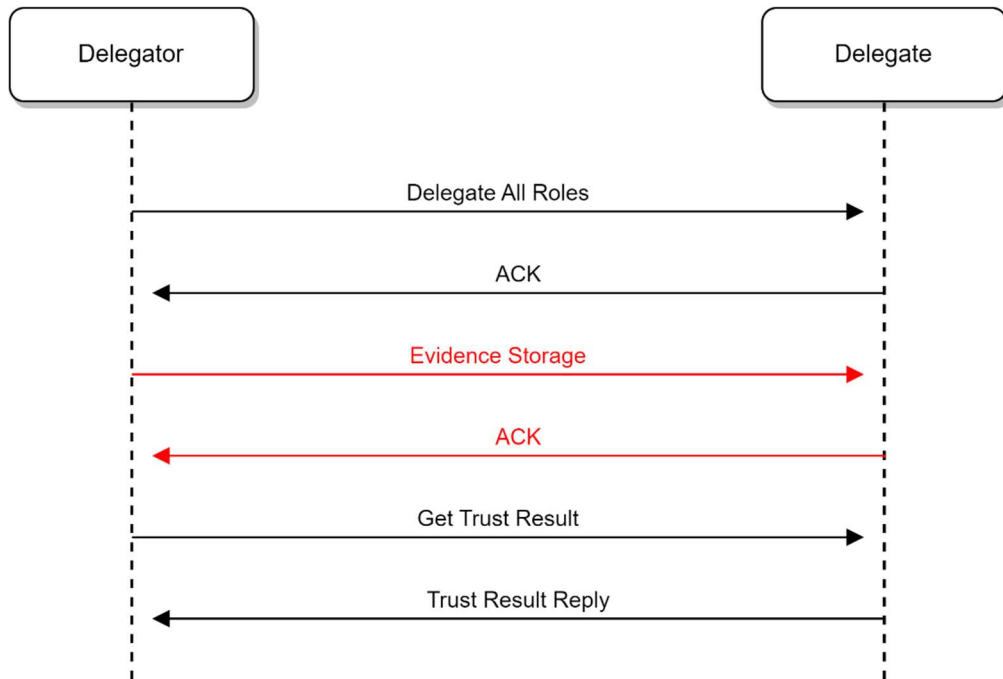


Figure 4.7: The process of delegating all roles to one delegate

The process in Figure 4.7 can be described in detail as follows.

1. The delegator sends a *Delegate All Roles* message to request delegation of all roles to the delegate. The message contains the following information.
 - **Trustor ID:** A delegator identifier.
 - **Trustee ID:** An identifier of a trustee, or a list of trustee identifiers if multiple trustees are included.
 - **Lease time:** Delegation time in seconds.
 - **Trust model:** A trust model for trust computation.

- **Trust_update:** Trust update scheme,
 - **Evidence:** A list of trust evidence.
 - **Data Sharing:** Boolean 0 or 1 indicates recommendation sharing (described in Section 4.8).
2. The delegate stores the delegation information in its database until the lease expires or when the delegation ends. In response to the *Delegate All Roles* message, the delegate sends an acknowledgement to confirm that the delegation is successful. The delegate then starts collecting trust evidence towards the trustee, storing trust evidence and updating trust values.

The delegate updates trust value towards trustees based on event-driven and time-driven approaches. In the event-driven method, trust is updated whenever the delegate completes evidence collection or receives new evidence from the delegator. In the time-driven approach, trust is updated periodically, for example, once every hour.

3. Some types of trust evidence, such as latency, cannot be delegated to the delegate. The delegator must collect this evidence itself. After collecting the evidence, the delegator sends an *Evidence Storage* message to the delegate, as shown in the process in red colour in Figure 4.7, with the following information:

- **Trustor ID:** An identifier of the delegator.
 - **Trustee ID:** An identifier of a trustee.
 - **Evidence:** A list of trust evidence with their corresponding value.
4. The delegator sends a *Get Trust Result* message to retrieve a trust result from the delegate. The message contains the following parameters:
 - **Trustor ID:** An identifier of the delegator.
 - **Trustee ID:** An identifier of a trustee, or a list of trustee identifiers.
 - **Threshold:** A threshold value for TD or left blank for TBD.
 5. The delegate performs trust decision-making and sends a *Trust Result Reply* message to the delegator, including the trustworthiness of the trustee (trusted

or untrusted) in case of TD, or the trustee with the highest trust value in case of TBD.

2. Delegating all trust management roles to multiple delegates

A TSS may reply to the delegator with multiple delegates to meet the delegator's delegation requirements. When delegating trust management to multiple delegates, the delegator must assign the role of trust management to each delegate. The process of delegating all trust management roles to multiple delegates is shown in Figure 4.8. The steps below explain how the delegation works:

1. The delegator sends the following messages to assign specific trust management roles to respective nodes. The *Delegate Trust computer* message includes the Trust repositior ID field for sending trust values to the trust repositior for storage. Similarly, the *Delegate Evidence Collector* message has the Trust repositior ID field for sending trust evidence to the trust repositior for storage. Meanwhile, the *Delegate Trust Repositor* message contains the Trust computer ID field, enabling the trust repositior to send new trust evidence to the trust computer for update. These processes enable seamless collaboration among the nodes and efficient execution of delegated trust management roles.

Delegate Trust computer message to a trust computer node.

- **Trustor ID:** An identifier of the delegator.
- **Trustee ID:** An identifier of a trustee, or a list of trustee identifiers.
- **Trust repositior ID:** A trust repositior identifier.
- **Lease time:** Duration of delegation.
- **Trust model:** Trust model for trust computation.
- **Trust update:** Trust update method, either event-driven or time-driven.
- **Evidence:** A list of trust evidence.

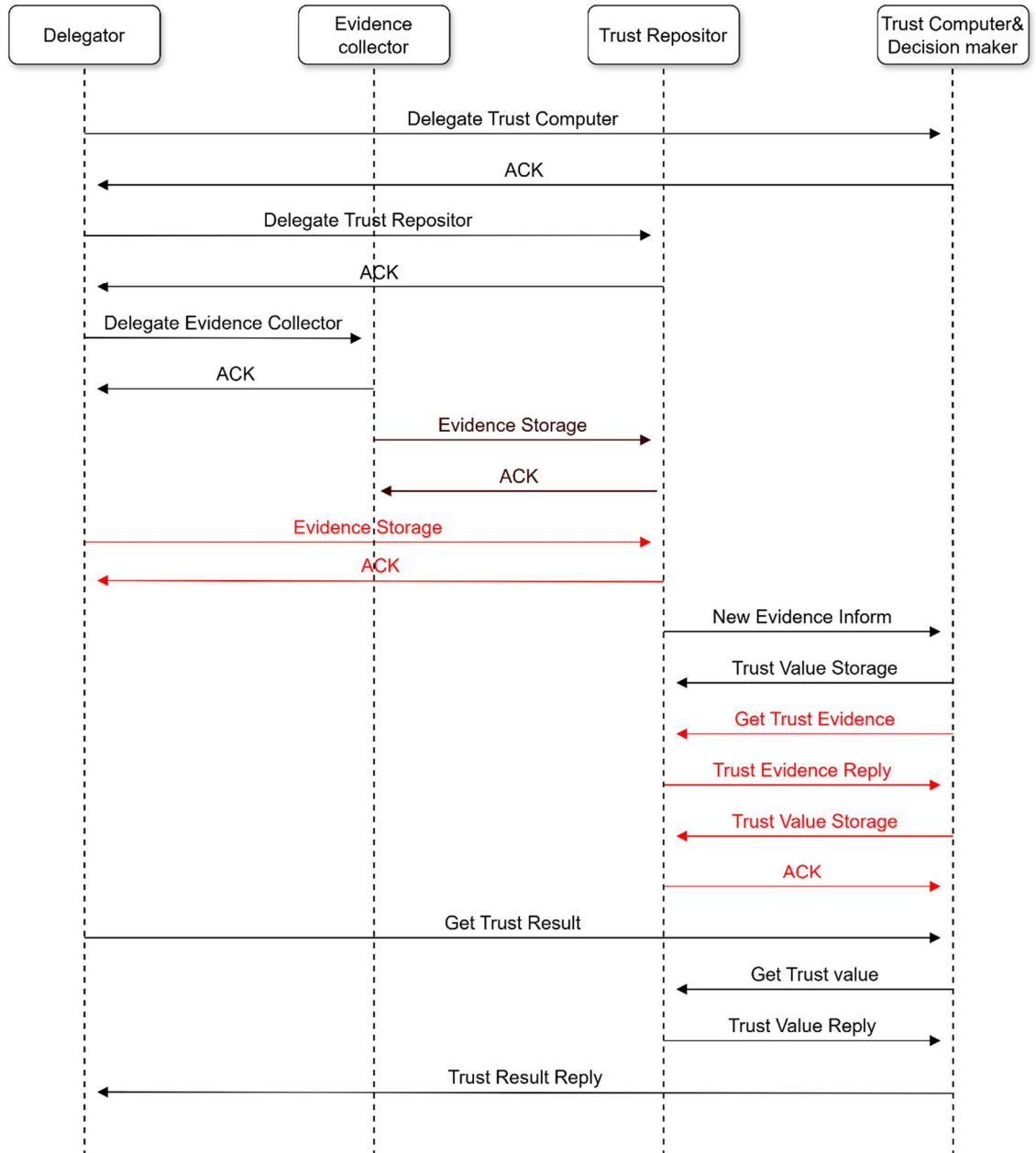


Figure 4.8: The process of delegating all roles to multiple delegates

Delegate Trust Repositor message to a trust repositon node.

- **Trustor ID:** An identifier of the delegator.
- **Trustee ID:** An identifier of a trustee, or a list of trustee identifiers.
- **Trust computer ID:** A trust computer identifier.
- **Lease time:** Duration of delegation.
- **Evidence:** A list of trust evidence.
- **Data Sharing:** Boolean 0 or 1 indicates recommendation sharing.

Delegate Evidence Collector message to an evidence collector node.

- **Trustor ID:** An identifier of the delegator.
- **Trustee ID:** An identifier of a trustee, or a list of trustees.
- **Trust repositon ID:** An identifier of a trust repositon.
- **Lease time:** Duration of delegation.
- **Evidence:** A list of trust evidence.

2. The trust computer, trust repositon, and evidence collector respond to the delegator with acknowledgement messages confirming the success of the delegation roles.
3. The evidence collector collects evidence towards the trustees and sends an *Evidence Storage* message containing the evidence to the trust repositon for storage. Additionally, the delegator can also send an *Evidence Storage* message with trust evidence that cannot be delegated directly to the trust repositon. The *Evidence Storage* message is the same as in the above section.
4. The trust computer can perform trust updates using either event-driven or time-driven approaches. In the time-driven approach, the trust computer periodically updates trust. In the event-driven approach, the trust repositon notifies the trust computer when it receives new evidence by sending a *New Evidence Inform* message. The *New Evidence Inform* message includes the following parameters:
 - **Trustor ID:** An identifier of the delegator.
 - **Trustee ID:** An identifier of the trustee.
 - **Evidence:** A list of trust evidence with their value.

Moreover, the trust computer can also request trust evidence from the trust repositior for trust updates by sending a *Get Trust Evidence* message with the following parameters:

- **Trustor ID:** An identifier of the delegator.
- **Trustee ID:** An identifier of the trustee.
- **Evidence:** A list of trust evidence names.

The evidence is then sent to the trust computer included in a reply message. After updating trust, the trust computer sends the new trust value back to the trust repositior for storage using a *Trust Value Storage* message with the parameters as follows:

- **Trustor ID:** An identifier of the delegator.
- **Trustee ID:** An identifier of the trustee.
- **Trust Value:** A new trust value after trust update

5. The delegator queries about the trustworthiness of a trustee by sending a *Get Trust Result* message to the trust computer. The parameters in the message vary depending on the type of decision-making, as described in the previous section. For TD, the message includes a threshold value and a trustee IP address. For TBD, the threshold is left blank, and the message contains a list of trustee identifiers. Because the trust computer does not store trust values in its database, it must retrieve the trust values from the trust repositior. This is done by sending a *Get Trust Value* message with the following parameters:

- **Trustor ID:** An identifier of the delegator.
- **Trustee ID:** An identifier of a trustee or a list of trustee identifiers for multiple trustees.

6. After receiving trust values from the trust repositior, the trust computer performs decision-making based on either TD or TBD, depending on the case. It then sends a reply message to the delegator, which includes either the trust result or the trustee with the highest trust value.

4.6.3 Delegating evidence collection to a delegate

A delegator must allocate resources to collect trust evidence over a long-term period. Trust evidence collection might last a long time and be more frequent in some scenarios. For limited-resourced devices, particularly battery-powered medical devices, it is crucial to conserve battery power for primary tasks. To address this, a delegator can delegate some types of trust evidence to an evidence collector. The evidence collector takes responsibility for collecting trust evidence on behalf of the delegator, thus reducing the resource burden. The process of delegating the trust evidence collector role will be illustrated in Figure 4.9.

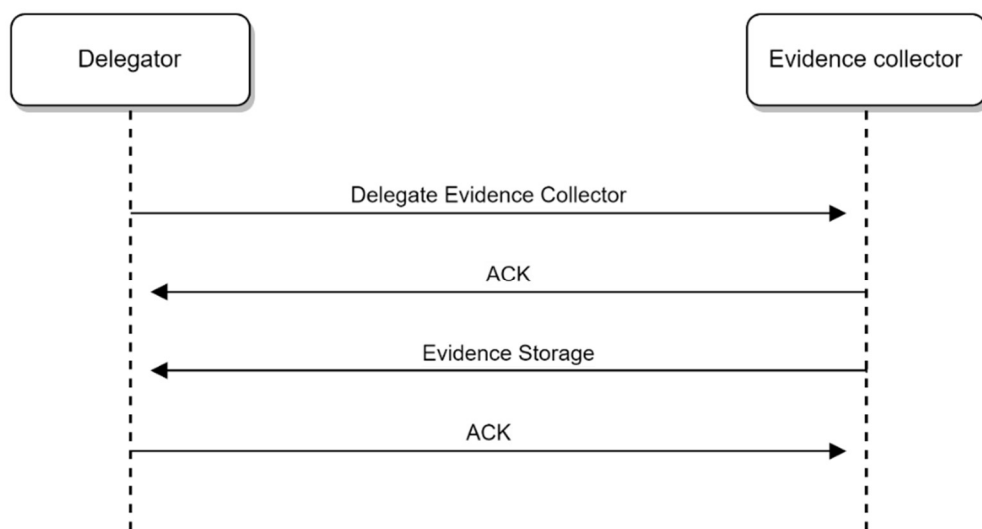


Figure 4.9: The process of delegating trust evidence collector role to a delegate

The details of delegating evidence collection are described as follows.

1. The delegator sends a *Delegate Evidence Collector* message with a list of trust evidence to be collected to the evidence collector, the same message as in the previous section. However, in this delegation, the trust repository ID field is left blank because the evidence collector does not send the evidence to be stored in a trust repository. The delegate responds with an acknowledgement message to confirm the successful delegation. At this point, the evidence collector starts collecting the specified evidence.

2. The evidence collector sends the evidence back to the delegator using an *Evidence Storage* message. The evidence can be sent immediately after collection or can be scheduled for periodic delivery to the delegator.
3. The delegator confirms the receipt of the trust evidence by sending an acknowledgement message. If the evidence is not successfully received, the evidence collector resends the evidence.

4.6.4 Delegating trust repository to a delegate

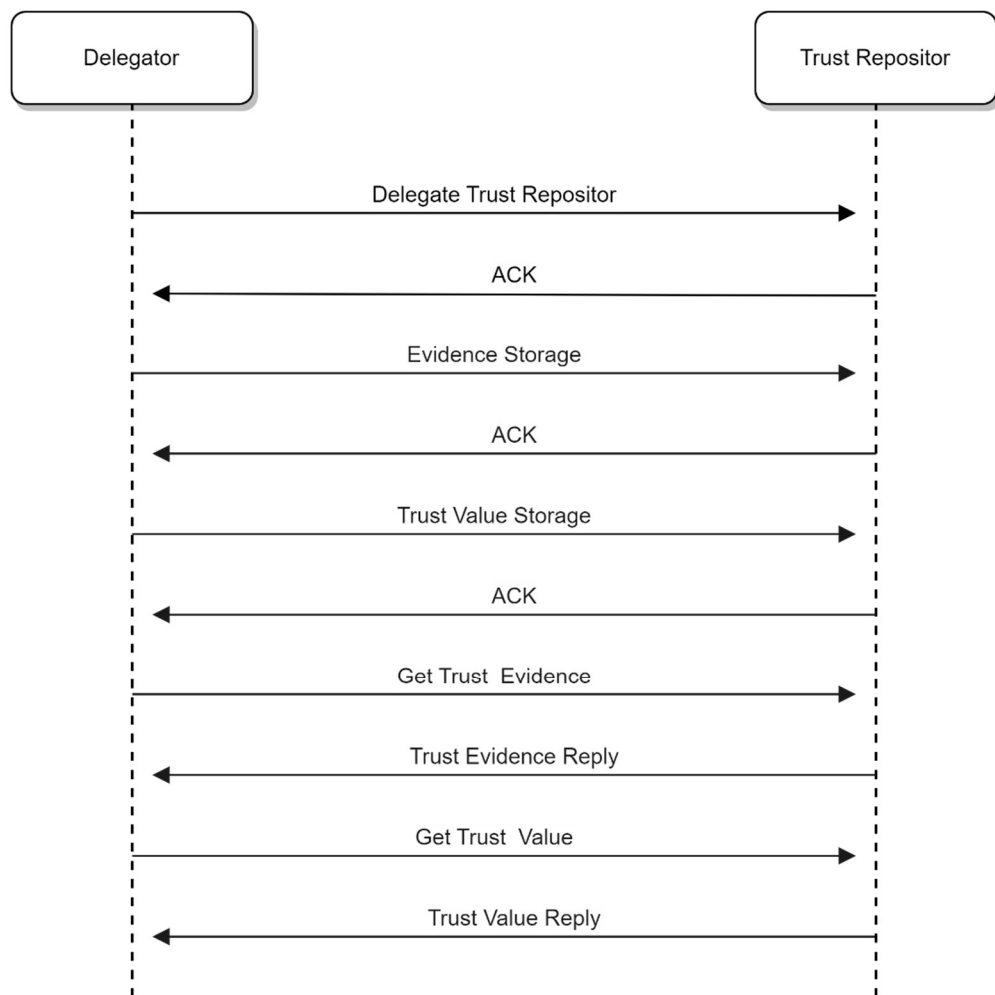


Figure 4.10: The process of delegating trust repository role to a delegate

Due to the limited storage capacity of end devices in a fog computing system, they can delegate trust repository role to a delegate to store trust values and evidence on their behalf. This is also important in scenarios like smart traffic, where a number of devices

participate in the system. resulting in significant storage capacity for trust value and evidence. Therefore, the delegator can send the trust value and evidence to a delegate node for storage.

Trust value and trust evidence can be sent to the delegate separately. The delegator may not be willing to update trust after the collection of trust evidence is finished. In other words, the delegator performs a time-driven trust update approach and chooses not to update trust values immediately after evidence collection. Furthermore, the trust values and trust evidence in the delegate can be retrieved by the delegator whenever needed. Figure 4.10 shows the process of delegating the trust repository role, and the details can be explained as follows.

1. The delegator sends a *Delegate Trust Repository* message to the delegate, the same message as the message in section 4.6.2. In this delegation, the Trust computer ID parameter is left blank because the delegate does not notify the delegator of new evidence. The delegate replies to the delegator with an acknowledgement message indicating whether the delegation succeeded or failed. If successful, the delegate will allocate storage space and prepare its database to store the data.
2. The delegator sends an *Evidence Storage* message to store trust evidence or a *Trust Value storage* message to store trust values, the same messages as in section 4.6.2. The delegate sends an acknowledgement message to inform the delegator of the status of the storage operation, indicating whether it was successful.
3. To retrieve stored data, the delegator sends a *Get Trust Evidence* message to retrieve trust evidence or a *Get Trust Value* message to retrieve trust values, the same message as in section 4.6.2. Upon processing the request, the delegate retrieves the data from its database and either sends trust evidence or trust values in an acknowledgement message to the delegator.

4.6.5 Delegating trust computer to a delegate.

Fog computing systems, such as e-health and smart traffic systems, are more susceptible to malicious devices than smart home systems. They require advanced and

effective trust evaluation models, such as fuzzy logic and machine learning trust models, to accurately assess node reliability and manage interactions with untrustworthy devices. However, these complex trust models demand significant computational resources, including CPU and RAM usage.

A limited-resource device delegates the trust computation role to a delegate to reduce the burden of resource consumption on the trust management. The delegate takes responsibility for performing trust evaluations on behalf of the delegator. The process of delegating the trust computer role to a delegate can be shown in Figure 4.11.

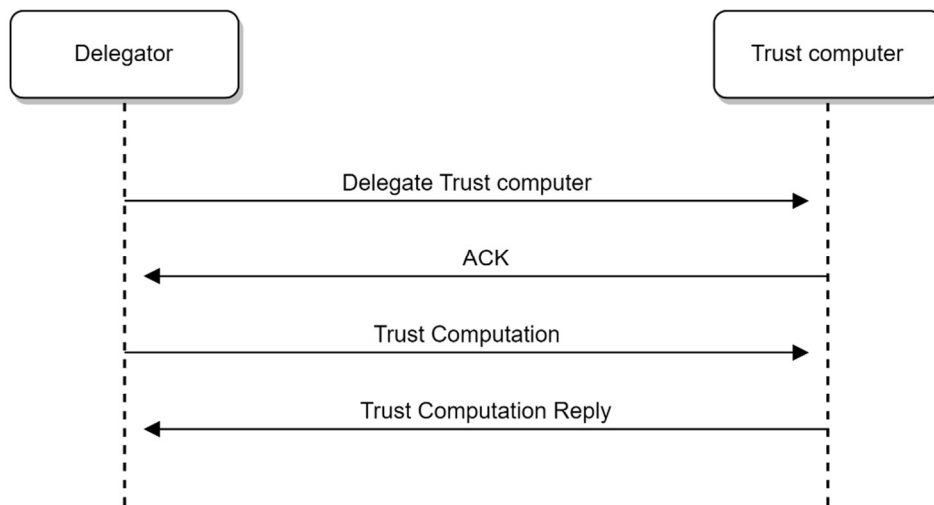


Figure 4.11: The process of delegating trust computer role to a delegate

The process of delegating the trust computer role in Figure 4.11 will be outlined as follows

1. The delegator requests the trust computer delegation role by sending a *Delegate Trust Computer* message to the delegate, as described in Section 4.6.2. The trust repositior ID field in the message is set to blank as the delegate does not send a trust value to be stored in a trust repositior. The delegate replies with an acknowledgement message to inform the delegator of the status of the trust computer delegation.
2. Once the delegator finishes evidence collection, it sends a *Trust Computation* message with a list of evidence and their respective values to the delegate. The delegate calculates a trust value based on the provided trust evidence and sends the trust value result to the delegator in a reply message.

4.6.6 Delegating trust computer and trust repositor roles to one or multiple delegates.

A delegator can delegate trust computer and trust repositor roles to a delegate or multiple delegates to minimise CPU usage and storage space. Separating the trust computer and trust repositor roles across different nodes is useful when computation and storage requirements are better served by different devices in the fog environment. The trust computer role may be placed on a node with high processing capability in order to perform trust calculation efficiently, while the trust repositor role may be assigned to a node with larger storage capacity to maintain trust values and evidence. This arrangement is particularly useful in heterogeneous fog scenarios, where devices differ in their capabilities. For example, in e-health, a nearby network router can be the choice for trust computation, but it lacks the storage capacity to store trust evidence and values. Therefore, trust data may be better stored on another node, such as a storage edge server, that offers greater storage capacity and better support for intensive database operations and stability.

However, this separation also introduces limitations. When the trust computer and trust repositor are deployed on different nodes, additional communication is required to transfer trust value and evidence, retrieve stored trust data, and update computed trust values. This increases network bandwidth usage and may also introduce additional latency, especially when trust updates are performed frequently or when the nodes are separated by multiple network hops. As a result, the separation of these roles may be less suitable in highly time-sensitive situations, where trust decisions must be made immediately, and communication delay may affect application response time. Therefore, the benefits of separating the trust computer and trust repositor roles must be balanced against the additional communication overhead.

1. Delegating trust computer and trust repositor roles to one delegate

If a single delegate is available to handle both the computer and trust repositor roles. The delegator can delegate these trust management roles to the delegate simultaneously. The process of delegating both the trust computer and trust repositor roles to one delegate can be shown in Figure 4.12.

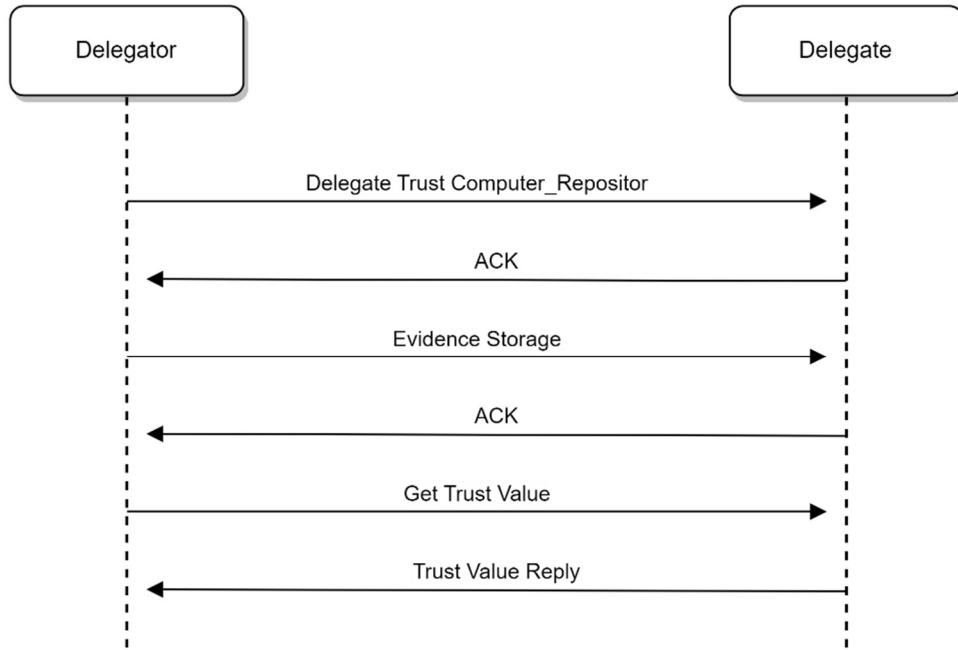


Figure 4.12: The process of delegating trust computer and repositor roles to one delegate

The following steps explain how the delegation of trust computer and trust repositor roles works.

1. The delegator sends a *Delegate Trust Computer_Repositor* message to the delegate to request the delegation of trust computer and repositor roles. The message contains the following parameters:
 - **Trustor ID:** An identifier of a delegator.
 - **Trustee ID:** An identifier of the trustee.
 - **Lease time:** Delegation duration time.
 - **Trust model:** A trust model for trust computation.
 - **Trust_update:** Trust update scheme, event-driven or time-driven.
 - **Evidence:** A list of trust evidence.
 - **Data Sharing:** Boolean 0 or 1 indicates recommendation sharing.

The delegate responds with an acknowledgement message to inform the delegator of the delegation status.

2. Upon receiving the acknowledgement message, the delegator starts collecting trust evidence towards the trustee and sends the evidence to the delegate for storage using an *Evidence Storage* message.
3. The delegate receives the trust evidence and stores it in its database. It can then perform a trust update either immediately upon receiving new evidence (event-driven approach) or periodically update (time-driven approach). After the trust update, the new trust value is stored in the delegate's database.
4. When the delegator needs to retrieve the trust value towards the trustee, it sends a *Get Trust Value* message to the delegate. In response to the *Get Trust Value* message, the delegate sends the required trust value attached in a reply message back to the delegator.

2. Delegating trust computer and trust repositior roles to multiple delegates

In addition to delegating to a single delegate, the delegator can delegate these trust management roles to multiple delegates. This allows the delegator to delegate the trust computer and trust repositior roles to different nodes. The process of delegating these trust computer and trust repositior roles to multiple delegates is shown in Figure 4.13. The delegation details are as follows.

1. The delegator initials the delegation process by sending a *Delegate Trust Computer* message and a *Delegate Trust repositior* message to a trust computer and a trust repositior, respectively, as described in section 4.6.2. The Trust repositior ID parameter in the *Delegate Trust Computer* message enables the trust computer to query trust evidence from the trust repositior using a pull approach for trust updates.
2. Both trust repositior and the trust computer respond to the delegator with acknowledgement messages. Once the delegation is confirmed, the delegator starts collecting trust evidence on its own and sends it to be stored in the trust repositior using an *Evidence Storage* message.

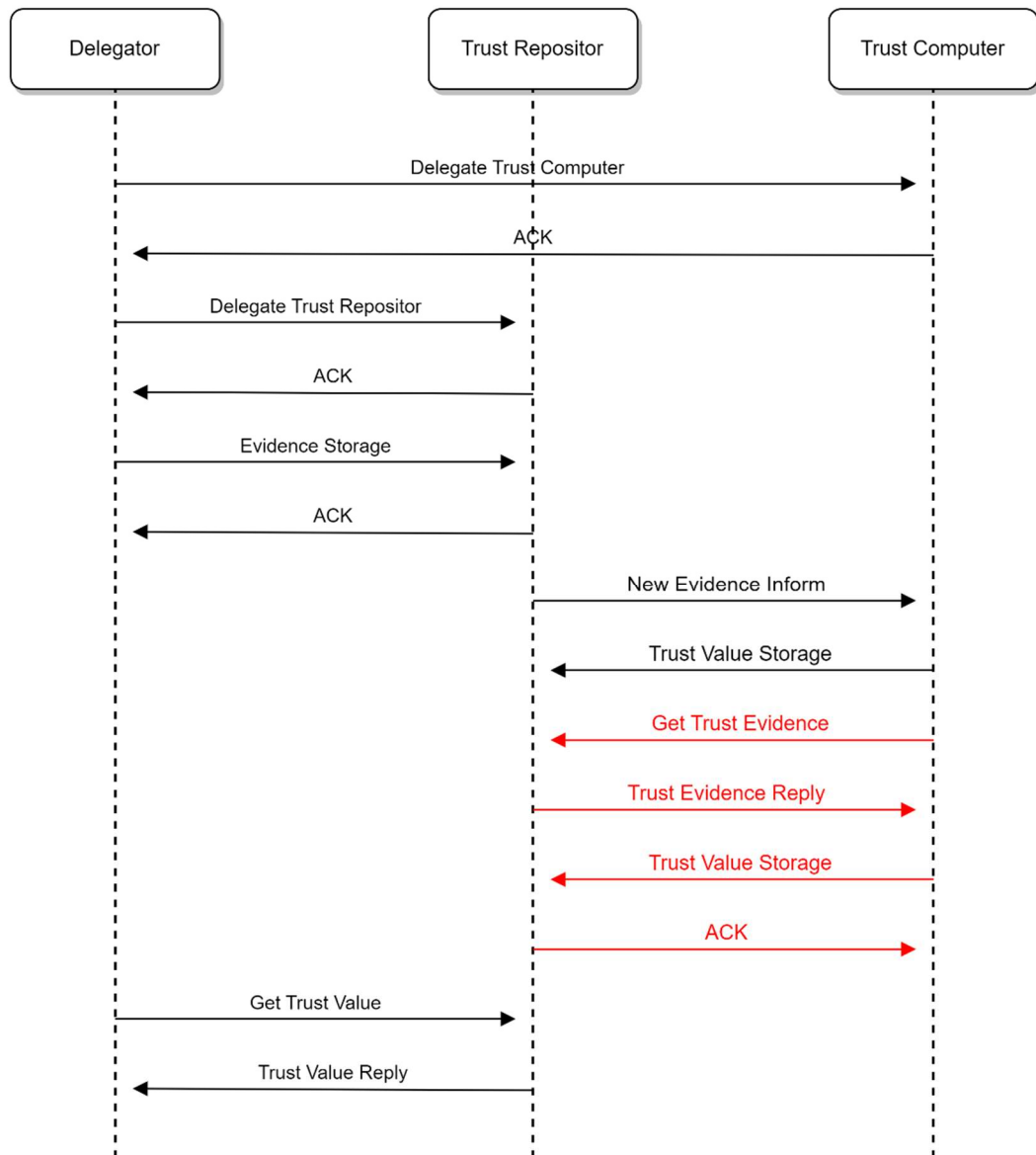


Figure 4.13: The process of delegating trust computer and trust repositor roles to multiple delegates

3. When the trust repositor receives new evidence from the delegator, it forwards the evidence to the trust computer using a *New Evidence Inform* message. Additionally, the trust computer can request trust evidence from the trust repositor by sending a *Get Trust Evidence* message. The trust repositor replies with a *Trust Evidence Reply* message with the requested trust evidence to the delegator.

4. Upon receiving the evidence from the trust repositior, the trust computer performs a trust update. It then sends the updated trust value to the trust repositior for storage using a *Trust Value Storage* message.
5. When the delegator requires the trust value, it sends a *Get Trust value* message to the trust repositior. The trust repositior retrieves the requested trust value from its database and sends it to the delegator in a reply message.

4.7 Access control for trust management delegation

Limiting access to trust values and trust evidence is crucial for a trust management delegation system that involves multiple participating entities. It ensures that entities are granted access to trust data based on their assigned roles, rather than exposing all trust data to all participants. Therefore, we adopt a role-based access control (RBAC) approach to assign permissions to the entities based on their roles within the system.

4.7.1 Roles

Roles define the actions and responsibilities of entities within a trust management delegation system. Therefore, we analyse the roles of the entities and classify them into 5 roles for access control as follows:

- Delegator (Owner)
- Trust computer
- Trust repositior
- Trust evidence collector
- Trust decision maker

4.7.2 Permissions

Permissions define the operations an entity can perform in the delegation system to access trust values and trust evidence. We define 3 permissions, including Read, Write, and Remove, as follows.

- Read – Access the content of trust value and trust evidence.
- Insert – Add the content of trust value and trust evidence.
- Delete – Remove stored trust value and trust evidence.

Update permission is unnecessary in the delegation system. Each new trust value and trust evidence is inserted into the databases as new records with a timestamp. Trust models require historical trust evidence to calculate new trust values. Moreover, some trust models integrate earlier trust values into the trust calculation.

4.7.3 Role-permission relationships

Role-permission relationships specify the permissions assigned to each role for accessing trust values and trust evidence. Permissions for trust values and trust evidence are handled separately, as shown in Table 4.2.

Table 4.2: Access permissions to trust value and trust evidence for different roles

Roles	Trust value	Trust evidence
Delegator (owner)	Read, Insert, Delete	Read, Insert, Delete
Trust Repositor	Read, Insert, Delete	Read, Insert, Delete
Trust evidence collector	-	Insert
Trust decision maker	Read	-
Trust computer	Read, Insert	Read
Other entities	-	-

4.7.4 Permission delegation

After a delegator, an owner of trust value and trust evidence, discovers a delegate to perform the trust repositore role on its behalf, the delegator informs the trust repositore about the permissions assigned to various entities. This is to specify who has what permissions to access trust values and trust evidence stored in the trust repositore's database. Figure 4.14 shows the process of delegating the permission of trust value and trust evidence access to the trust repositore.

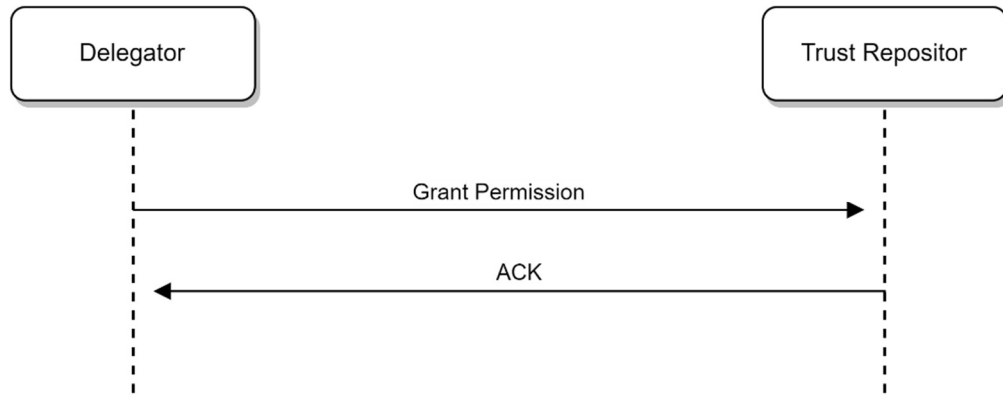


Figure 4.14: The process of trust value and trust evidence access permission delegation

1. The delegator sends a *Grant Permission* message with evidence to the trust repositior, containing a list of device identifiers and their assigned permissions for accessing trust values and trust evidence. The message includes the following parameters:
 - **Trustor ID:** IPv6 of the delegator (owner of the data).
 - **Device ID** = ['P1', 'P2']
 - **Device ID** = ['P1', 'P2']

Where P1 is permission for trust value access, and P2 is for trust evidence access.

2. The trust repositior acknowledges receipt of the *Grant Permission* message by sending an acknowledgement message back to the delegator. If the permission delegation of the delegator has already existed in the database, The repositior updates the permission in its database. The trust repositior then permits the entities to access trust value and trust evidence according to the permissions defined in the message.

4.7.5 Permission delegation revocation

After the delegation of trust repositior roles expires. The trust repositior will keep data and allow access by others before it revokes all permissions and removes the data. However, the delegator can revoke the permission granted to a specific entity before

the leasing period expires. The process of permission delegation revocation is shown in Figure 4.15 and described as follows.

1. The delegator sends a *Revoke Permission* message to the trust repositor with the IPv6 address of the delegator (owner of the data) to revoke the permissions.
2. The trust repositor processes the revocation request and replies to the delegator with an acknowledgement message to notify the status of the revocation.

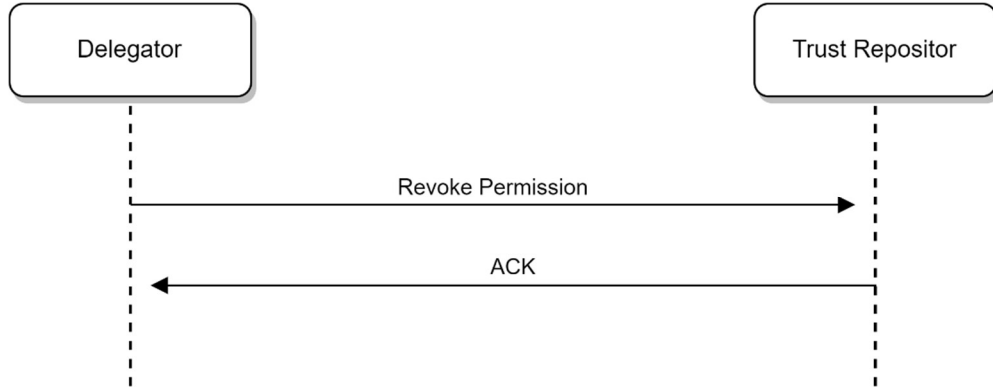


Figure 4.15: The process of trust value and trust evidence access permission delegation revocation

4.7.6 Role-permission relationships operation

In the proposed protocol, permissions are linked to delegation roles for accessing trust values and trust evidence rather than being granted independently, as shown Table 4.2. When a delegator assigns a delegation role to a delegate, the corresponding permissions are granted at the same time. A role is assigned when a delegation relationship is established and remains for the duration of that delegation until the delegation lease expires or is revoked by the delegator.

Rather than using an authentication mechanism, the proposed protocol uses IP addresses as node identities for permission verification. For example, when the delegator delegates permission to a trust repositor, it provides the IP address of the authorised requester along with the associated permission relationships to the trust repositor. The trust repositor stores this information in its local database for subsequent access verification. When a message requesting access to trust values or trust evidence is received. The trust repositor then verifies the IP address of the

incoming request against the stored permissions in the database to determine whether the requester is authorised to perform the requested operation on the trust data. Access is granted only if the IP address matches a record with the appropriate permission for that operation.

4.8 Recommendation sharing

Recommendations can be shared in the form of trust value and evidence. A delegator informs a delegate who is responsible for the trust repositor role that its data would be shared with others by indicating `share=1` in the delegation messages: *Delegate All Role*, *Delegate Trust Repositor* and *Delegate Trust Computer_Repositor*. However, the delegate shares the recommendation only with entities that also share their data with others. To verify this, the trust repositor checks the trustor IP address in a *GET Trust Evidence* message against the information in its database to confirm whether the requesting entity also shares its data. In case the delegate is responsible for all trust management roles, the delegate locally checks its database to see whether the delegator also shares the data with others.

4.9 Mobility management

In fog computing scenarios, end nodes can move from one location to another, whereas fog nodes are typically stationary. When an end node moves to a new location, it must find a new delegate and restart the trust management delegation process from the beginning. Moreover, it must begin collecting trust evidence from a starting point, which can be resource-intensive. To address this challenge, we propose a handover mechanism to manage the mobility of end nodes in the trust management delegation system.

The main objective of the proposed handover mechanism is to seamlessly transfer the delegation information of a delegator moving out of the coverage area of an old delegate to a new delegate so the delegator can continue performing the trust management delegation with the new delegate. The handover mechanism transfers the delegation information, trust evidence, and trust values belonging to the delegator from the old delegate to the new delegate without transferring through the delegator.

This reduces energy consumption for communication and other resources on resource-constrained end nodes.

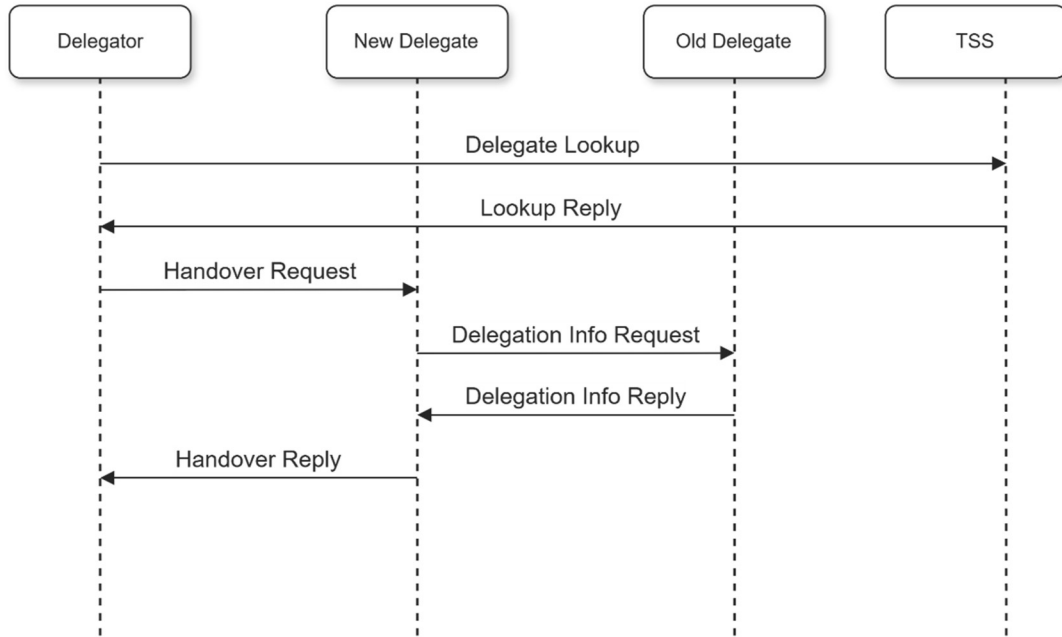


Figure 4.16: The process of trust management delegation handover

The handover procedure of trust management delegation from an old (previous) delegate to a new delegate in Figure 4.16 is explained in the following steps:

1. The delegator moves to another location, which is out of the coverage of the old delegate. Before disconnecting communication with the old delegate, it sends a *Delegate lookup* message to request the TSS for a new delegate that meets its delegation requirements. The TSS responds with a reply message with the IP address of a new delegate. If the delegator cannot contact the TSS, it repeats the TSS discovery operation.
2. The delegator then sends a *Handover Request* message to notify the new delegate that it needs to perform the handover of trust management delegation from the old delegate. The message includes the IP address of the old delegate, allowing the new delegator to request delegation information.
3. With the identifier of the old delegate received from the delegator, the new delegate sends a *Handover Info Request* message to request the transfer of trust management delegation information associated with the delegator. The old

delegate replies with delegation information, trust evidence, and trust values towards all trustees attached to a reply message to the new delegate.

4. After receiving the delegation information, the new delegate responds with a reply message to the delegator to notify the completion of the handover process to the delegator. The parameters included in this message are delegation information and relevant information as follows.
 - **Trustor ID:** An identifier of the delegator.
 - **Trustee ID:** An identifier of the trustee, or a list of trustee identifiers.
 - **Delegation roles:** A list of trust management roles.
 - **Lease time:** Delegation duration.
 - **Trust model:** A trust model for trust computation.
 - **Trust Value:** A list of trustee IP addresses with their trust values.
 - **Evidence:** A list of trust evidence with its values.
 - **Data Sharing:** Boolean 0 or 1 indicates recommendation sharing.
5. At this point, the handover process is completed. The delegator disconnects communication with the old delegate and continues trust management delegation tasks with the new delegate.

If the delegator has delegated trust management to multiple delegates, it might perform the handover operation to all delegates. However, the delegator can consider changing the delegation pattern to one delegate or multiple delegates after it contacts the TSS.

4.10 Conclusion

This chapter has presented the design of trust delegation components, the delegation procedure, exchanged messages, role-based access control, and mobility management used in the trust management delegation protocol. The purpose of the design is to transform the requirements of trust management in fog computing, which we have analysed, into the technical specifications and approach for the trust management delegation protocol. The protocol design in this chapter will be used in the implementation phase in Chapter 5.

Chapter 5: The implementation of trust management delegation protocol

This chapter aims to provide a proof-of-concept implementation to demonstrate that the trust management delegation protocol designed in the previous chapter can work in practice.

5.1 Why use Python and CoAP

Python was selected as the programming language to implement the proposed trust management delegation protocol due to its cross-platform compatibility. It ensures the protocol can operate across various operating systems and hardware platforms in fog computing environments without modification. Moreover, Python offers useful library support for protocol implementation, including specialised packages for networking (such as `asyncio` for asynchronous operations), database management (`SQLite3` for lightweight local data storage), and fuzzy logic (`scikit-fuzzy` for implementing trust computation models). Additionally, Python provides essential tools for data analysis and performance measurement through libraries, such as `matplotlib`, which are crucial for analysing the protocol evaluation in the next chapter. These libraries not only reduce development time but also facilitate maintenance and future extensions of the protocol.

Constrained Application Protocol (CoAP) [147] was chosen as the communication protocol for the trust management delegation protocol because it provides lightweight communication that is well-suited to resource-constrained devices. CoAP offers lightweight RESTful interactions similar to HTTP but with a much lower packet overhead [147]. CoAP over User Datagram Protocol (UDP) also minimises connection overhead and memory usage compared to CoAP over Transmission Control Protocol (TCP), making it particularly suitable for devices with limited communication, memory and processing resources [152]. Additionally, the protocol is standardised by the Internet Engineering Task Force (IETF) to ensure reliability and broad industry support, which are critical for integrating the delegation protocol into real-world fog computing systems. The protocol utilises the `aiocoap` library [13] to

implement CoAP. It is a Python library for CoAP development. The library is written in Python 3 and utilises asyncio for asynchronous support in concurrent network operations. It also supports various operating systems, including Windows, Raspbian, FreeBSD, and macOS. This ensures that the trust management delegation protocol can be deployed across diverse hardware in fog and IoT scenarios

5.2 Protocol message exchange

CoAP over UDP is utilised in the trust management delegation protocol for message exchange and data transmission between entities. Although CoAP over UDP does not guarantee reliable communication, it is considered more suitable than TCP-based protocols for fog computing. The following are the key reasons why CoAP over UDP outperforms CoAP over TCP in resource-constrained environments.

1. CoAP over UDP is simple, lightweight, connectionless, and does not require a handshake, which leads to significant resource savings for constrained devices.
2. CoAP over UDP headers has a size of 8 bytes, compared to CoAP over TCP headers, which start at 20 bytes and can increase with additional options. A smaller packet size reduces bandwidth usage and latency for data transmission.
3. With Confirmable (CON) messages in CoAP over UDP, recipients can decide which messages need an acknowledgement instead of acknowledging every data packet, thus avoiding unnecessary delays for less important data.
4. CoAP over UDP supports IP multicast, while TCP does not natively support multicast. This multicast feature enables trust service discovery in our protocol.

5.2.1 Message format

The CoAP message format is simple and lightweight to reduce message header overhead. Figure 5.1 depicts the CoAP message format over UDP that can be elaborated in detail as follows:

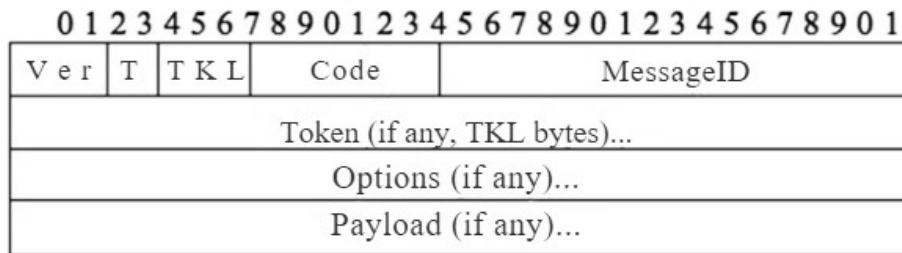


Figure 5.1: CoAP message format [147]

1. Header (4 Bytes)

1.1 Version Number (Ver) – is a 2-bit value that indicates the CoAP version.

The current version is 01 for RFC 7252, other values are reserved for future versions.

1.2 Message type (T) – contains 2 bits that specify the message type, Confirmable (0), non-confirmable (1), Acknowledgement (2), or Reset (3) as mentioned in the previous section.

1.3 Token Length (TKL) – 4 bits which represent the length of the token field, ranging from 0 to 8 bytes.

1.4 Code – contains 8 bits that indicate the request method codes or the response codes. The request method codes are 0.01 (GET), 0.02 (POST), 0.03 (PUT), 0.04 (DELETE). The response codes are, for example, 2.01 (Created) or 4.04 (Not Found).

1.5 Message ID (MID) – is a 16-bit value used to detect message duplication and manage retransmissions in the case of Confirmable messages, matching an acknowledgement with a Confirmable message or a Reset message with a non-confirmable message.

2. Token – the size varies between 0 and 8 bytes based on the value in the Token Length field. It is used to correlate requests and responses. For example, in a piggy-backed response, a Confirmable message must have the same Message ID as an acknowledgement, and the Token value of both messages must match. In a separate response, only the Token of the response and request messages must match.

3. Options – defines CoAP options that can be specified in a message. Common CoAP options include Uri-Path and Content-Format. Uri-Path identifies the resource path, e.g., “sensor/data”. Content-Format indicates the representation media type of the payload being transmitted, e.g., “application/json”.
4. Payload - represents the actual content or data included in a CoAP message. Payload has a prefix marker (0xFF) to indicate the beginning of the payload.

5.2.2 Message types

There are four types of messages in the CoAP protocol that are utilised in the delegation protocol:

1. Confirmable message (CON): is designed for scenarios where reliable delivery is essential. It requires the recipient to send an acknowledgement message back to the sender to confirm the receipt of the message.
2. Non-confirmable message (NON): is used when timely messaging is more important than reliability. This type of message does not require reliable delivery but still has a Message ID for duplicate detection. This means that the recipient does not send an ACK message. However, each NON message includes a unique Message ID for duplicate detection.
3. Acknowledgement message (ACK): is a message that is sent by the recipient in response to a received Confirmable (CON) message. The ACK confirms successful receipt and, if necessary, may also carry a payload containing the outcome of the requested operation or additional trust information.
4. Reset message (RST): is generated when a recipient receives a CON or NON message that it cannot process, typically due to internal errors.

5.2.3 CoAP methods

Similar to HTTP protocol, CoAP supports four basic methods, namely GET, POST, PUT, and DELETE, to perform operations on a URI resource in a CoAP server.

They can be elaborated as follows.

- GET: is used to retrieve information or data that currently corresponds to the resource URI indicated in the message from a server.

- POST: is used to request the server to process information or data that is enclosed within the message URI. It results in the creation of a new resource or the update of an existing resource on the server.
- PUT: is used for creating or updating the information identified in the request URI like POST method. However, PUT method is idempotent while POST is not. When sending the same request multiple times, the result will remain the same.
- DELETE: is used to request the server to delete the information or data that is identified in the resource URI.

5.2.4 Message payload

JavaScript Object Notation (JSON) is a lightweight, text-based, language-independent data interchange format that has become widely adopted across numerous programming languages. JSON is commonly used in web applications and distributed systems for storing, transmitting, and exchanging data. In the context of our trust management delegation protocol, JSON was selected to represent data in the message payload for both request and response messages. This decision ensures seamless and interoperable data exchange among devices in fog computing environments.

Each parameter in the message payload consists of a key paired with a value that can be a string, number, array, Boolean, etc. This flexible data representation makes JSON particularly suitable for fog computing scenarios where devices with varying capabilities and different operating systems need to communicate effectively. The lightweight nature of JSON significantly reduces message size, bandwidth usage, and computational overhead compared to other formats like XML. The following example demonstrates how parameters in a message payload can be formatted using JSON.

```

{
  "Delegate_IP": "2a01:4b00:ea2e:ed00:2391:df94:11ab:4397",
  "Lease_time": "36800",
  "Trust_model": ["TC","TR"],
  "E_storage": ["availability","task_completion_rate","response_time"],
  "E_collector": ["availability","task_completion_rate","response_time"],
  "D_maker": ["TD"]
}

```

5.2.5 Response message codes

There is a range of response codes in CoAP that allow a CoAP server to respond to requests from CoAP clients. However, the following CoAP response codes are used in the proposed delegation protocol to indicate successful operations, errors, and other relevant statuses.

- 2.01 Created: indicates that the request was successful and resulted in the creation of a new resource. For example, when a delegate successfully registers delegation services with the TSS, the TSS responds with a 2.01 Created code to acknowledge the successful service registration.
- 2.02 Deleted: means that the resource specified in the request URI has been successfully deleted. An example within the protocol is when a delegate deregisters its delegation services, the TSS returns a 2.02 Deleted response code to the delegate to indicate successful deregistration.
- 2.03 Changed: is used to acknowledge successful completion of a POST or PUT request. For instance, when a delegate sends a PUT request to renew its delegation lease time, the TSS replies with a 2.03 Changed response to indicate that the lease has been renewed.
- 2.05 Content: is used in response to GET successful requests with the relevant data included in the response payload. For example, when a delegator requests trust evidence or a trust value, a 2.05 Content response is returned along with the requested information.
- 4.03 Forbidden: means that the client's request was rejected by the server due to insufficient access rights. For instance, if an unauthorised node attempts

to access trust evidence stored by the trust repositor, the trust repositor will deny the request and reply with a 4.03 Forbidden response.

- 4.04 Not Found: is used to indicate that the requested resource does not exist on the server. For example, the TSS returns a 4.04 Not Found response when the TSS is unable to find a delegate that matches the delegation criteria specified by the delegator
- 5.00 Internal Server Error: indicates that the server encounters an unexpected error that prevented it from fulfilling the request. For example, if a database failure occurs during delegate registration, the TSS would respond with a 5.00 Internal Server Error.

5.3 Protocol repository

SQLite [153] is used as a database engine to store trust values, trust evidence, delegation information, and other relevant data within the delegation protocol. SQLite is an open-source embedded database management system, particularly well-suited for resource-constrained environments due to its lightweight, serverless architecture. One of its main advantages is that it stores the entire database in a single file on the local device without the need for a separate server process or complex configuration, unlike traditional database management systems such as MySQL and SQL Server. In addition, SQLite is compatible with Python through the Python standard library. By leveraging SQLite, the trust management delegation protocol achieves high resource efficiency and reliability. This ensures that resource-constrained devices can effectively participate in trust management operations. We also utilise phpLiteAdmin [154] for managing SQLite databases via a web-based user interface. It is a very lightweight tool compared to phpMyAdmin, but it provides essential operations, features, interfaces and user experience similar to phpMyAdmin.

Table 5.1: Database for each delegation role

Role	Repository
TSS	Delegation_Registration
Trust computer	Delegation
Evidence collector	Delegation Trust_evidence
Trust decision maker	Delegation
Trust repositor	Delegation Trust_value Trust_evidence Access_permission

The database structure implemented in the delegation protocol is designed to be simple, straightforward and efficient. There are five databases, each corresponding to specific delegation roles within the protocol, as shown in Table 5.1. The description of these repositories is as follows:

1. *Delegation_Registration*: is present only in a TSS to store delegation information registered with the TSS by delegates. It consists of 6 tables: *Service_Registration*, *Roles*, *Trust_model*, *E_storage*, *E_collector* and *Decision_making_method*.
2. *Delegation*: is present in all delegate nodes to store the delegation information. It includes 4 tables: *Delegation_info*, *Roles*, *E_storage* and *E_collector*
3. *Trust_value*: is only in a trust repositor to store trust values. It contains a single table namely *Trust_value*.
4. *Trust_evidence*: is only in a trust repositor and an evidence collector to store trust evidence. It has a single table, namely *Trust_evidence*.
5. *Access_permission*: is only in a trust repositor to store the permission for access control. It has a single table, namely *Access_permission*.

Instead of providing explicit data type and size declarations for each column, SQLite determines the column type dynamically based on the actual stored value. Therefore, the declaration of a specific data type and size for each column is unnecessary. SQLite supports five different primitive data types, commonly referred to as storage classes: NULL, INTEGER, REAL, TEXT and BLOB. The attributes and data types used in each table are depicted in Tables 5.2 – 5.11 below.

TABLE: *Service_Registration*

Description: *Service_Registration* stores registered delegation services

Table 5.2: Service_Registration table

Attribute	Data type	Description
Reg_id	INTERGER	Registration ID, auto generated
Delegate_IP	TEXT	IPv6 of a delegate
Lease_time	INTERGER	Delegation duration time, time value stored in the number of seconds since 1970-01-01 00:00:00 UTC.
Roles	INTERGER	Delegation roles, This field links to Roles table.
Trust_model	INTERGER	Trust model. This field links to Trust_model table.
E_storage	INTERGER	Evidence storage. This field links to E_storage table.
E_collector	INTERGER	Evidence collection. This field links to E_collector table.
D_maker	INTERGER	Decision-making types supported by a delegate. This field links to D_maker table.
timestamp	INTERGER	Registration time. It is stored in the number of seconds since 1970-01-01 00:00:00 UTC

TABLE: *Delegation_info*

Description: *Delegation_info* stores delegation information

Table 5.3: *Delegation_info* table

Attribute	Data type	Description
Delegation_id	INTERGER	Delegation ID, auto generated
Trustor_IP	TEXT	IPv6 of a trustor
Trustee_IP	TEXT	IPv6 of a trustee
Roles	INTERGER	Delegation Roles. This field links to Roles table.
Lease_time	INTERGER	Delegation duration time. Time value stored in the number of seconds since 1970-01-01 00:00:00 UTC.
Trust_model	TEXT	Trust model used for trust calculation.
Trust_update	TEXT	Trust update method, event-driven or time-driven
E_storage	INTERGER	Evidence that needs a delegate to store. This field links to E_storage table.
E_collector	INTERGER	Evidence that needs a delegate to collect. This field links to E_collector table.
Trust_evidence	TEXT	Trust evidence
Share	INTERGER	Indicate sharing recommendations to others
timestamp	INTERGER	Delegation time. It is stored in the number of seconds since 1970-01-01 00:00:00 UTC

TABLE: *Trust_value*

Description: *Trust_value* stores trust evidence

Table 5.4: Trust_value table

Attribute	Data type	Description
TValue_id	INTERGER	Trust value ID, auto generated
Trustor_IP	TEXT	IPv6 of a trustor
Trustee_IP	TEXT	IPv6 of a trustee
Trust_value	REAL	Trust_value stored in decimals.
timestamp	INTERGER	Time to insert trust value. It is stored in the number of seconds since 1970-01-01 00:00:00 UTC

TABLE: *Trust_evidence*

Description: *Trust_evidence* stores trust evidence. Evidence value can be an integer, a decimal, or a Boolean. Therefore, the attribute Evidence_value does not require a data type because a single column in SQLite can store mixed data types.

Table 5.5: Trust_evidence table

Attribute	Data type	Description
Evidence_id	INTERGER	Evidence ID, auto generated
Trustor_IP	TEXT	IPv6 of a trustor
Trustee_IP	TEXT	IPv6 of a trustee
Evidence_name	TEXT	Evidence name.
Evidence_value	-	Evidence value
timestamp	INTERGER	Time to insert trust value. It is stored in the number of seconds since 1970-01-01 00:00:00 UTC

TABLE: Roles

Description: *Roles* is divided from the *Delegation_Registration* or *Delegation* databases to avoid duplicate rows. In *Delegation_Registration*, it stores the roles supported by a delegate. In *Delegation*, it stores the roles that are delegated to a delegate.

Table 5.6: Roles table

Attribute	Data type	Description
Role_id	INTERGER	Role ID, auto generated
Role	TEXT	TC = Trust Computer, TR = Trust Repositor, EC = Evidence Collector and DM = Trust Decision Maker

TABLE: Trust_model

Description: *Trust_model* stores trust model for trust calculation supported by a delegate.

Table 5.7: Trust_model table

Attribute	Data type	Description
TModel_id	INTERGER	trust model ID, auto generated
Trust_model	TEXT	Trust model

TABLE: E_storage

Description: *E_storage* is decomposed from the *Delegation_Registration* or *Delegation* databases to avoid duplicate rows. In *Delegation_Registration*, *E_storage* stores the evidence name supported by a delegate. In *Delegation*, it stores delegated evidence.

Table 5.8: E_storage

Attribute	Data type	Description
Estorage_id	INTERGER	Evidence storage ID, auto generated
Evidence	TEXT	Evidence name

TABLE: *E_collector*

Description: *E_collector* is also decomposed from the *Delegation_Registration* or *Delegation* databases. In *Delegation_Registration*, *E_collector* stores evidence supported by a delegate for collection. In *Delegation*, it stores delegated evidence for collection.

Table 5.9: *E_collector* table

Attribute	Data type	Description
Ecollector_id	INTERGER	Evidence collector ID, auto generated
Evidence	TEXT	Evidence name

TABLE: *Decision_making_method*

Description: *Decision_making_method* stores the decision-making method.

Table 5.10 *Decision_making_method* table

Attribute	Data type	Description
DMaking_id	INTERGER	decision-making ID, auto generated
Decision_making	TEXT	decision-making method, TD or TBD

TABLE: *Access_permission*

Description: *Access_permission* stores the permission for trust values and evidence access control.

Table 5.11: *Access_permission* table

Attribute	Data type	Description
P_id	INTERGER	Permission ID, auto generated
IP_address	TEXT	IP_address that is granted
Permission	TEXT	Permission: Read, Insert, Delete
Trustor_IP	TEXT	IPv6 of a trustor

5.4 Trust update

Trust update determines how trust values towards trustees are updated. In the proposed delegation protocol, two trust update approaches are supported: time-driven and event-driven. In a time-driven approach, trust values are recalculated at fixed intervals. The

delegator specifies the desired update interval during the trust delegation process. The delegate then schedules trust value updates according to this interval. This method is suitable for scenarios where continuous monitoring is needed or events occur regularly. In contrast, the event-driven approach updates trust values in response to specific actions or events. Each time the delegate finishes an interaction with a trustee, it immediately triggers a recalculation and update of the corresponding trust value. This strategy ensures that trust values are up to date following activities. In our implementation, the delegator specifies the preferred trust update approach during the delegation setup phase.

5.5 Delegation expiration and lease renewal

To ensure that delegation services remain current and valid and to prevent outdated or inactive delegate nodes in the systems, the proposed delegation protocol employs a lease management mechanism. During the registration process, each delegate specifies a lease duration with their delegation service information within the TSS. The TSS records the lease expiration time in its database as a UNIX timestamp elapsed since 1970-01-01 00:00:00 UTC. The TSS periodically compares the stored lease expiration times with the current system time. If the lease for a delegate expires, the TSS promptly removes the corresponding delegation registration from its database.

The delegation protocol adopts ideas from the Dynamic Host Configuration Protocol (DHCP) lease renewal process, which is designed to provide reliable network address assignment with low communication overhead. In this approach, a delegate automatically initiates the lease renewal process when its lease duration reaches 50% of its validity period. If the delegate does not successfully renew its lease with the TSS by the time the lease reaches 87.5%, it then attempts to perform TSS discovery using either pull-based or push-based methods to renew its lease. If the renewal is unsuccessful and the lease expires, the TSS will withdraw all delegation services provided by the delegate. At this point, the delegate must re-register its delegation services.

5.6 Device identifier

Various IoT device identifiers are used to identify entities in IoT environments. These identifiers uniquely identify objects or devices across different domains. A serial number [64] is the simplest form of unique identifier, assigned by each manufacturer to differentiate between similar objects. However, duplicates can occur, especially with devices from different manufacturers. A universally unique identifier (UUID) [155] is a 128-bit value generated by an algorithm based on a timestamp and other factors, such as the network address, making it highly likely to be globally unique. However, they can be easily altered by users to mimic or clone on the network.

Physically unclonable functions (PUFs) [156] exploits variations in manufacturing and fabrication processes to generate a unique device key or identifier. Since these variations are not fully controllable during manufacturing, each chip has distinct physical properties that cannot be copied or cloned. PUF generation schemes require specialised hardware modules for implementation. TPM is widely used in most PCs, laptops, tablets and smartphones. Although TPMs are nearly impossible to forge or transfer to another device and are stored in a way that prevents modification, they are not integrated in certain IoT devices, such as Raspberry Pis. For these devices, an additional TPM module is required. An IoT-ID proposed in [157] is based on the concept of PUF without the requirement of any additional hardware support. IoT-ID is derived from basic system components, such as clock oscillator and Analog-to-Digital Converters (ADCs), which are present in every microcontroller. However, there is less industry support and standardisation compared to other identifiers. An X.509 certificate is a widely used digital document that ensures secure communication and authentication in a network. However, resource-constrained devices in fog computing systems still struggle to generate the X.509 certificate. It also requires a CA to manage the certificate and maintain trust within the Public Key Infrastructure (PKI).

Based on the analysis of various IoT device identifiers, IP addresses are justified as suitable device identifiers for the trust management delegation protocol. Although IP addresses can be spoofed and do not provide a secure identifier, they are universally available and natively supported across all types of networked devices. This ensures

compatibility in heterogeneous environments without the need for additional hardware or software. Moreover, using IP addresses has no extra computational or memory overhead, allowing resource-constrained devices to effectively participate in the trust management delegation protocol. In our implementation, we utilise IPv6 addresses as device identifiers and will investigate other identifier options to further enhance the efficiency of the trust management systems in future work.

5.7 Service discovery functions

5.7.1 Delegation service registration

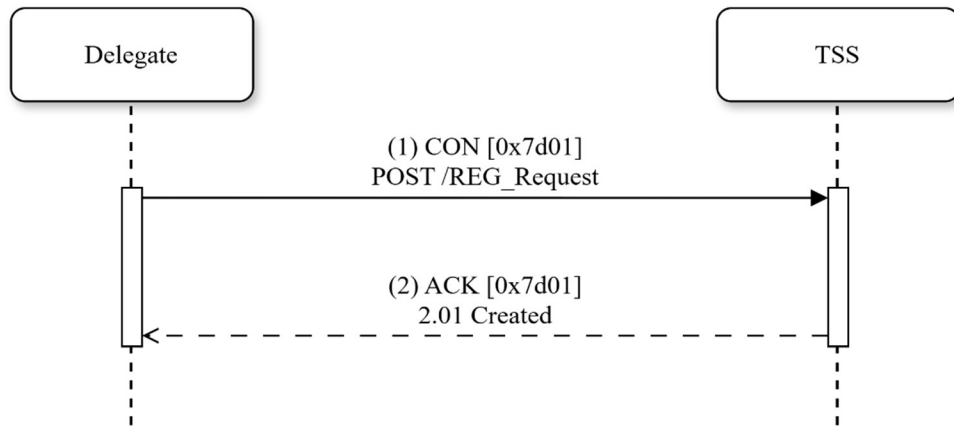


Figure 5.2: Delegation service registration

To register delegation services, a TSS continuously listens for registration requests from delegates. A delegate sends a POST message with “/REG_Request” in the URI-Path to register its delegation services with the TSS, as shown in Figure 5.2. The information related to the delegation service is included in the message payload in JSON format as below:

POST /REG_Request:

- **Delegate_IP:** IPv6 of the delegate.
- **Lease_time:** Delegation time in seconds.
- **Roles:** Delegation roles, where: TC = Trust Computer, TR = Trust Repositor, EC = Evidence Collector and DM = Trust Decision Maker.

- **Trust_model:** Trust model used for trust calculation, e.g., M1 = weighted sum, M2 = fuzzy logic. This field is left blank if the delegate does not provide the Trust Computer role.
- **E_storage:** A list of trust evidence names. This field is left blank if the delegate does not provide the Trust Repositor role.
- **E_collector:** A list of trust evidence names. This field is left blank if the delegate does not provide the Evidence Collector role.
- **D_maker:** Types of decision-making supported by the delegate: TD or TBD.

The TSS stores the delegation information in its database and sends an acknowledgement piggybacked with the response to the delegate.

- **2.01 “Created”** – If the registration is successful. A registration reference is included in the acknowledgement message payload. The registration reference is an 8-bit number used to reference the registration.
- **5.00 “Internal Server Error”** – If an issue occurs during registration (e.g., database failure). In this case, the delegate can retry sending the registration request after a while.

5.7.2 Delegation service deregistration

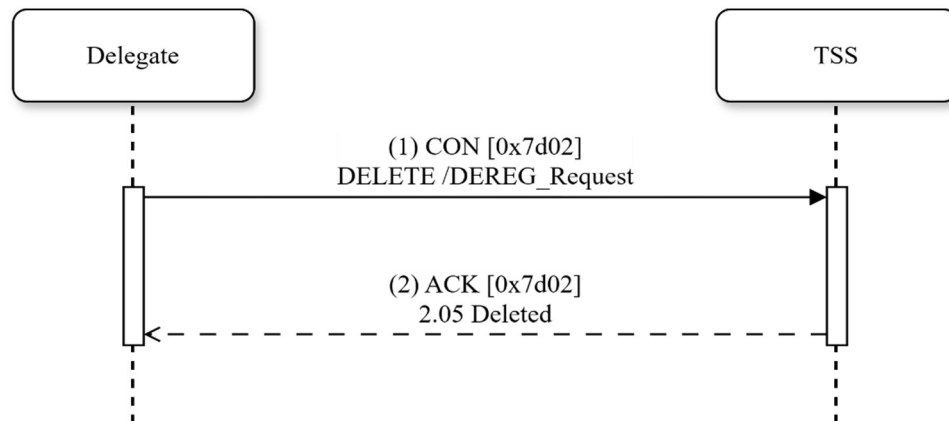


Figure 5.3: Delegation service deregistration

In Figure 5.3, the delegate requests the deregistration of delegation services from the TSS by sending a DELETE message with the URI-Path “/DEREG_Request”. The

message includes the registration reference number and the trust management roles to be deregistered in the payload, as follows:

DELETE /DEREG_Request:

- **Ref:** The registration reference number.
- **Roles:** The delegation roles to be deregistered. This field can be left blank to indicate that all roles are deregistered.

The TSS then withdraws the delegation service from its database. Afterwards, it returns an acknowledgement message with the following response codes:

- **2.02 “Deleted”** – If the deregistration is successful.
- **4.04 “Not Found”** with a message “Reference not found” or “Delegation roles not found” – If the TSS cannot find the registration reference or the specified delegation roles, respectively.
- **5.00 “Internal Server Error”** – If anything goes wrong with the deregistration.

5.7.3 Registration expiration and lease renewal

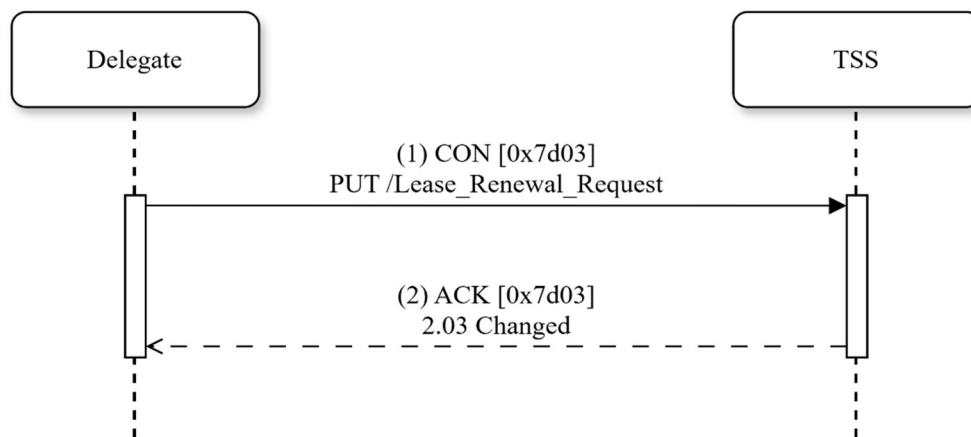


Figure 5.4: Delegation registration lease renewal

Figure 5.4 illustrates the process for renewing the delegation lease. The delegate sends a PUT request with the URI-Path “/Lease_Renewal_Request” to renew the delegation lease. The message payload includes the registration reference number and the new lease duration, as shown below:

PUT /Lease_Renewal_Request:

- **Ref:** The registration reference number.
- **Lease_time:** The new delegation time, in seconds.

The TSS updates the lease time in its database and sends an acknowledgement with the following piggybacked response code.

- **2.03 “Changed”** – If the lease renewal is successful.
- **4.04 “Not Found”** with the message “Reference not found” – If the TSS cannot locate the registration reference.

5.7.4 Delegate lookup

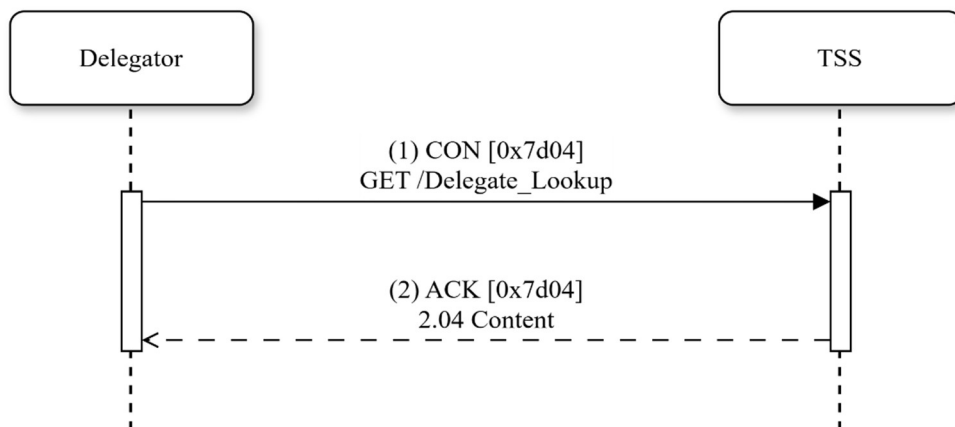


Figure 5.5: Delegate lookup

Figure 5.5 shows how the delegator looks up delegate(s) from the TSS. The delegator sends a Confirmable GET message with the URI-Path “/Delegate_Lookup” to the TSS. The payload of the message contains the delegate specification, as shown below:

GET /Delegate_Lookup:

- **Roles:** Delegation roles.
- **Trust_model:** Trust model used for trust calculation. This field is blank if the delegator does not delegate the Trust Computer role.
- **E_storage:** A list of trust evidence names for storage. This field is blank if the delegator does not delegate the Trust Repositor role.
- **E_collector:** A list of trust evidence names for collection. This field is blank if the delegator does not delegate the Evidence Collector role.

- **D_maker:** Decision-making types required by the delegator, TD or TBD.

The TSS searches its database for the registered delegate(s) that match the provided specification. To perform the search, an SQL statement is used to query the database and retrieve the delegate(s) that meet the requirements. There might be more than one delegate that meets the requirement. The TSS gathers the results and sends an acknowledgement message to the delegator with one of the following response codes:

- **2.05 “Content”** – If the TSS finds delegate(s) that meet the delegator’s requirements.

The response payload contains the IP address of the delegate(s). Information for multiple delegates is included in a single JSON structure, represented as an array of objects. Each object contains the details for one delegate.

- **4.04 “Not Found”** – with a message “No delegates found” – If the TSS cannot find any delegates matching the required specifications.

5.7.5 TSS discovery

There are two approaches to finding a TSS, namely the pull and push approaches. The pull approach allows a delegator or delegate to broadcast a message to search for the TSS, while the push method allows the delegator or delegate to receive the broadcast message from the TSS.

1. Pull approach

Figure 5.6 illustrates the pull TSS discovery approach, where a delegator or delegate uses UDP multicast to broadcast a message to all devices on the local network to locate a TSS. All TSS nodes must join a multicast group to receive packets sent to that group. However, the delegator or delegate need not be part of that group to send the packet. Routers between the delegator or delegate and the TSS nodes duplicate packets and forward multiple copies to all TSS nodes.

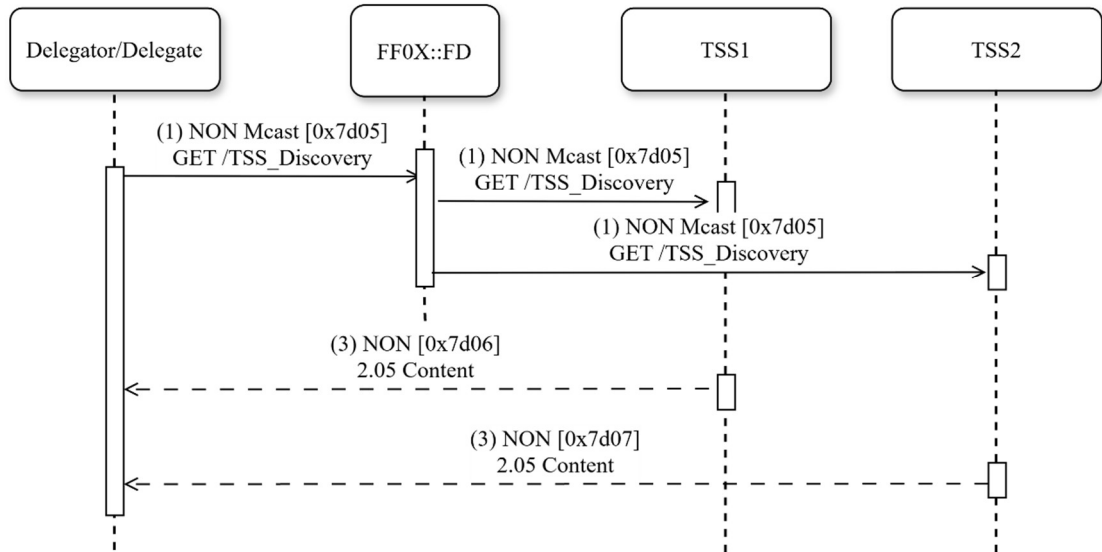


Figure 5.6: Pull TSS discovery

The delegator or delegate sends a non-confirmable GET request to the IPv6 multicast address for “All CoAP Nodes” at the link-local scope, FF02::FD. This address allows the delegator or delegate to perform discovery only within the local subnet, which is sufficient to find a nearby TSS. The GET request includes the URI-path “TSS_Discovery”, indicating the pull-based TSS discovery approach.

Any TSS that receives the request replies with a non-confirmable 2.05 “Content” response. The response payload includes the IP address of the TSS. The delegator or delegate might receive multiple responses from multiple TSS nodes. However, it accepts and immediately processes the first response message received and ignores any subsequent messages to minimise delay.

2. Push approach

The push TSS discovery approach is illustrated in Figure 5.7. Unlike the pull approach, where the delegator or delegate broadcasts a packet to find a TSS, this approach allows the TSS to advertise packets to delegators and delegates. To receive these packets, delegators or delegates must join a multicast group, which they can dynamically join or leave at any time.

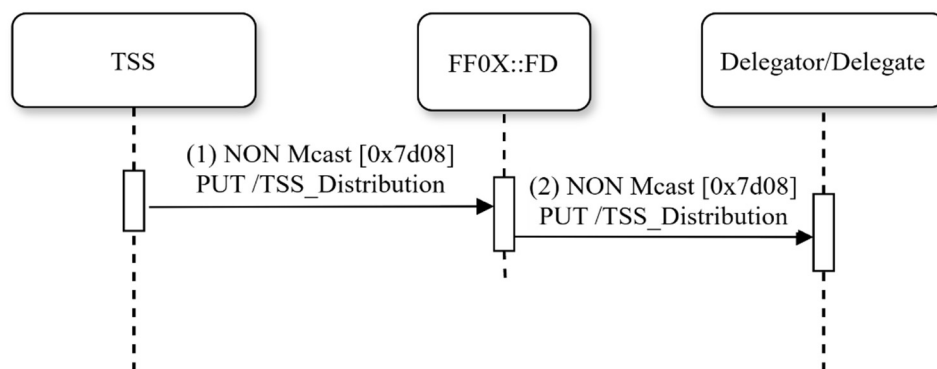


Figure 5.7: Push TSS discovery

The advertising intervals can be configured based on desired performance, battery life and application scenario. For instance, intervals can be as short as 1 millisecond to improve discoverability or as long as 10 seconds to conserve energy and reduce network congestion when discovery speed is less critical. This parameter can be adjustable. However, we configure TSS advertising at 1-second intervals for balancing broadcasting speed and power consumption.

The TSS advertises a non-confirmation PUT request with the URI-path “TSS_distribution” to indicate the push TSS discovery approach. Any delegator or delegate that has joined the multicast group will receive the request and immediately start contacting the TSS directly to access its services.

5.8 Trust management delegation functions

5.8.1 Delegating all trust management roles to one delegate

How a delegator delegates all of the trust management roles can be shown in Figure 5.8 which consists of the following tasks.

1. Delegating all trust management roles

The delegator sends a trust management delegation request using a POST request with the URI-Path “/Delegate_All_Roles” to the delegate. This delegation approach allows the delegator to include multiple trustees in a single request. However, all trustees must have the same delegation information.

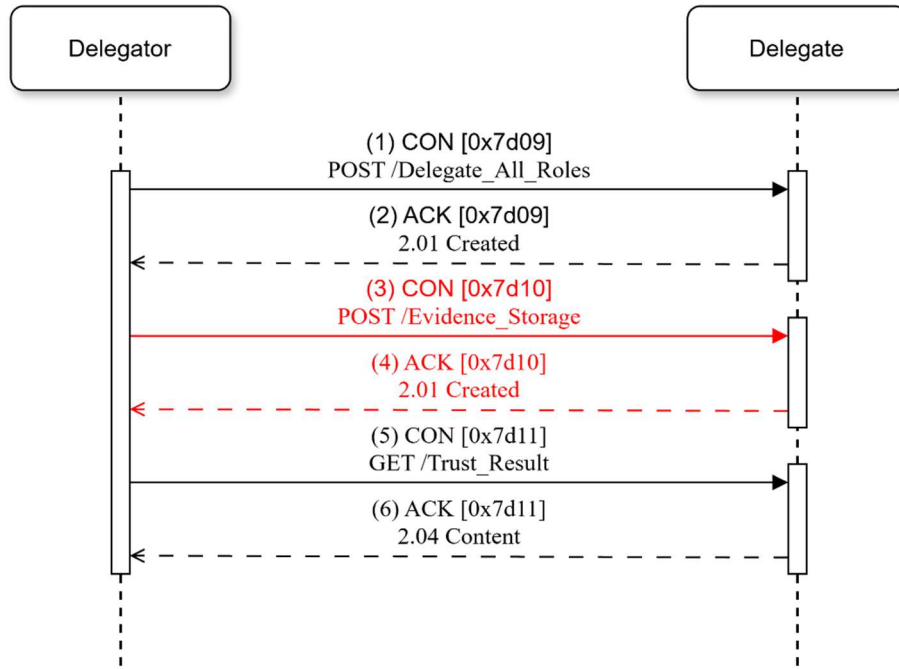


Figure 5.8: Delegating all roles to one delegate

The parameters included in the payload are as follows:

POST /Delegate_All_Roles:

- **Trustor_IP:** IPv6 address of the delegator.
- **Trustee_IP:** IPv6 address of a trustee, or a list of IPv6 addresses if multiple trustees are included.
- **Lease_time:** Delegation time, in seconds.
- **Trust_model:** Trust model used for trust calculation.
- **Trust_update:** Trust update scheme, event-driven or time-driven. If time-driven, update duration in seconds is included.
- **Evidence:** A list of trust evidence names.
- **Share:** Boolean value (0 or 1) to indicate sharing recommendation with others (as described in Chapter 4). If set to 1, the delegate only shares recommendations with entities that also share their data.

The delegation information is saved in the delegate's database. The delegate then sends a response piggybacked in an acknowledgement message back to the delegator

2.01 “Created” – if the delegation is successful. If the request fails, the delegate responds with 5.00 “Internal Server Error”.

If the delegation is successful, the delegate begins collecting evidence towards the trustee(s), storing trust evidence in its database and updating trust based on the selected trust model. If the delegation is unsuccessful, the delegator can resend the delegation request.

2. The delegator sends trust evidence to the delegate.

Some types of evidence, such as latency, cannot be delegated. The delegator must collect this evidence itself and send it to the delegate. In this case, the delegator attaches the evidence in a POST message with the URI-Path “/Evidence_Storage” and sends it to the delegate, as shown in red. Evidence collection for a single trustee is treated as a thread, allowing the delegator to collect and send evidence for multiple trustees simultaneously. The parameters included in the payload are as follows:

POST /Evidence_Storage:

- **Trustor_IP:** IPv6 address of the delegator.
- **Trustee_IP:** IPv6 address of the trustee.
- **Evidence:** A list of trust evidence with corresponding values.

Upon the receipt of the evidence, the delegate stores it in its database and sends an ACK message with a response code to the delegator:

- **2.01 “Created”** – If the evidence is successfully stored. No content is included in the payload.
- **4.03 “Forbidden”** – If the delegator does not have the right permission to insert trust evidence.

3. Requesting trust result

To request a trust result towards trustee(s), the delegator sends a GET request with the URI-Path “/Trust_Result”. The payload includes the following parameters:

GET /Trust_Result:

- **Trustor_IP:** IPv6 address of the delegator.

- **Trustee_IP:** IPv6 address of a trustee, or a list of trustees' IPv6 addresses.
- **Threshold:** A threshold value for trust decision-making based on TD (Trust Decision).

If using TD, the Trustee_IP field contains the IPv6 address of a specific trustee, and the delegate compares the trustee's trust value with the provided threshold to determine whether the trustee is "trusted" or "untrusted."

If using TBD (Trust-Based Decision), the threshold parameter is blank. A list of IPv6 addresses in the Trustee_IP field is required. The delegate selects the trustee with the highest trust value and returns their IPv6 address.

In reply to the delegator, the delegate sends an acknowledgement message with response 2.05 "Content" code if the delegator has permission to access the trust value. The response payload includes the trust result ("trusted" or "untrusted") for TD, or the IPv6 of the selected trustee for TBD.

5.8.2 Delegating all trust management roles to multiple delegates

If no single delegate is available to perform all trust management roles. The TSS replies to the delegator with information about multiple delegates, compressed in JSON format within the payload, that can fulfil all trust management roles. This delegation approach involves four delegate roles: an evidence collector, a trust repositior, a trust computer and a decision maker, as shown in Figure 5.9.

1. Delegating all trust management roles

The delegator prepares POST requests with URI-Path "/Delegate_Trust_Computer", "/Delegate_Trust_Repositor" and "/Delegate_Evidence_Collector", and sends them to respective delegates, namely trust computer & decision maker, trust repositior and evidence collector, as shown in the steps 1-6 in Figure 5.9. The decision-making and trust computation roles are performed by the same delegate as in our design. This delegation approach also supports delegating multiple trustees simultaneously.

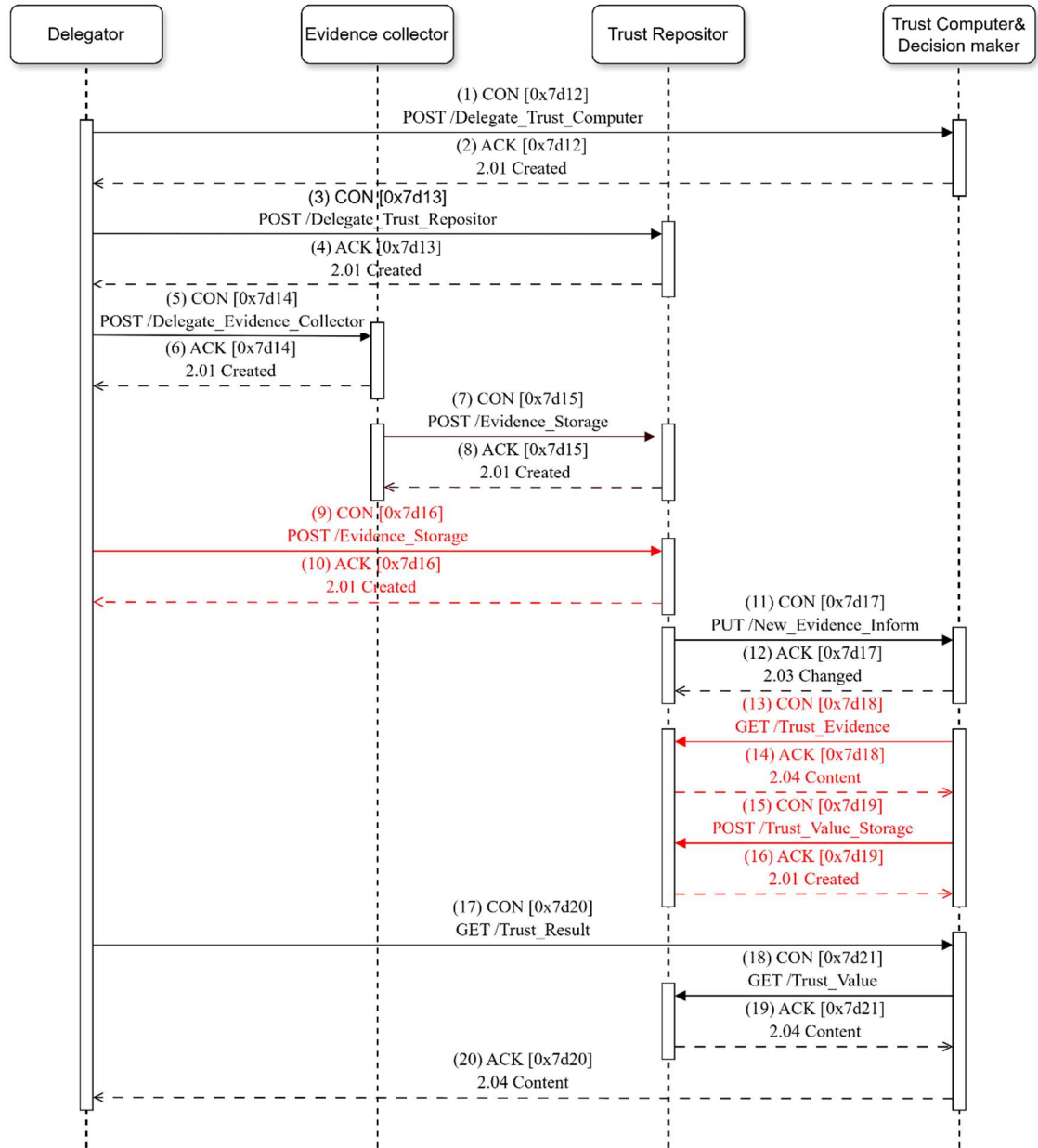


Figure 5.9: Delegating all roles to multiple delegates

Each message request contains parameters as follows:

POST /Delegate_Trust_Computer:

- **Trustor_IP:** IPv6 address of the delegator.
- **Trustee_IP:** IPv6 address of a trustee, or a list of IPv6 addresses if delegating a group of trustees.

- **Trust_repositor_IP:** IPv6 address of the trust repositor, used to request trust evidence from the trust repositor for trust update.
- **Lease_time:** Delegation time, in seconds.
- **Trust_model:** Trust model used for trust calculation.
- **Trust_update:** Trust update method, either event-driven or time-driven (if time-driven, specify the update interval in seconds).
- **Evidence:** A list of trust evidence.

POST /Delegate_Trust_Repositor:

- **Trustor_IP:** IPv6 address of the delegator.
- **Trustee_IP:** IPv6 address of a trustee, or a list of IPv6 addresses if delegating a group of trustees.
- **Trust_computer_IP:** IPv6 address of trust computer, used to inform the trust computer of new trust evidence for trust update.
- **Lease_time:** Delegation duration, in seconds.
- **Evidence:** A list of trust evidence.
- **Share:** Boolean 0 or 1 to indicate whether to share recommendations with others. If set to 1, the delegate shares recommendations only with entities that also share their data.

POST /Delegate_Evidence_Collector:

- **Trustor_IP:** IPv6 address of the delegator.
- **Trustee_IP:** IPv6 address of a trustee, or a list of IPv6 addresses if delegating a group of trustees.
- **Trust_repositor_IP:** IPv6 address of the trust repositor, used to send collected trust evidence to the trust repositor.
- **Lease_time:** Delegation duration, in seconds.
- **Evidence:** A list of trust evidence.

Each delegate stores the delegation information in its database and replies to the delegator with an acknowledgement message and a response code 2.01 “Created” if the delegation is successful. Otherwise, it responds with 5.00 “Internal Server Error”.

2. Trust evidence collection and storage

The evidence collector collects evidence towards the trustee(s) and sends it in JSON format to the trust repositor using a POST request with URI-Path “/Evidence_Storage” as shown in steps 7-8. Additionally, the delegator uses the POST message to send evidence that cannot be delegated, which the delegator collects itself, to the trust repositor in the process 9-10. The parameters included in the message payload are as follows:

POST /Evidence_Storage:

- **Trustor_IP:** IPv6 address of the delegator.
- **Trustee_IP:** IPv6 address of the trustee.
- **Evidence:** A list of trust evidence with values.

To confirm receipt of the evidence, the trust repositor replies to the evidence collector or the delegator with an acknowledgement message containing one of the following response codes: 2.01 “Created” if the evidence storage is successful, or 4.03 “Forbidden” if access is denied.

3. Updating trust

To update trust, the trust computer can use one of two approaches. In the first approach, as shown in the process 11-12, the trust repositor informs the trust computer of new trust evidence using a PUT request with the URI-Path “/New_Evidence_Inform”. The payload of the message consists of a list of trust evidence and their corresponding values, as follows:

PUT /New_Evidence_Inform:

- **Trustor IP:** IPv6 address of the delegator.
- **Trustee IP:** IPv6 address of the trustee.
- **Evidence:** A list of trust evidence and their corresponding value.

Upon receiving the message, the trust computer calculates a new trust value based on the new evidence provided. It then piggybacks the new trust value for storage in the trust repositor onto an acknowledge message with response code 2.04 “changed”, which is sent back to the trust repositor. If the trust calculation process takes a long

time and the new trust value cannot be piggybacked in the acknowledgement, the trust computer acknowledges the confirmable message first and then sends a separate confirmable message with a 2.05 “Content” response containing the new trust value.

In the second approach, as shown in the process 13-16 in red colour, the trust computer requests trust evidence from the trust repositior for trust update. The trust computer sends a GET confirmable message with the URI-Path “/Trust_Evidence” to request trust evidence from the trust repositior. The payload contains trustor and trustee’s IP addresses, and a list of evidence names required to update trust between the trustor and trustee. If the request for trust evidence is successful, the trust repositior replies with the trust evidence piggybacked in a 2.04 “Content” response. However, before providing evidence, the trust repositior must verify the permissions of the trust computer. If the trust computer does not have the necessary permission to access the evidence, a 4.03 “Forbidden” response is sent. The details of the “/Trust_Evidence” message are as follows:

GET /Trust_Evidence:

- **Trustor_IP:** IPv6 address of the delegator.
- **Trustee_IP:** IPv6 address of the trustee.
- **Evidence:** A list of trust evidence names.

Afterwards, the trust computer performs trust calculation and sends the updated trust value to the trust repositior for storage. A POST message with the URI-Path “/Trust_value_Storage” is used to carry the new trust value. To confirm successful storage of the trust value, the trust repositior responds with an acknowledgement message with a code 2.01 “Created”.

POST /Trust_value_Storage:

- **Trustor_IP:** IPv6 address of the delegator.
- **Trustee_IP:** IPv6 address of the trustee.
- **Trust_value:** The updated trust value after trust update.

4. Requesting trust result

The delegator can require a trust result towards a trustee, as shown in steps 17-20. It requests the trust result by sending a GET request with the URI-Path “/Trust_Result” to the trust computer. The trust computer queries the trust value of the trustee(s) from the trust repositor by sending a GET request with the URI-Path “/Trust_value”. The parameters in the payload are as follows:

GET /Trust_value:

- **Trustor_IP:** IPv6 address of the delegator.
- **Trustee_IP:** IPv6 address of a trustee, or a list of IPv6 addresses for multiple trustees.

After verifying that the trust computer has the appropriate permissions, the trust repositor sends an acknowledge message with a 2.01 “Content” including the required trust value or a list of trustees and their trust values piggybacked in the payload.

Upon receiving the trust value, the trust computer performs decision-making based on either TD or TBD, depending on the case. It then sends the result piggybacked in the acknowledgement message of the GET request. The trust computer responds to the delegator with 2.05 “Content” if the trust result request is successful. The response message includes the trust result or the IPv6 address of the selected trustee in the payload.

5.8.3 Delegating evidence collection to a delegate

A delegator can delegate only evidence collection to a delegate to collect essential trust information against trustees on its behalf. The process of this delegation is shown in Figure 5.10 and can be explained as follows.

1. Delegating evidence collector role

To delegate the evidence collector role, the delegator sends a confirmable POST request with the URI-Path “/Delegate_Evidence_Collector” to the evidence collector. The parameters included in the payload are as shown below. However, the Trust Repositor IP parameter is left blank because the evidence collector does not send the evidence to be stored in the trust repositor. The delegator can delegate evidence

collection for a group of trustees, but the same evidence types must be collected towards all trustees.

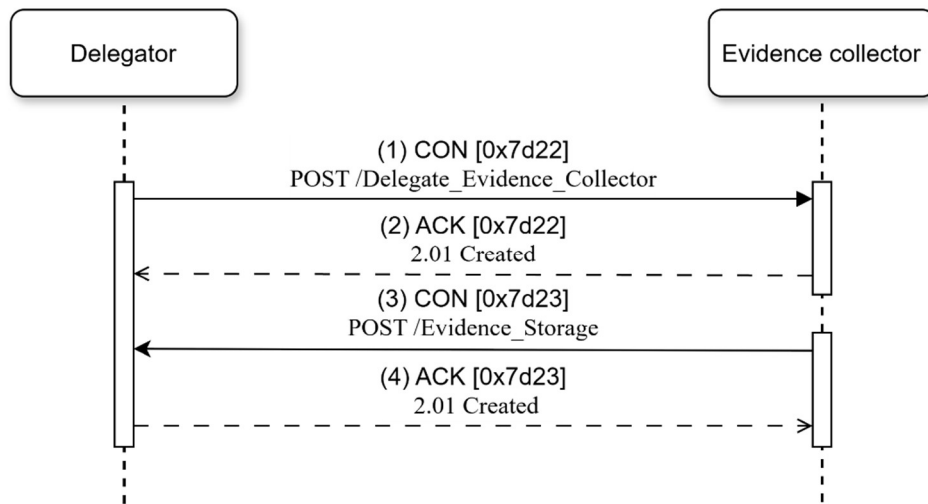


Figure 5.10: Delegating evidence collector role

POST /Delegate_Evidence_Collector:

- **Trustor IP:** IPv6 address of the delegator.
- **Trustee IP:** IPv6 address of a trustee, or a list of IPv6 addresses if delegating a group of trustees.
- **Trust repositor IP:** IPv6 address of the trust repositor (left blank in this delegation).
- **Lease time:** Delegation time, in seconds.
- **Evidence:** A list of trust evidence.

The evidence collector processes the request and responds with an appropriate acknowledgement 2.01 “Created” if the delegation is successful.

2. Evidence collection and storage

The evidence collector collects trust evidence on behalf of the delegator and sends it either immediately or at scheduled intervals. To send the evidence back to the delegator, the evidence collector uses a POST request with the URI-Path “/Evidence_Storage”, which includes the evidence in the payload. The delegator

responds to the evidence collector with an acknowledgement message containing 2.01 “Created” after the evidence storage is completed.

5.8.4 Delegating trust repositor to a delegate

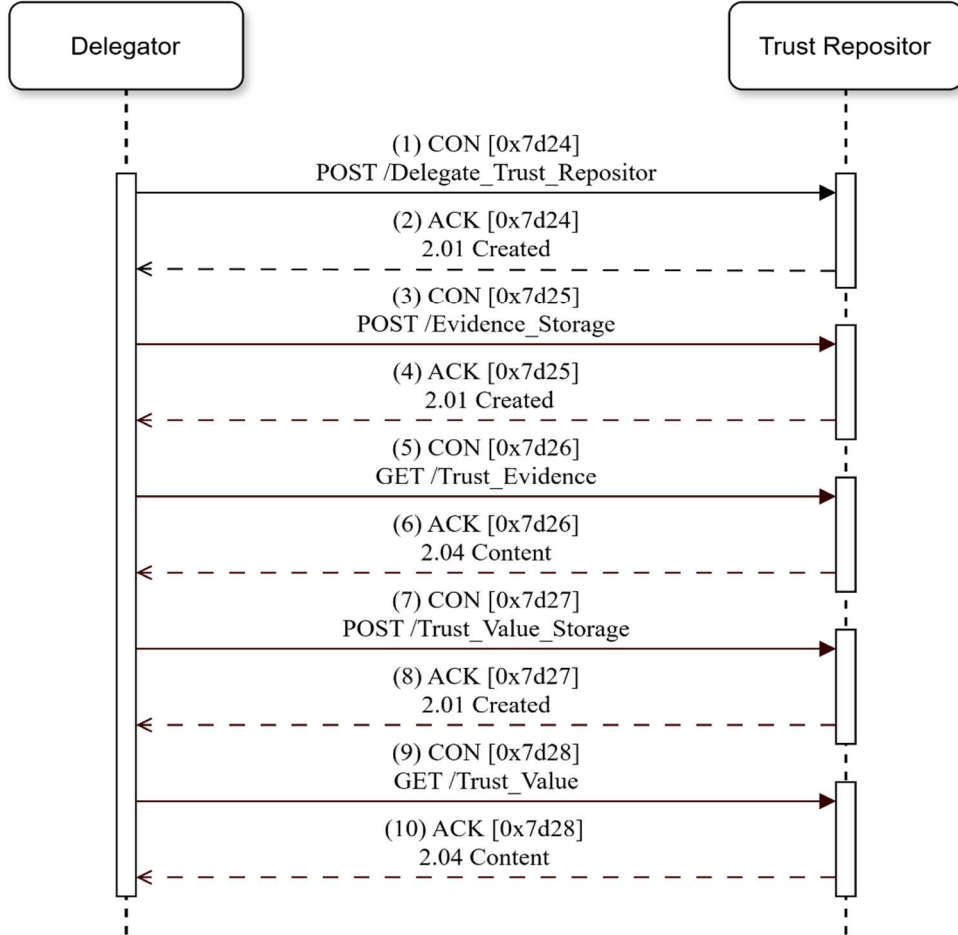


Figure 5.11: Delegating trust repositor role

Figure 5.11 shows the process of delegating a trust repository to a delegate, which is explained in detail below.

1. Delegate trust repositor roles

To delegate the trust repository role, the delegator first sends a POST message with the URI-Path “/Delegate_Trust_Repositor” and includes the delegation information below in the payload to the trust repository. Because the delegate does not inform the trust computer of new trust evidence to the trust computer. The Trust_computer_IP field is left blank.

POST /Delegate_Trust_Repositor:

- **Trustor_IP:** IPv6 address of the delegator.
- **Trustee_IP:** IPv6 address of a trustee, or a list of IPv6 addresses if delegating a group of trustees.
- **Trust_computer_IP:** IPv6 address of trust computer (left blank in this delegation).
- **Lease_time:** Delegation duration, in seconds.
- **Evidence:** A list of trust evidence.
- **Share:** Boolean 0 or 1 indicates whether to share recommendations with others.

The trust repositor prepares its database to store trust value and trust evidence on behalf of the delegator. It then sends a response with either 2.01 “Created” if the operation is successful or 5.00 “Internal Server Error” if an error occurs.

2. Storing and retrieving trust evidence

In this delegation, the delegator collects trust evidence itself. It sends the evidence to be stored in the trust repositor and can later retrieve it to perform trust updates. These processes are illustrated as steps 3-6.

To store trust evidence, the delegator includes the evidence in the payload of a POST message with the URI-Path “/Evidence_Storage” to send it to the trust repositor for storage.

POST /Evidence_Storage:

- **Trustor_IP:** IPv6 address of the delegator.
- **Trustee_IP:** IPv6 address of the trustee.
- **Evidence:** A list of trust evidence with corresponding values.

After successfully receiving and storing the evidence, the trust repositor replies with an acknowledgement message containing a response code 2.01 “Created” if the evidence storage is successful or 4.03 “Forbidden” if the delegator does not have the right permission to the evidence.

The delegator can retrieve the trust evidence from the trust repositor by sending a GET message with the URI-Path “/Trust_Evidence” to request the required trust evidence for updating trust.

GET /Trust_Evidence:

- **Trustor_IP:** IPv6 address of the delegator.
- **Trustee_IP:** IPv6 address of the trustee.
- **Evidence:** A list of trust evidence names.

The trust repositor replies with an acknowledgement message to the delegator containing a response code: 2.04 “Content” if the trust evidence request is successful, with the trust evidence piggybacked in the payload or 4.03 “Forbidden” If access is denied.

3. Storing and retrieving trust value

Similarly, the delegator can send the updated trust value to be stored in the trust repositor after completing trust updates and can later retrieve it when needed. These processes are illustrated as steps 7-10.

After completing the trust update, the delegator transfers the new trust value to the trust repositor for storage via a POST request with the URI-Path “/Trust_Value_Storage”.

POST /Trust_value_Storage:

- **Trustor_IP:** IPv6 address of the delegator.
- **Trustee_IP:** IPv6 address of the trustee.
- **Trust Value:** The updated trust value after trust update.

In response to the request, the trust repositor sends an acknowledgement message with a response code 2.01 “Created” or 4.03 “Forbidden”.

When the delegator needs to estimate the trustworthiness of a trustee, it sends a GET request with the URI-Path “/Trust_Value” to retrieve the trust value from the trust repositor.

GET /Trust_value:

- **Trustor_IP:** IPv6 address of the delegator.
- **Trustee_IP:** IPv6 address of a trustee, or a list of IPv6 addresses for multiple trustees.

In response to the request, the trust repositor retrieves the required trust value from its database and includes it in the payload of the acknowledgement message with a response code 2.01 “Content”. However, the delegator can access only its own trust values. If the requested trust value is not accessible, the trust repositor responds with 4.03 “Forbidden”.

5.8.5 Delegating trust computer to a delegate.

A delegator uses the following processes to delegate trust computer role to a single delegate, as shown in Figure 5.12.

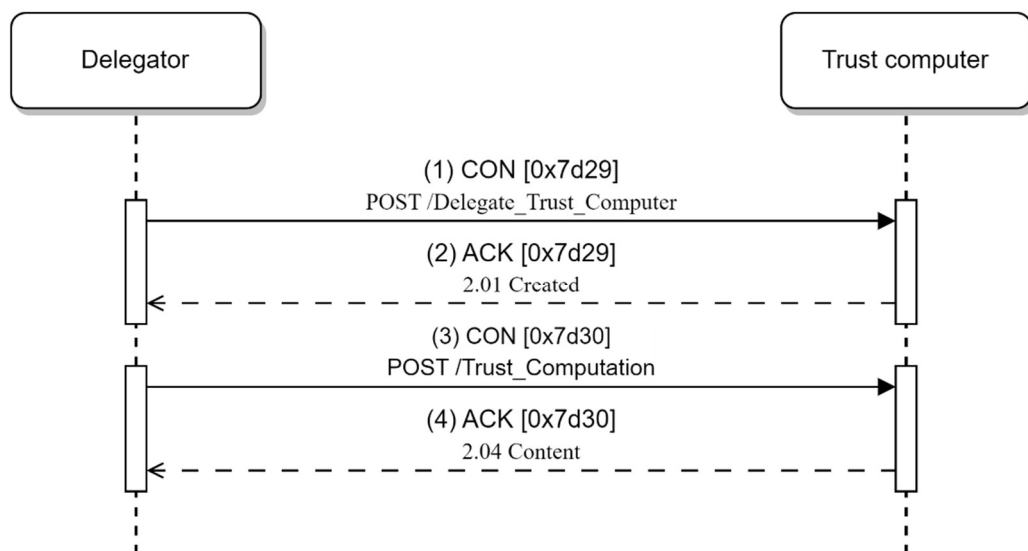


Figure 5.12: Delegating trust computer role

1. Delegating trust computer role

To delegate trust computer role, the delegator sends a POST request with the URI-Path “/Delegate_Trust_Computer and the parameters below to the delegate. However, the Trust repositor IP parameter is left blank because the trust computer does not request trust evidence from a trust repositor for updating the trust.

POST /Delegate_Trust_Computer:

- **Trustor_IP:** IPv6 address of the delegator.
- **Trustee_IP:** IPv6 address of a trustee, or a list of IPv6 addresses if delegating a group of trustees.
- **Trust_repositor_IP:** IPv6 of the trust repositor (left blank in this case).
- **Lease_time:** Delegation time, in seconds.
- **Trust_model:** Trust model used for trust calculation.
- **Trust_update:** trust update scheme (event-driven or time-driven). If time-driven, the update duration in seconds must be specified.
- **Evidence:** A list of trust evidence.

The trust computer responds to the delegator with an acknowledgement containing a response code 2.01 “Created” or 5.00 “Internal Server Error”, depending on the case, to the delegator.

2. Trust calculation

After successful delegation, the delegator sends trust evidence related to the trustee to the trust computer for trust calculation. A POST request with the URI-Path “/Trust_Computation” is sent with trust values and trust evidence in the message payload as follows.

POST /Trust_Computation:

- **Trust_value:** Trust value
- **Evidence:** A list of trust evidence with corresponding values.

The trust computer calculates a trust value based on the sent evidence and the trust model specified in the delegation request. Once the calculation is complete, it sends the trust value back to the delegator in the payload of an acknowledgement message with the response code 2.01 “Content”. However, if the trust calculation takes longer, the trust computer sends an acknowledgement message to the request and later sends a separate message with the response code 2.05 “Content”, including the new trust value in the payload.

5.8.6 Delegating trust computer and trust repositior roles to one delegate

Figure 5.13 illustrates the process of delegating both the trust computer and trust repositior roles to a single delegate. The delegation process can be illustrated as follows.

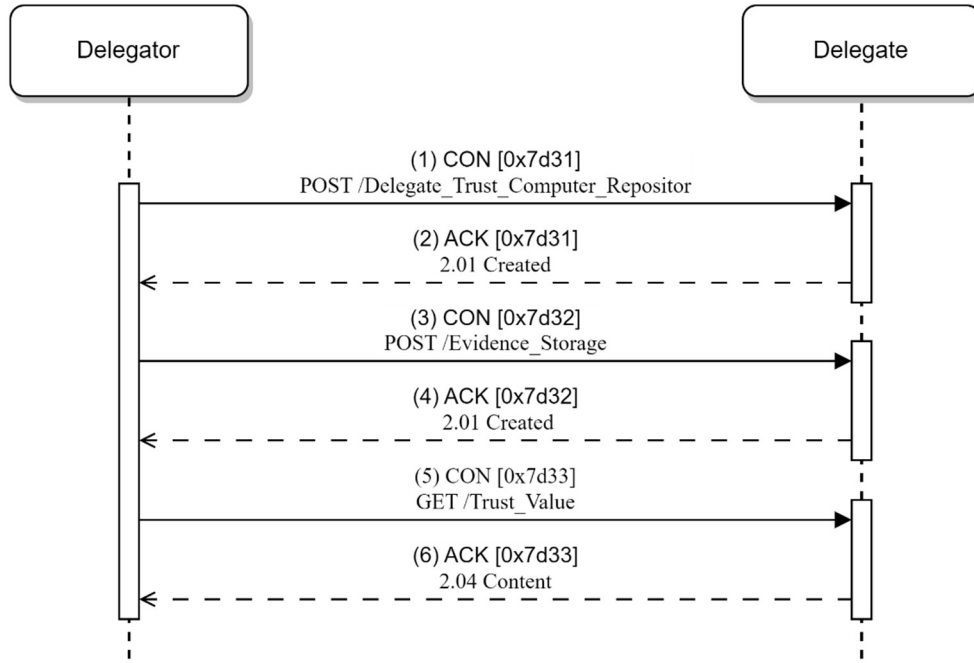


Figure 5.13: Delegating trust computer and repositior roles to one delegate

1. Delegating trust computer and repositior roles

The delegator initiates the delegation by sending a POST request with the URI-Path “/Delegate_Trust_Computer_Repositor” to the delegate. The request payload contains the following delegation information.

POST /Delegate_Trust_Computer_Repositor:

- **Trustor_IP:** IPv6 address of the delegator.
- **Trustee_IP:** IPv6 address of a trustee, or a list of IPv6 addresses if delegating a group of trustees.
- **Lease_time:** Delegation time, in seconds.
- **Trust_model:** A trust model used for trust computation.
- **Trust_update:** Trust update scheme (event-driven or time-driven). If time-driven, the update duration in seconds must be specified.

- **Evidence:** A list of trust evidence.
- **Share:** Boolean 0 or 1 to indicate sharing recommendations with others.

The delegate stores the delegation information in its database and responds to the delegator with an acknowledgement message containing the response code: 2.01 “Created” or 5.00 “Internal Server Error”.

2. Trust evidence storage

After completing the delegation, the delegator begins collecting evidence towards the trustee itself and sends the evidence to the delegate for storage. A POST message with the URI-Path “/Evidence_Storage”, including the trust evidence in the payload, is used to transfer the evidence from the delegator to the delegate.

POST /Evidence_Storage:

- **Trustor_IP:** IPv6 address of the delegator.
- **Trustee_IP:** IPv6 address of the trustee.
- **Evidence:** A list of trust evidence with corresponding values.

Upon receiving the evidence, the delegate sends an acknowledgement to the POST message with response code 2.01 “Created” if the evidence storage is successful or 4.03 “Forbidden” if the delegator does not have the required permission to store the evidence. The delegate then updates the trust and stores the trust evidence and the updated trust value in its database. However, the trust value could be updated using a time-driven approach. In this case, the delegate stores the trust evidence in its database and performs the trust update later.

3. Requesting trust result

To query a trust value from the delegate, the delegator sends a GET request with the URI-Path “/Trust_Value”, including the IPv6 address of a trustee, or a list of IPv6 addresses, the same as in Section 5.8.4. The delegate retrieves the trust value from its database and sends a response code 2.04 “Content” with the required trust value in the payload.

5.8.7 Delegating trust computer and trust repositor roles to multiple delegates

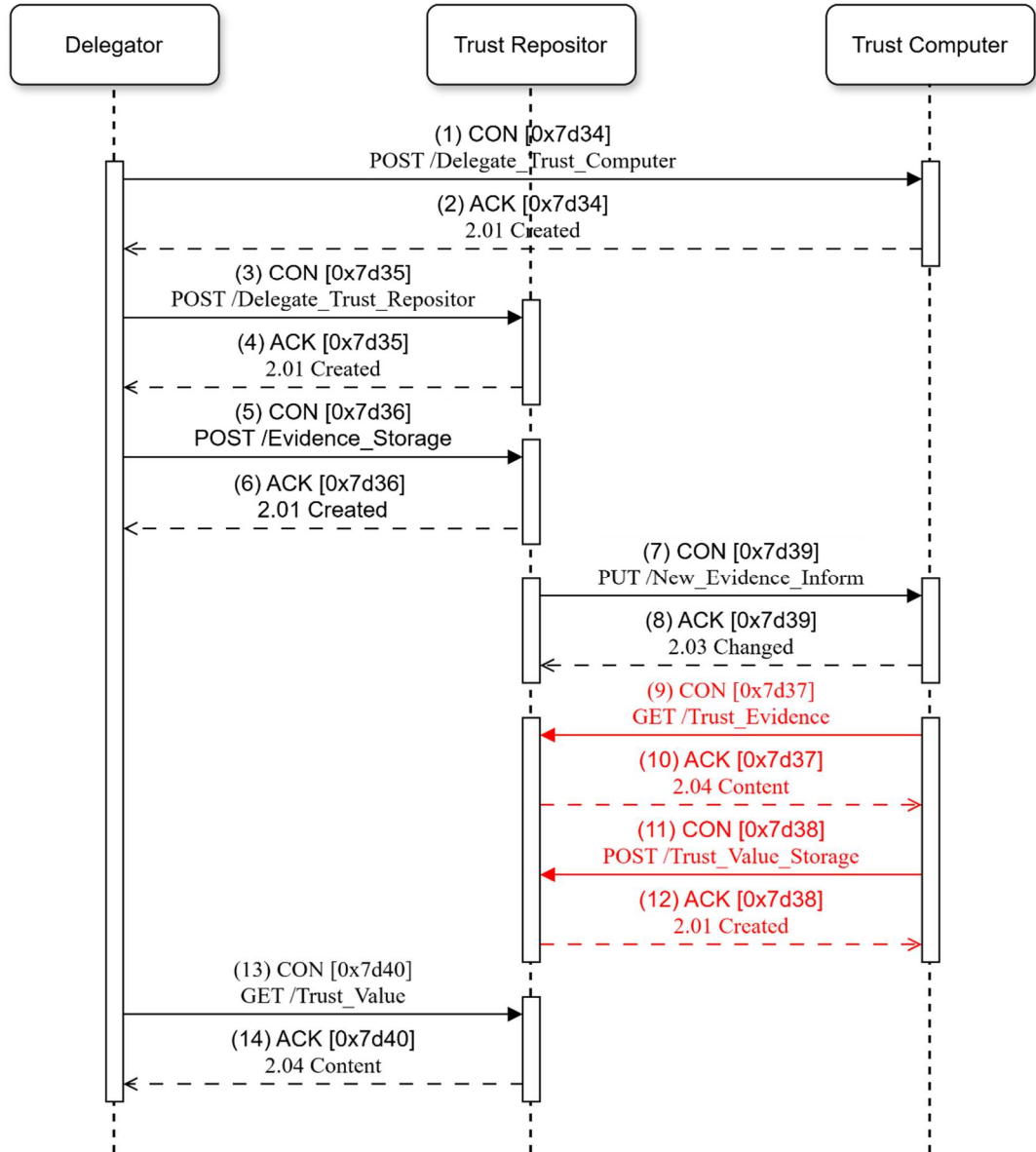


Figure 5.14: Delegating trust computer and repositor roles to multiple delegates

The process of delegating trust computer and trust repositor roles to multiple delegates consists of 4 tasks, as shown in Figure 5.14. They can be explained as follows.

1. Delegating trust computer and trust repositor

To begin, the delegator sends delegation information to the trust computer and trust repositor, as shown in steps 1-4. A POST request with the URI-Path.

“/Delegate_Trust_Computer” is used to carry the trust computer role delegation information to the trust computer. Simultaneously, a POST message with the URI-Path “/Delegate_Trust_repositor” is sent to the trust repositor to delegate the trust repositor role.

POST /Delegate_Trust_Computer:

- **Trustor_IP:** IPv6 address of the delegator.
- **Trustee_IP:** IPv6 address of a trustee, or a list of IPv6 addresses in case the delegator delegates a group of trustees.
- **Trust_repositor_IP:** IPv6 of the trust repositor.
- **Lease_time:** Delegation time, in seconds.
- **Trust_model:** Trust model used for trust calculation.
- **Trust_update:** Trust update scheme, event-driven or time-driven. If it is time-driven, the update duration in seconds must be specified.
- **Evidence:** A list of trust evidence.

POST /Delegate_Trust_Repositor:

- **Trustor_IP:** IPv6 address of the delegator.
- **Trustee_IP:** IPv6 address of a trustee, or a list of IPv6 addresses if delegating a group of trustees.
- **Trust_computer_IP:** IPv6 address trust computer.
- **Lease_time:** Delegation time, in seconds.
- **Evidence:** A list of trust evidence.
- **Share:** Boolean 0 or 1 to indicate sharing recommendations with others.

To acknowledge the receipt of the requests, both the trust computer and repositor response with a response code 2.01 “Created” or 5.00 “Internal Server Error” when the delegation is successful or unsuccessful, respectively.

2. Trust evidence storage

Due to the delegator collecting trust evidence towards a trustee on its own, the delegator uses a confirmable POST request with the URI-Path “/Evidence_Storage”

to send the trust evidence for storage in the trust repositor. The trust evidence, in JSON format, is included in the payload, the same as in Section 5.8.6.

In response, the trust repositor sends an acknowledgement message to the delegator with a response code 2.01 “Created”, 4.03 “Forbidden” or 5.00 “Internal Server Error”, depending on the outcome.

3. Updating trust

The trust repositor informs the trust computer when new evidence is received from the delegator. A PUT message with the URI-Path “New_Evidence_Inform” carries the trust evidence to the trust computer for updating the trust. The trust computer then calculates a new trust value and sends it back to the trust repositor for storage. The new trust value is included in the acknowledgement of the POST message or sent as a separate confirmable message. This process is the same as in Section 5.8.2 to inform the new trust evidence to the trust computer.

Alternatively, as in steps 9-12 in Figure 5.14 that highlights in red, the trust computer can request trust evidence from the trust repositor. A GET message with the URI-Path “Trust_Evidence” is sent to request the trust evidence. The requested trust evidence is returned in the response message payload with code 2.04 “Content”.

After receiving the trust evidence, the trust repositor calculates a new trust value based on the trust evidence. The trust repositor then uses a POST message with the URI-Path “Trust_Value_Sotrage” to carry the updated trust value back to the trust repositor for storage. The trust repositor acknowledges the receipt of the new trust value with the response code 2.01 “Created”, 4.03 “Forbidden”, or 5.00 “Internal Server Error”, depending on the outcome.

4. Querying trust value

When the delegator needs to interact with a trustee, it requests the trustee's trust value from the trust repositor using a GET request with the URI-Path “/Trust_Value”. The trust repositor retrieves the required trust value from its database and responds with a 2.01 “Content” code and trust values piggybacked in the payload. After receiving the trust value, the delegator makes the trust decision itself. If the delegator does not have the right permission to access the trust value, 4.03 “Forbidden” is sent.

5.9 Resource access control

5.9.1 Permission delegation

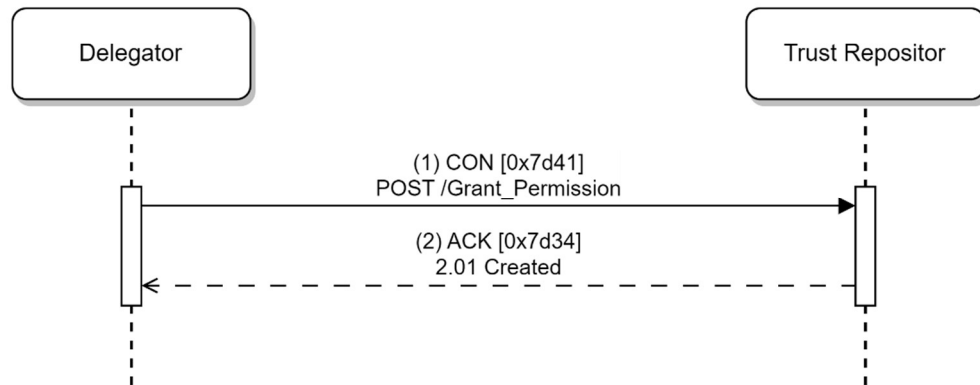


Figure 5.15: Granting trust value and trust evidence access permission

As described in the previous Chapter, three types of permissions are defined for accessing trust values and trust evidence: Read, Insert, and Delete. Furthermore, role-permission relationships are established to specify who has which of the permissions to access the trust value and trust evidence.

The delegator delegates access permission of its trust evidence and trust value to the trust repositor by using a POST message, as shown in Figure 5.15. The delegator sends a POST message with the URI-Path “/Grant_Permission” to the trust repositor. The payload of the message includes IP addresses and their associated permissions for trust value and trust evidence, as shown below.

POST /Grant_Permission

- **Trustor_IP:** IPv6 addresses of the delegator (owner of the data).
- **Permission:** [IPv6 address = [‘P1’, ‘P2’], IPv6 address = [‘P1’, ‘P2’], ...]

Where P1 represents the permission for trust value access, and P2 represents the permission for trust evidence access. The permissions are specified in the form of RID, ‘R’ means read permission, ‘I’ means insert permission and ‘D’ means delete permission. The trust repositor stores the permission information in its database and replies with one of the following response codes.

- **2.01 “Created”** – If the permission delegation is successful and no existing permission for the delegator is found in the database.
- **2.04 “Changed”** – If the permission delegation is successful and the delegator's permissions already exist in the database.
- **5.00 “Internal Server Error”** – If an error occurs during the delegation process.

5.9.2 Permission delegation revocation

To revoke delegated permissions, the delegator sends a DELETE request with the URI-Path “/Revoke_Permission” to remove all permissions previously delegated to the trust repositor, as shown in Figure 5.16.

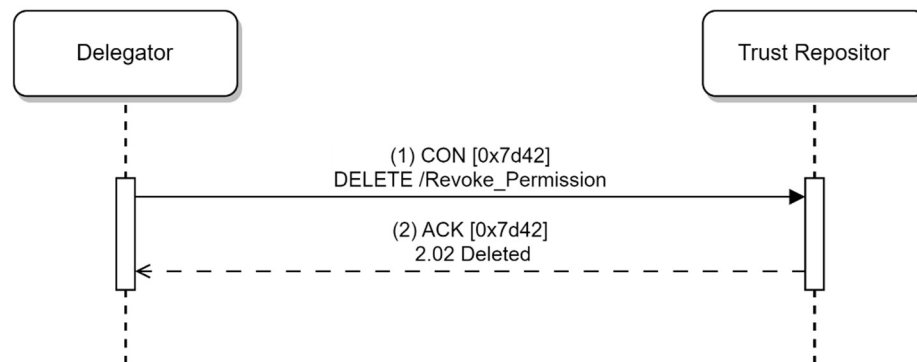


Figure 5.16: Revoking trust value and trust evidence access permission

DELETE /Revoke_Permission:

- **Trustor IP:** IPv6 address of the delegator (owner of the data).

The trust repositor delete the corresponding permission records from its database and responds with a response code 2.02 “Deleted” if the revocation is successful or 5.00 “Internal Server Error” if an error occurs.

5.9.3 Access control verification

After the delegator delegates permission to a repositor, the repositor stores the role-permission information in its database for subsequent access verification. When an entity attempts to access trust data, the following verification process is performed.

1. The trust repositor receives the request message and extracts the IP address of the requesting entity. It then queries the access permission database to identify the permission records associated with that IP address.
2. The request operation in the request message is then compared against the stored permissions (read, insert, or delete) to determine whether the requester is authorised to perform that operation on the requested data.
3. If the requester has the valid permission for the requested operation, access is granted, and the requested data is returned. Otherwise, access is denied, and the trust repositor sends a 4.03 “Forbidden” response message back to the requester. If the requester is the owner of the data (the delegator), the trust repositor will grant full access to the data.

5.10 Mobility management

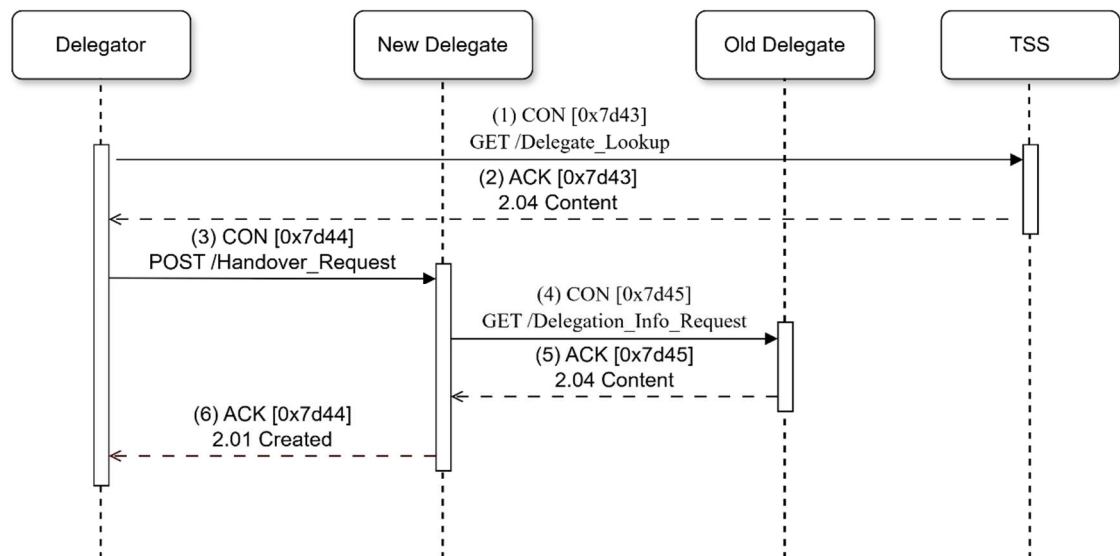


Figure 5.17: Trust delegation handover process

The proposed handover method transfers the delegator's delegation information from the previous delegate to a new delegate when the delegator needs to leave the coverage area of the previous delegate. This allows the delegator to seamlessly continue delegating trust management tasks to the new delegate. The delegation information is transferred directly from the old delegate to the new delegate, without involving the delegator. The proposed handover method can be illustrated in Figure 5.17.

1. Looking up a new delegate

Before disconnecting from the previous delegate, the delegator needs to look up and establish a new connection with a new delegate to ensure continuous communication and prevent packet loss. The delegator sends a GET request to a TSS with the URI-Path “Delegate_Lookup”. The TSS responds with the IP address of a new delegate IP in an acknowledgement message with the 2.04 “Content” response code.

2. Handover

After receiving the new delegate's IP address from the TSS, the delegator initiates the handover process by sending a POST request with the URI-Path “Handover_Request”, to the new delegate. The payload contains the IP address of the previous delegate, enabling the new delegate to contact the old delegate and retrieve the delegator's delegation information.

POST /Handover_Request

- **Delegator IP:** IPv6 address of the delegator.
- **Old delegate IP:** IPv6 address of the previous delegate.

The new delegate then sends a GET request to the old delegate with the URI-Path “Delegation_Info_Request” to request the delegator's delegation information.

GET /Delegation_Info_Request

- **Delegator IP:** IPv6 address of the delegator.

The old delegate responds to the GET request with a 2.04 “Content” response code and the following delegation information in the payload. It also deletes the delegator's delegation information from its database.

- **Trustor_IP:** IPv6 address of the delegator.
- **Trustee_IP:** IPv6 address of a trustee, or a list of IPv6 addresses.
- **Lease_time:** Delegation time, in seconds.
- **Trust_model:** Trust model used for trust calculation.
- **Trust_update:** Trust update scheme.
- **Evidence:** A list of trust evidence names and their corresponding values.
- **Share:** Boolean 0 or 1 to indicate sharing recommendations with others.

When the new delegate receives the delegation information, it stores the data in its database and sends an acknowledgement with a response code below to the delegator to indicate the status of the handover. However, if it is not possible to return an acknowledgement message. The new delegate first sends an empty acknowledgement in response to the POST request, then sends a separate message with the appropriate response code.

- **2.01 “Created”** – If the handover is successful.
- **2.04 “Changed”** – If the handover is successful and the delegator’s delegation information already exists in the database.
- **5.00 “Internal Server Error”** – If anything goes wrong with the handover.

5.11 Conclusion

This Chapter presented the proof-of-concept implementation of the trust management delegation protocol design in Chapter 4. The implementation is written in Python, a cross-platform programming language that can be deployed on various operating systems. This flexibility supports the deployment of the delegation across different devices in a fog computing system. Furthermore, CoAP is leveraged to facilitate data and message transfers between entities in the proposed protocol. The SQLite database engine is employed to store trust management and delegation information. In Chapter 6, we will evaluate the implementation of the trust management delegation protocol and measure the resource consumption.

Chapter 6: Experiments and results

This chapter presents an experimental evaluation of the proposed trust management delegation protocol within different experimental settings. The evaluation focuses on resource consumption in terms of CPU time, RAM, HDD space, and energy consumption across different device roles. It also examines the impact of different trust models on resource consumption.

6.1 Experimental evaluation questions

The experimental evaluation is designed to answer the following key questions:

- EQ1: Does the delegation protocol reduce resource consumption (CPU, RAM, storage, energy) on resource-constrained devices compared to local trust management?

This question directly addresses the problem that the delegation protocol was designed to solve. Demonstrating measurable reductions in resource consumption validates that the protocol can accommodate resource-constrained devices for trust management, particularly when handling computationally intensive trust models.

- EQ2: What is the impact of different trust computation models (weighted sum vs. fuzzy logic) on the protocol's performance and resource overhead?

Different applications may require different trust models with varying computation complexity. This question demonstrates that the trust management delegation protocol can effectively support this flexibility with reasonable overhead.

- EQ3: What is the resource overhead on delegate nodes when performing trust management on behalf of multiple constrained devices?

Fog systems may contain hundreds or thousands of devices. Understanding how resource consumption on delegate nodes scales with the number of delegating devices is essential to ensure the protocol can operate effectively with less overhead.

6.2 Experimental methodology and setup

6.2.1 Experimental testbed

The experimental testbed used in this research consists of Raspberry Pi Zero devices acting as end nodes, Raspberry Pi 4 devices acting as fog nodes and delegates, and virtual machines running Raspberry Pi Desktop on Oracle VirtualBox acting as additional end nodes and fog nodes in larger-scale experiments. The testbed was designed with reference to the device capability classes defined in Section 4.2. The Raspberry Pi Zero devices were selected to represent constrained end nodes in Class 0 and Class 1 because they have limited processing power, memory, and energy capacity, similar to many IoT devices deployed at the end layer, such as the medical devices, home appliances, and on-board units, described in the fog computing scenarios analysed in Chapter 3. In contrast, the Raspberry Pi 4 devices were used to represent fog nodes or delegates in Class 2 and Class 3 because they provide greater computational and storage resources. The difference in capability between the two device platforms reflects the heterogeneous nature of real fog environments. Moreover, it makes the testbed suitable for evaluating whether trust management delegation can reduce the burden on constrained devices and transfer trust management tasks to more capable nodes.

Virtual machines running Raspberry Pi Desktop on VirtualBox were used to simulate additional end nodes and fog nodes when the experiments required a large number of participants without requiring physical hardware. This approach is commonly used in large-scale IoT research where the size of physical devices is limited by hardware availability [158]. The virtual machines run the same operating system, network stack, and application software as the physical devices, meaning that the protocol behaviour, message exchange, and data processing logic are identical to what would occur on physical hardware. However, virtual machines do not accurately replicate the CPU scheduling, memory management, or energy characteristics of physical devices. For this reason, they were used only to represent target trustee nodes in the experiments and were excluded from all resource consumption measurements.

The network configuration was configured to reflect the characteristics of fog computing, in which devices typically communicate over local networks rather than across the wider Internet. In the experiment, all devices communicate over a local Wi-Fi network, reflecting short-distance, low-latency communication. This is expected in practical fog deployments where nodes are located close to one another. Moreover, the distance between two nodes was simulated by implementing network routers in VirtualBox. For example, two virtual routers were deployed in VirtualBox to create an intermediate network between trustors and trustees.

6.2.2 Measurement

CPU time, RAM usage, and HDD space were monitored during the experiments using dedicated Python scripts. These scripts were run separately from the trust management delegation processes to ensure that only the delegation process was measured and excluded from additional measurement overhead. CPU time and RAM usage were measured using the psutil Python library. It retrieves the cumulative CPU time of the process via its process identifier (PID). This ensures that only the CPU time consumed by the delegation protocol operations was captured. Similarly, RAM usage is measured directly from the resident set size (RSS) of the delegation process. HDD space usage was measured by the size of the database file written to disk.

Energy consumption was measured using an INA219 sensor module [159] attached to the Raspberry Pi Zero. The sensor was connected between the power source and the device to continuously measure DC voltage (V), current (I), and Power (W). The total energy consumption, in Watt-hours (Wh) unit, was then calculated by multiplying the measured power by the experiment duration in hours. This method is considered reliable because it is based on direct hardware measurement of the electrical power consumed by the device, rather than on software-based estimates of CPU utilisation or execution time. According to the INA219 datasheet, the device provides high accuracy of up to 0.5% maximum over temperature [159].

To minimise the impact of background activity for energy measurements, all non-essential background processes were terminated before each experiment. However, some residual energy consumption from essential system processes may still be

included in the measurements. To ensure consistency, the experiments were conducted under the same system configuration and operating conditions. As a result, any residual energy consumption from essential system processes was assumed to remain constant across experiments. Therefore, the analysis focuses on differences in measured energy consumption between experimental setups rather than on exact energy values. Moreover, each experiment was repeated to reduce the effect of unusual results and improve the reliability of the measurements.

6.2.3 Experimental procedure and data analysis approach

- Each experiment is repeated ten times under the same conditions. The standard deviation and 95% confidence interval (CI) are calculated from these repetitions to show experimental uncertainty.
- The results from each experiment run are stored in data files. The files are then analysed using Python scripts to extract and process the recorded measurements.
- Matplotlib, a Python plotting library, is used to generate all graphs presented in this chapter. The graphs include error bars representing the 95% confidence intervals.

6.3 Trust metrics and models

The following are trust metrics and trust models used in the experiments to evaluate the trustworthiness of trustees.

6.3.1 Trust metrics

The trust metrics used in the experiments are based on the trust properties discussed in Chapter 2. Although similar properties may be defined differently in other studies, this section explains in detail how each metric is defined and measured in this research.

1. Response time

Response time is the time that a trustee takes to respond to a request from a trustor. The trustor expects the trustee to respond to the request with a low response time. The trustor evaluates the response time towards the trustee by sending a task for processing, and the trustor measures the time interval between when the task is sent

and when the response is received from the trustee. The task might be, for example, a request for energy consumption data from a home hub in a smart home scenario.

Response time is scaled to values between 0 and 1 for use as an input in trust evaluation models. We follow the acceptable response time for web services in [141] to scale the response time to 3 ranges. We define a good response time as below 100 ms, a medium response time as between 100 ms and 200 ms, and a poor response time as over 200 ms. Therefore, good, medium, and poor response times are denoted as 1, 0.5, and 0, respectively, as shown below.

$$RT = \begin{cases} \text{response time} < 100 \text{ ms}, & 1 \\ \text{response time between } 100 - 200, & 0.5 \\ \text{response time} > 200, & 0 \end{cases}$$

2. Availability

Availability means that a trustee is accessible and operational when requested [21]. It is called unavailable or inaccessible if it is shut down or too busy to process a request. A trustor measures the availability of the trustee by sending a ping message. The trustor might send the ping every 1 hour or 2 hours, depending on the application scenarios. However, in the experiments, the trustor sends an ICMP ping message every 10 seconds. The trustor calculates availability as the ratio of the number of echo reply messages the trustor receives from the trustee to the total number of echo request messages.

$$AV = \frac{\text{Number of ICMP echo reply}}{\text{Number of ICMP echo request}}$$

3. Task completion rate

Task completion rate is the ratio of the number of successful tasks to the total number of tasks that a trustor has sent to a trustee. It shows that the trustee has the capability to perform a task requested by the trustor [6]. The task could be offloading data to store in a fog node or sending a command to control an end node. The trustor keeps the record of successful tasks and calculates the task completion rate by the following formula.

$$TR = \frac{\text{Number of successful tasks}}{\text{Total number of tasks}}$$

4. Distance

Distance indicates how far a trustee is from a trustor. We define distance as the number of network hops between the trustor and the trustee. Data sent by a sender must travel through multiple intermediate devices, like routers, to reach a recipient. Each hop the data passes through introduces a delay, resulting in an increase in latency. Moreover, the closer the distance between the sender and the recipient, the more reliable and faster the communication is. The trustor can measure the distance between itself and the trustee by several approaches. However, we use a network monitoring tool called traceroute to measure the number of hops between them. In a fog computing system, the maximum number of hops is not greater than two hops. The distance is mapped into the scores as shown in Table 6.1 to be an input value in trust models.

Table 6.1: Distance scores

Hop count	Score
0	1
1	0.5
2	0

5. Packet loss rate

The packet loss rate is defined as the ratio of dropped packets to the total number of packets sent. A trustor requires high reliability when transferring data to a trustee. Packet loss can cause data accuracy issues and delays as data must be retransmitted. Some IoT applications, such as E-health, require a low packet loss rate to transmit patients' vital signs and other relevant data. High packet loss rates cause delays and affect the analysis of time-sensitive data and the response to emergency situations in critical health monitoring systems. The accuracy of data affects the diagnostic process, leading to misdiagnosis and serious consequences for patients.

To simulate packet loss rate in our experiment, we use Linux utilities, NetEM (Linux Network Emulator) and TC (Traffic Control). The trustor sends a ping message 10 times in total and calculates the packet loss rate using the following formula.

$$PLR = \frac{\text{Packets sent} - \text{Packets received}}{\text{Packets sent}} \times 100\%$$

According to [160], an acceptable packet loss rate for medical data transfer in E-health applications is less than 3%. Therefore, we scale the packet loss rate to 3 ranges varying from good, medium, and poor [1] , which are denoted as 1, 0.5, and 0, respectively, as shown in Table 6.2.

Table 6.2: Packet loss rate scores

Packet loss percentage	score
0%	1
1–2%	0.5
>3%	0

6. Reputability

Reputability indicates that a trustee is not malicious and is well-behaved. A malicious node can perform malicious activities, such as sending false data. Moreover, it could have unintentional misbehaviour, such as software or hardware failure. In our experiment, a trustor detects a misbehaving node by checking data against the normal data range. For example, in an E-health scenario, a fog node checks a patient’s body temperature sent by an end node within the normal range of 35.0 to 42.3 °C. The fog node counts the number of times it experiences this misbehaviour and calculates reputability using the following formula.

$$\text{Reputable score} = \frac{\text{Number of interaction} - \text{Number of misbehaviour}}{\text{Number of interactions}}$$

7. Network latency

Latency refers to the delays that occur within a network. Network latency is caused by several factors, such as link failures and network congestion. A trustor sends a ping message to a trustee and measures the time it takes to receive a response, or round-trip time (RTT), in milliseconds. We consider RTT to be good if it is less than 50 ms, medium if between 50 and 100 ms, and poor if over 100 ms. Therefore, the network latency ranges are denoted as 1, 0.5, and 0, respectively, as shown in Table 6.3.

Table 6.3: Network latency values

Response time	value
0 – 49ms	1
50 – 100ms	0.5
>101ms	0

8. Cooperativeness

Cooperativeness indicates whether the trustee is cooperative and willing to support the trustor. In the smart traffic management scenario, the vehicle expects high cooperativeness from other vehicles to cooperate with others to provide information within the smart traffic management system.

$$\text{cooperativeness} = \frac{\text{Number of common friends}}{\text{Total number of friends}}$$

The cooperativeness of a trustee as observed by a trustor is computed by the ratio of the number of common friends over the total number of trustor's and trustee's friends, as the formula above. Our hypothesis is that friends are likely to be cooperative towards each other. The trustor maintains a list of its friends that it has successfully interacted with in the past. When the trustor and the trustee encounter each other, they exchange their friend lists. The trustor computes the number of their common friends and then calculates the degree of cooperativeness.

6.3.2 Trust models

We implement two trust evaluation models, namely weighted-sum and fuzzy-based models, as follows.

1. A weighted sum-based trust model

Weighted sum is used to aggregate individual trust properties where each piece of evidence is assigned a static weight, then an overall trust value is calculated using the following formula:

$$\text{Trust value} = \sum_{i=1}^n W_i X_i$$

Where $W_i = \{w_1, w_2, \dots\}$ are static weighting factors for X_i and $\sum_{i=1}^n W_i = 1$.

2. A fuzzy logic-based trust model

A scikit-fuzz tool [161] is used to implement a fuzzy logic-based trust model. Scikit-fuzzy is a fuzzy logic package for Python, including fuzzy logic algorithms, methods, and tools for fuzzy logic applications. The fuzzy logic approach in [26] is adopted to implement a fuzzy logic-based trust model in the experiments. Linguistic qualifiers, including low, medium, and high, are defined to convert trust properties into fuzzy sets. The linguistic variables for output are also high, medium, and low. Fuzzy sets are defined by triangle and trapezoid-shaped curves. The fuzzy logic model consists of IF-THEN rules that express the relationship between the inputs and output variables in the defuzzification process. The output of the calculation is a trust value ranging from 0 to 1.

6.4 Experiment 1: Local trust management and delegation

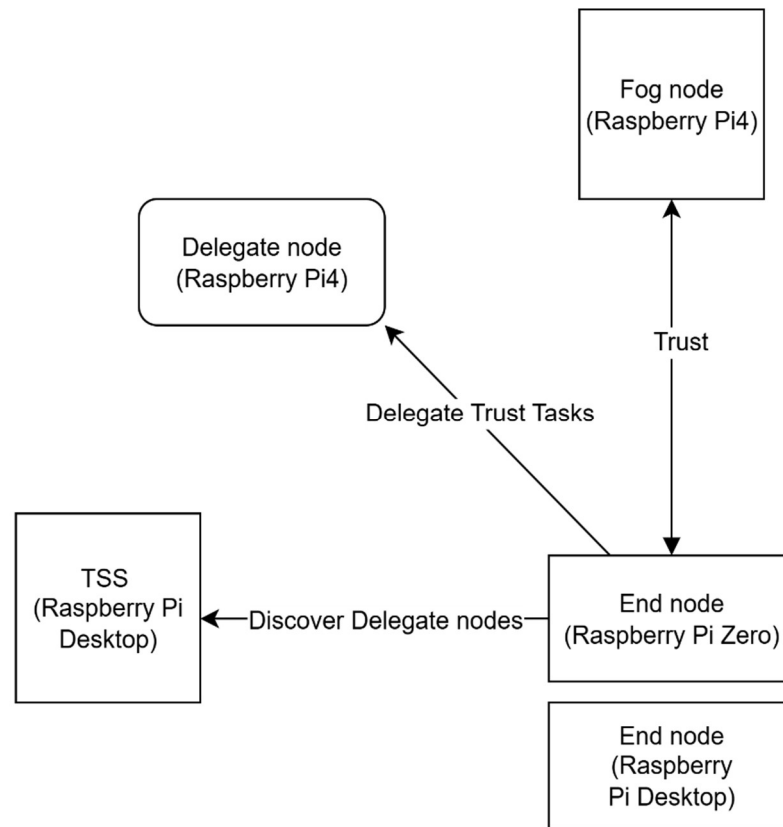


Figure 6.1: Experiment 1 setup

The first experiment evaluates the effect of delegation on resource consumption by examining local trust management with delegating trust management. The experiment is conducted in three parts. First, the end node performs trust management locally and delegates trust management to the delegate. Second, the experiment analyses the resource consumption of the delegate node when managing trust on behalf of 1 to 15 end nodes. Third, the experiment evaluates the fog node to perform trust management towards 1 to 15 end nodes.

As shown in Figure 6.1, the end node evaluates the trustworthiness of the fog node using response time, availability, and task completion rate and applies a weighted sum-based trust model as defined in Section 6.3. The delegation process begins with the end node discovering the TSS using the pull-based service discovery. It then queries the TSS to discover suitable delegates that meet delegation requirements. After delegate selection, the end node sends a trust management delegation request to the delegate to initiate the delegation. The end node then periodically collects all trust evidence towards the fog node itself and forwards it to the delegate for storage. The delegate updates trust values towards the fog node using a weighted sum trust model via an event-driven approach upon receiving new evidence from the end node. The end node sends a trust result request to the delegate that contains a threshold value for decision-making when a trust result towards the fog node is needed.

To simulate trust evidence collection, the end node sends sensor data (e.g., room temperature) to the fog node and measures the time interval between sending the data and receiving a response to collect the response time. Availability is assessed by sending ICMP ping messages to the fog node and calculating the ratio of received echo replies to the total number of pings sent. Task completion rate is measured by tracking the number of successful acknowledgements for data storage from the fog node.

Similarly, the fog node performs trust management towards multiple end nodes locally, including evidence collection, evidence storage, trust update, and decision-making, using the same trust metrics and trust model. Response time here is measured by sending control commands to end nodes rather than sensor data. Availability and task completion rate are calculated using ping replies and successful control command executions, respectively.

6.4.1 Resource consumption on the end node: Local trust management versus delegating trust management

The first part measures resource consumption on the end node performing trust management locally versus delegating trust to the delegate. In the first phase, the end node handles trust management locally. It collects evidence, stores it in its database, performs a trust update and makes a decision towards the fog node every 10 seconds and 3000 times in total or until the number of evidence records reaches 3000 records. In the second phase, the end node delegates all trust management tasks, except for evidence collection, to the delegate. The end node collects trust evidence against the fog node and forwards it to the delegate. Then, it periodically requests trust results from the delegate every 10 seconds and 3000 times.

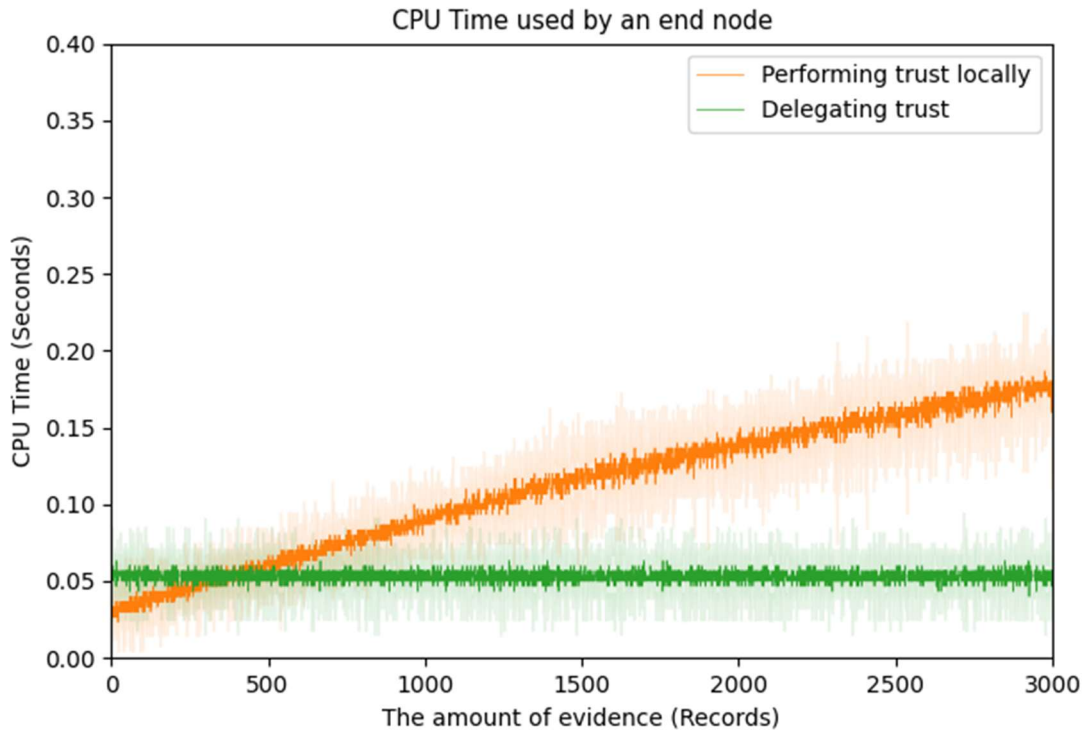


Figure 6.2: CPU time used by an end node on doing trust locally versus delegating trust

Table 6.4: CPU time used by an end node on doing trust locally versus delegating trust

Evidence (records)	1	301	601	901	1201	1501	1801	2101	2401	2701
CPU Time for local trust (Seconds)	0.03	0.0467	0.0667	0.08	0.103	0.12	0.123	0.143	0.15	0.157
SD	0	0.0058	0.0058	0	0.0058	0	0.0058	0.0058	0.0173	0.0115
CI (95%)	(0.03, 0.03)	(0.0324, 0.061)	(0.0524, 0.081)	(0.08, 0.08)	(0.089, 0.118)	(0.12, 0.12)	(0.109, 0.138)	(0.129, 0.158)	(0.107, 0.193)	(0.128, 0.185)
CPU Time for delegation (Seconds)	0.0533	0.0533	0.0533	0.0467	0.05	0.05	0.05	0.0533	0.05	0.0467
SD	0.0115	0.0058	0.0058	0.0058	0.01	0.01	0	0.0058	0	0.0058
CI (95%)	(0.0246, 0.082)	(0.039, 0.0676)	(0.039, 0.0676)	(0.0324, 0.061)	(0.0252, 0.0748)	(0.0252, 0.0748)	(0.05, 0.05)	(0.039, 0.0676)	(0.05, 0.05)	(0.0324, 0.061)

Figure 6.2 and Table 6.4 compare the CPU time required by the end node when trust management is performed locally and when it is delegated to a delegate. The horizontal axis represents the number of trust evidence records, while the vertical axis shows CPU time in seconds. At low numbers of evidence records, CPU time for local trust management is lower than delegating trust until the number of evidence records reaches approximately 400 records. Beyond this point, local trust management requires more CPU time, which increases steadily with the number of evidence records increases. In contrast, the CPU time of delegating trust remains stable throughout the experiment. This demonstrates that the overhead introduced by delegation is outweighed when larger volumes of evidence must be processed. Consequently, the protocol is particularly effective for reducing the computational burden in scenarios where large amounts of trust evidence are handled using the trust algorithm applied in this study.

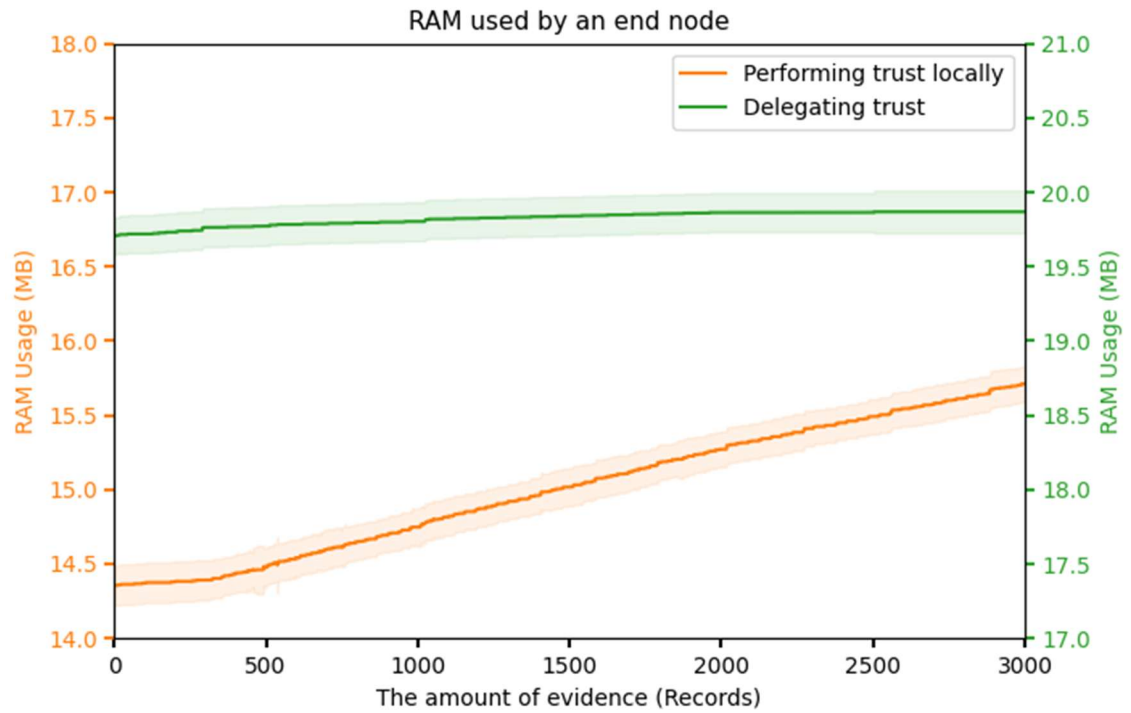


Figure 6.3: RAM used by an end node on doing trust locally versus delegating trust

Table 6.5: RAM used by an end node on doing trust locally versus delegating trust

Evidence (records)	1	301	601	901	1201	1501	1801	2101	2401	2701
RAM usage for local trust (MB)	14.4	14.4	14.5	14.7	14.9	15	15.2	15.3	15.4	15.6
SD	0.054	0.054	0.05	0.048	0.048	0.051	0.048	0.048	0.045	0.049
CI (95%)	(14.2, 14.5)	(14.3, 14.5)	(14.4, 14.7)	(14.6, 14.8)	(14.8, 15)	(14.9, 15.1)	(15.1, 15.3)	(15.2, 15.4)	(15.3, 15.6)	(15.5, 15.7)
RAM usage for delegation (MB)	19.7	19.8	19.8	19.8	19.8	19.8	19.9	19.9	19.9	19.9
SD	0.05	0.052	0.052	0.052	0.053	0.053	0.053	0.053	0.053	0.058
CI (95%)	(19.6, 19.8)	(19.6, 19.9)	(19.7, 19.9)	(19.7, 19.9)	(19.7, 20)	(19.7, 20)	(19.7, 20)	(19.7, 20)	(19.7, 20)	(19.7, 20)

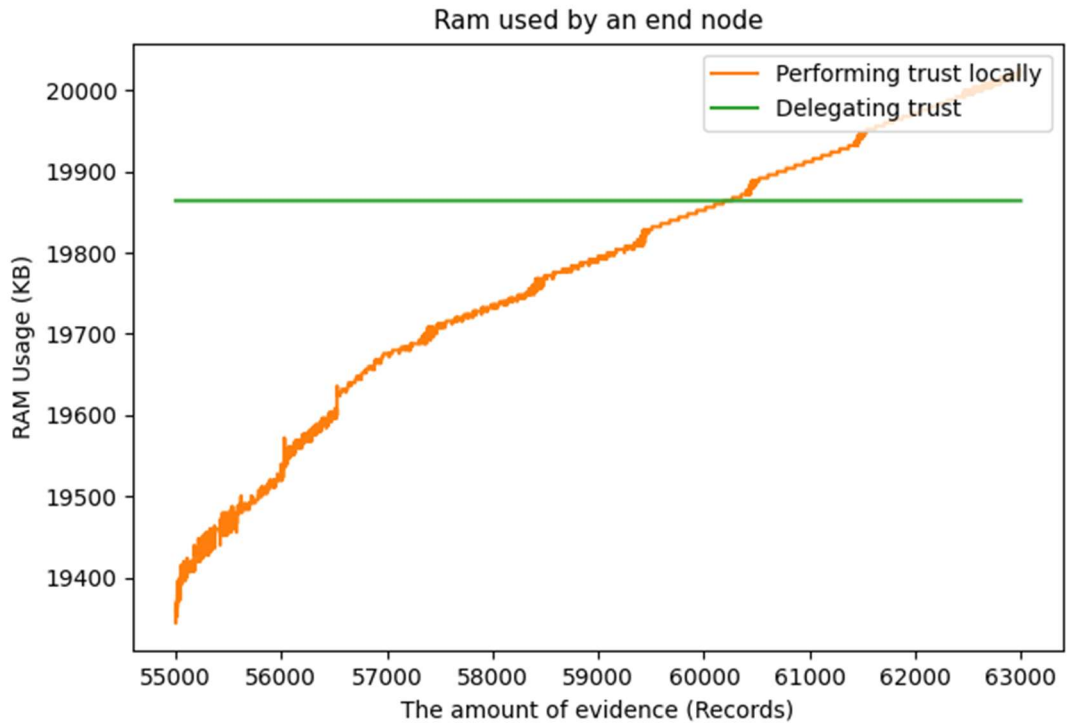


Figure 6.4: The point at which RAM consumption performing trust locally becomes higher than the RAM consumption of delegating trust

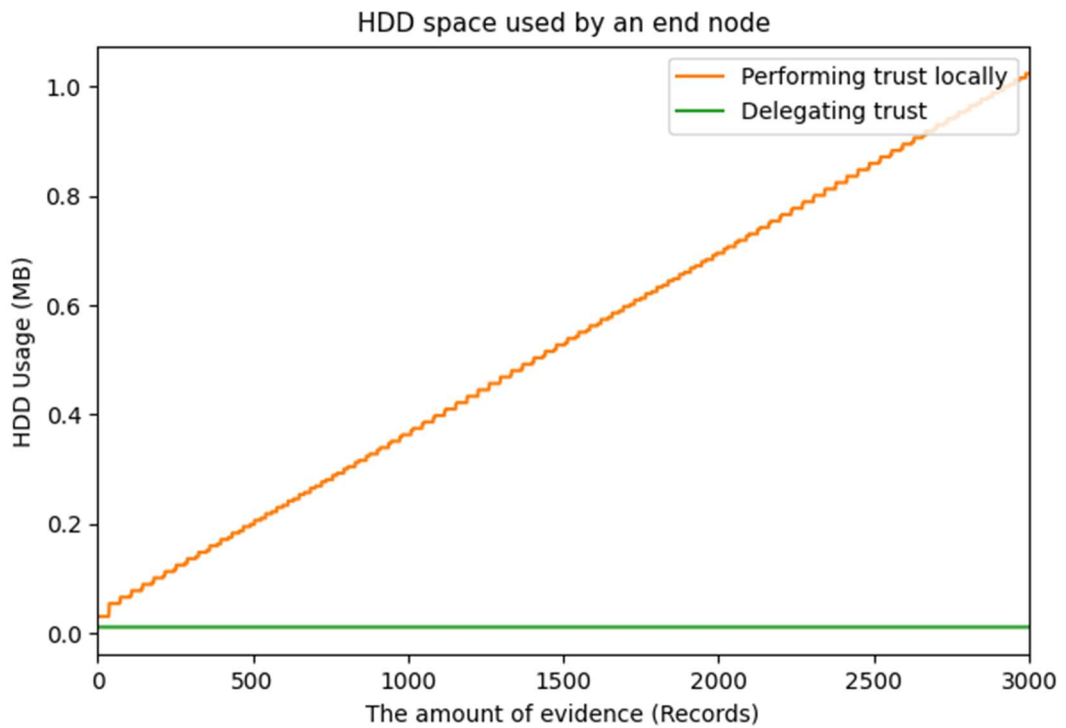


Figure 6.5: HDD used by an end node on doing trust locally versus delegating trust

Table 6.6: HDD used by an end node on doing trust locally versus delegating trust

Evidence (records)	1	301	601	901	1201	1501	1801	2101	2401	2701
HDD usage for local trust (MB)	0.031	0.14	0.23	0.34	0.43	0.53	0.63	0.73	0.82	0.93
SD	0	0	0	0	0	0	0	0	0	0
CI (95%)	(0.031, 0.031)	(0.14, 0.14)	(0.23, 0.23)	(0.34, 0.34)	(0.43, 0.43)	(0.53, 0.53)	(0.63, 0.63)	(0.73, 0.73)	(0.82, 0.82)	(0.93, 0.93)
HDD usage for delegation (MB)	0.012	0.012	0.012	0.012	0.012	0.012	0.012	0.012	0.012	0.012
SD	0	0	0	0	0	0	0	0	0	0
CI (95%)	(0.012, 0.012)	(0.012, 0.012)	(0.012, 0.012)	(0.012, 0.012)	(0.012, 0.012)	(0.012, 0.012)	(0.012, 0.012)	(0.012, 0.012)	(0.012, 0.012)	(0.012, 0.012)

Figure 6.3 and Table 6.5 present the RAM usage in megabytes (MB) of the end node for local trust management versus delegating trust. The results show that delegation requires more RAM than local trust management due to the overhead of the CoAP library, which consumes around 5000 KB on the Raspberry Pi Zero. However, the RAM usage for delegation increases very slowly as the number of evidence records grows, whereas local trust management increases rapidly. Consequently, local trust management surpasses delegation at around 60,300 evidence records, as shown in Figure 6.4. These findings suggest that delegation has a high memory cost, but that cost becomes less significant when handling large volumes of evidence. Compared with CPU time, delegation is beneficial for RAM usage when handling a very large amount of trust evidence, because its initial memory overhead is higher.

Figure 6.5 and Table 6.6 illustrate the HDD space consumption in megabytes for the end node when performing trust management locally versus delegating trust management. When trust is managed locally, the end node must store both trust values and trust evidence. As a result, it consumes HDD space in accordance with an increase in the amount of evidence. In contrast, the end node collects and stores trust evidence in its storage during delegation, but the evidence is forwarded to the delegate and then deleted immediately. As a result, the HDD space usage of the end node remains consistently low and stable throughout the delegation experiment.

Figure 6.6 compares the total energy consumption in watt-hour (Wh) consumed by the end node. The total energy consumption for delegating trust management is 2.25 Wh, which is slightly lower than doing trust management locally, at around 0.042 Wh. Obviously, the energy consumption of delegating trust is not much lower than doing trust locally. This is because the end node still performs trust evidence collection locally towards the trustees, and evidence collection is one of the most power-intensive operations in trust management.

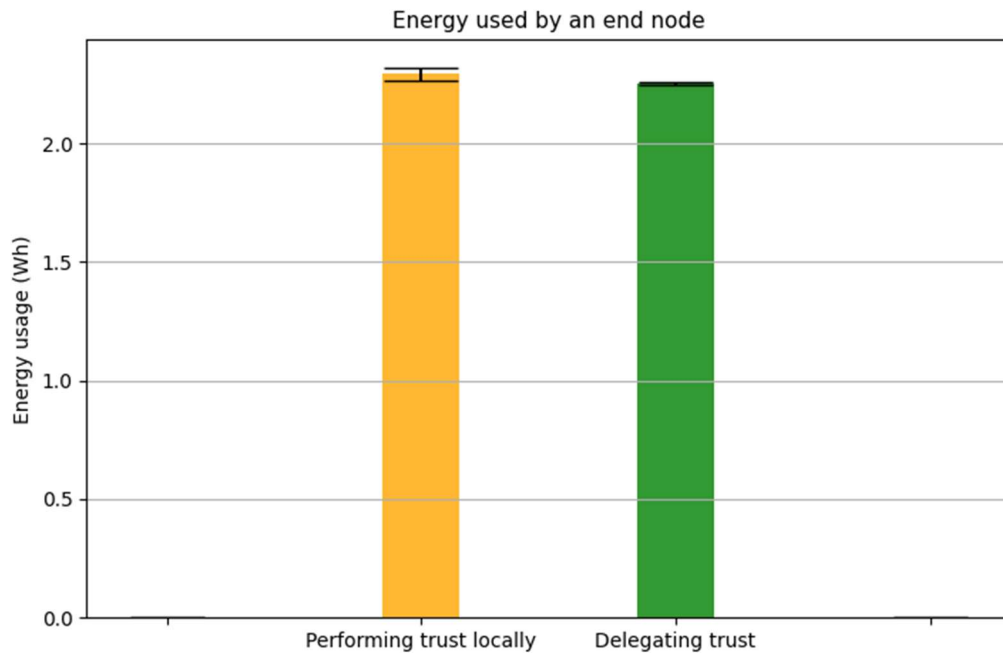


Figure 6.6: Energy consumption by an end node on doing trust locally versus delegating trust

6.4.2 Resource consumption on a delegate performing trust management using a weighted sum on behalf of multiple end nodes.

This experimental result shows the resource consumption of the delegate when performing trust management towards the fog node on behalf of multiple end nodes (delegators). Unlike the previous part, which focused on the benefits of delegation for an end node, this second part examines the costs on the delegate side to effectively support multiple delegators in a fog environment. The number of end nodes in this study ranges from 1 to 15. Multiple end nodes are emulated using Raspberry Pi Desktop instances running on VirtualBox. Each end node delegates all trust management tasks to the delegate, except for evidence collection. Those processes are executed every 10 seconds and 1000 runs in total.

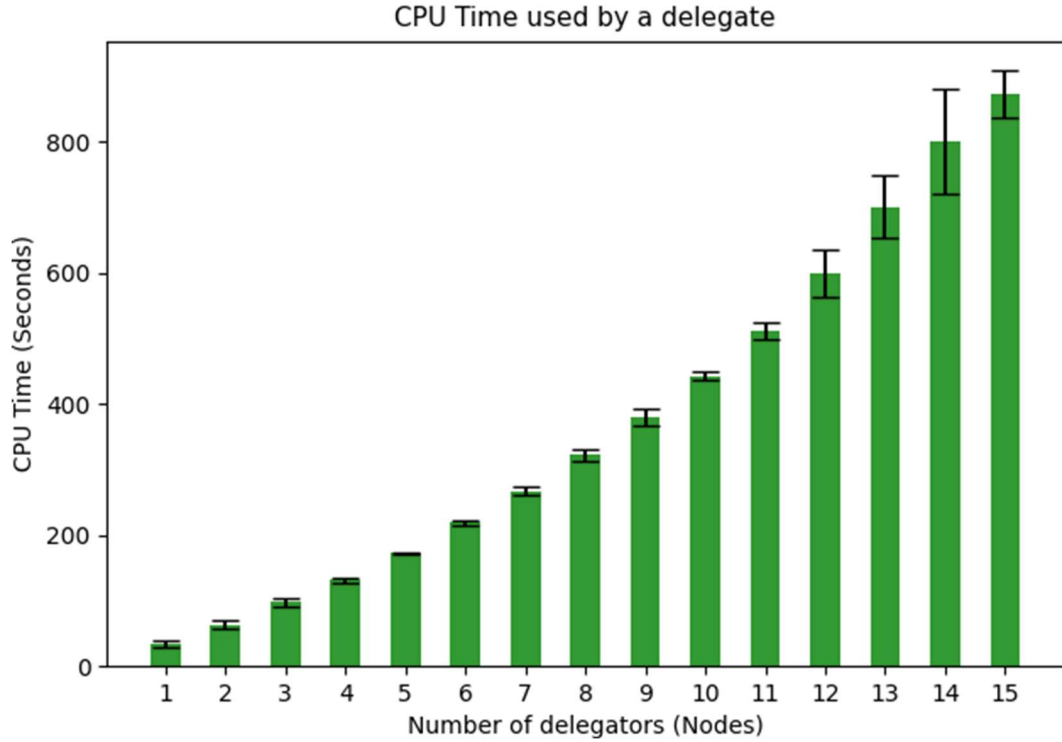


Figure 6.7: CPU time used by a delegate when performing trust using a weighted sum on behalf of 1 to 15 delegators

The results in Figure 6.7 and Table 6.7 show the total CPU time consumed by the delegate to perform trust management for 1-15 delegators (end nodes). The CPU time increases sharply as the number of end nodes grows. For a single end node, the delegate spends 33.6 seconds of CPU time. This nearly doubles to 63.5 seconds for 2 end nodes. As the delegation scales up, CPU time increases significantly, reaching 871 seconds when performing trust management on behalf of 15 end nodes.

Figure 6.8 and Table 6.8 along with Figure 6.9 and Table 6.9 show RAM and HDD space used by the delegate, respectively. Peak RAM usage was measured when the number of end nodes varied from 1 to 15 nodes. Overall, RAM consumption steadily rises throughout the experiment. The delegate consumes 23.2 MB of RAM when managing trust for a single end node, rising by nearly 1 MB to 24.1 MB for 2 end nodes. RAM usage then continues to grow, reaching 43.9 MB for 15 end nodes. In contrast, HDD consumption rises dramatically, starting at 0.38 MB for 1 end node and rising sharply to 5.02 MB for 15 end nodes.

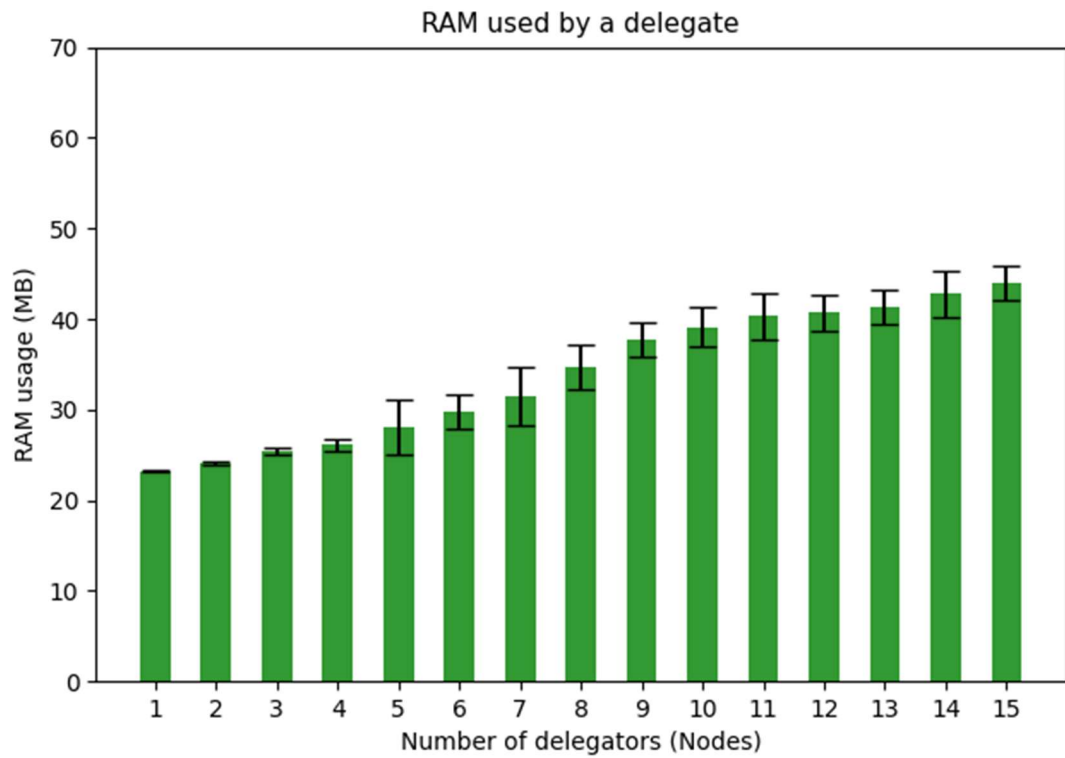


Figure 6.8: RAM used by a delegate when performing trust using a weighted sum on behalf of 1 to 15 delegators

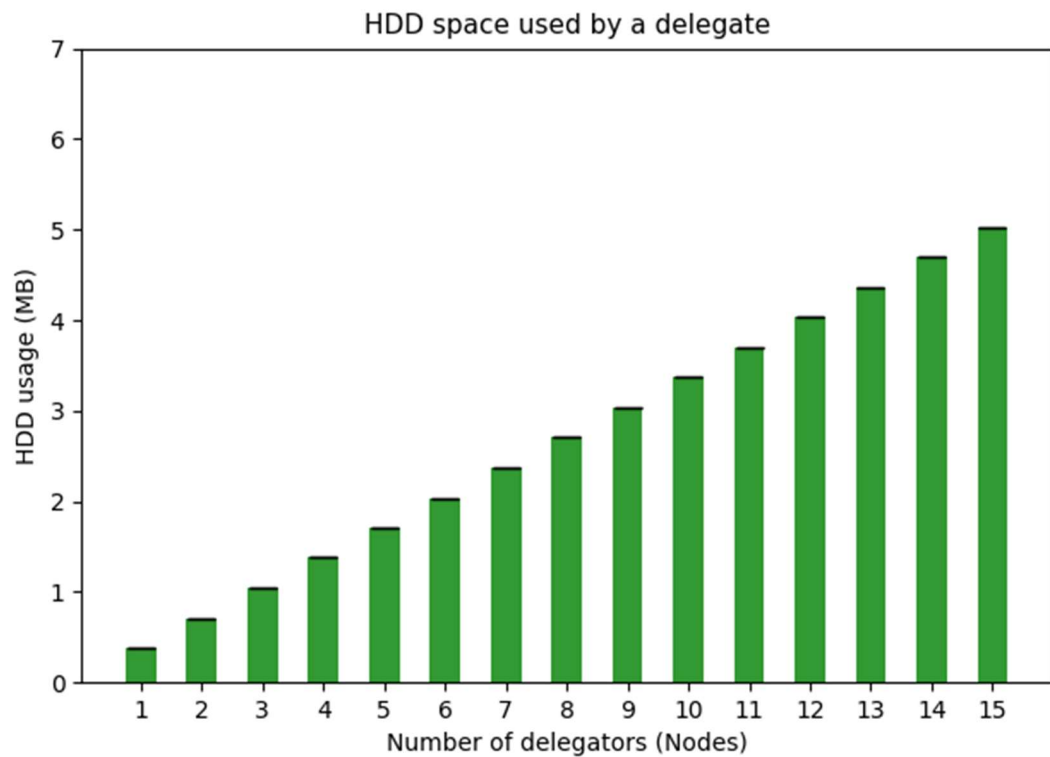


Figure 6.9: HDD used by a delegate when performing trust using a weighted sum on behalf of 1 to 15 delegators

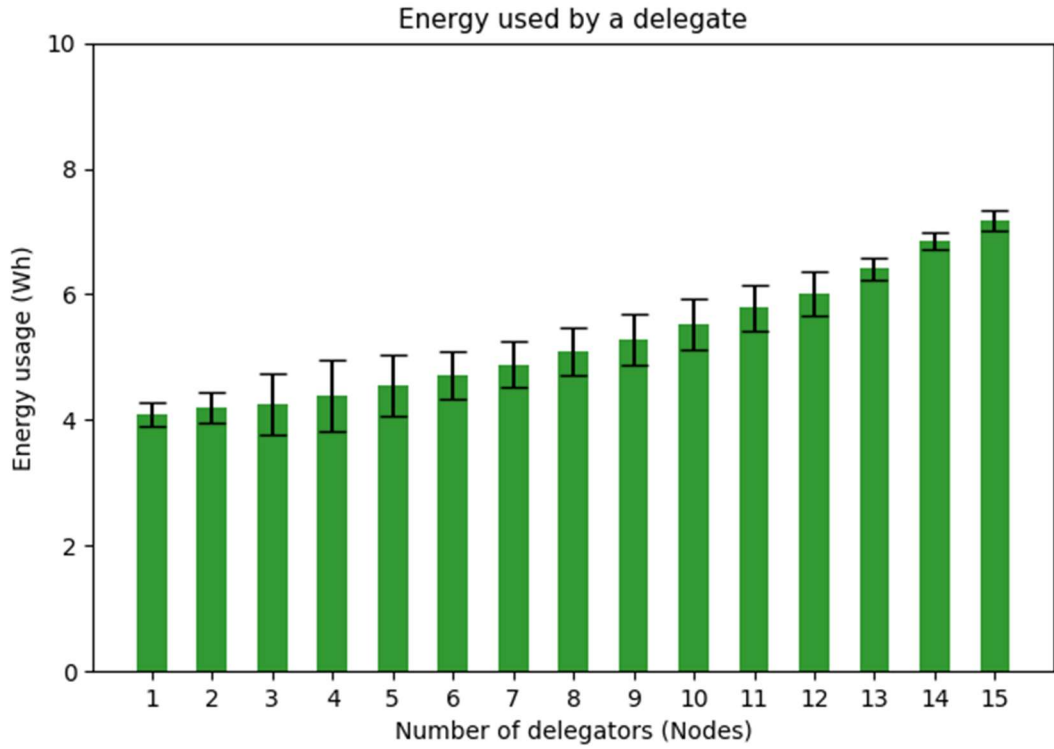


Figure 6.10: Energy used by a delegate when performing trust using a weighted sum on behalf of 1 to 15 delegators

Figure 6.10 and Table 6.10 show the total energy consumption, measured in watt-hour (Wh), of the delegate as the number of end nodes varies from 1 to 15. Energy usage begins at 4.1 Wh for 1 end node, increases moderately to 4.21 Wh for 2 end nodes, and continues climbing to reach 7.17 for 15 end nodes.

These findings demonstrate that the proposed protocol does not remove the costs of trust management. It transfers the costs from resource-constrained end nodes to a delegate. This is significant in a real fog environment because a single delegate may need to support multiple end nodes simultaneously. As a result, the effectiveness of the delegation approach depends on the availability of the required resources on resource-rich nodes, such as the fog nodes. The approach trades resources on resource-constrained nodes for resources on resource-rich nodes, allowing for higher flexibility in trust management choices.

Table 6.7: CPU time used by a delegate when performing trust using a weighted sum on behalf of 1 to 15 delegators

No. delegators (Nodes)	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
CPU Time (Seconds)	33.6	63.5	97.5	131	172	219	267	322	379	442	512	599	701	800	871
SD	3.42	3.16	4.13	1.89	1.09	2.2	3.49	5.38	6.91	3.7	7.47	20.9	27.3	46.2	20.7
CI (95%)	(27.6, 39.6)	(58, 69.1)	(90.2, 105)	(128, 135)	(170, 174)	(215, 223)	(261, 273)	(313, 332)	(367, 391)	(436, 449)	(499, 525)	(562, 636)	(653, 749)	(719, 881)	(835, 908)

Table 6.8: RAM used by a delegate when performing trust using a weighted sum on behalf of 1 to 15 delegators

No. delegators (Nodes)	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
RAM usage (MB)	23.2	24.1	25.4	26.2	28.1	29.8	31.4	34.6	37.8	39.1	40.3	40.7	41.4	42.8	43.9
SD	0.0361	0.102	0.184	0.267	1.2	0.759	1.29	0.986	0.775	0.855	1.01	0.787	0.767	1.03	0.77
CI (95%)	(23.1, 23.3)	(23.8, 24.3)	(25, 25.9)	(25.5, 26.8)	(25.1, 31)	(27.9, 31.7)	(28.2, 34.6)	(32.2, 37.1)	(35.8, 39.7)	(37, 41.3)	(37.8, 42.8)	(38.7, 42.6)	(39.5, 43.3)	(40.3, 45.4)	(42, 45.8)

Table 6.9: HDD used by a delegate when performing trust using a weighted sum on behalf of 1 to 15 delegators

No. delegators (Nodes)	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
HDD usage (MB)	0.379	0.711	1.05	1.38	1.7	2.04	2.37	2.71	3.03	3.37	3.7	4.03	4.37	4.69	5.02

Table 6.10: Energy used by a delegate when performing trust using a weighted sum on behalf of 1 to 15 delegators

No. delegators (Nodes)	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Energy usage (Wh)	4.1	4.21	4.26	4.38	4.56	4.71	4.89	5.09	5.28	5.53	5.79	6.01	6.41	6.86	7.17
SD	0.0763	0.0972	0.198	0.229	0.194	0.156	0.147	0.152	0.162	0.166	0.145	0.139	0.0696	0.0557	0.062
CI (95%)	(3.91, 4.29)	(3.97, 4.45)	(3.76, 4.75)	(3.81, 4.95)	(4.08, 5.04)	(4.33, 5.1)	(4.52, 5.25)	(4.71, 5.47)	(4.88, 5.68)	(5.11, 5.94)	(5.42, 6.15)	(5.66, 6.35)	(6.24, 6.58)	(6.72, 6.99)	(7.02, 7.32)

6.4.3 Resource consumption on a fog node performing trust management towards multiple end nodes.

The third part investigates the resource consumption of the fog node when performing trust management towards varying numbers of end nodes, ranging from 1 to 15. This part differs from the previous one. The focus is not on a delegate performing trust management on behalf of external delegators, but rather on a fog node performing trust management towards end devices or trustees. In this part, the end nodes are emulated using Raspberry Pi Desktop instances running on VirtualBox. The fog node handles all trust management roles on its own, including collecting evidence, storing evidence, calculating trust value, and making decisions. These processes are executed separately and simultaneously for each end node every 10 seconds and a total of 1000 iterations.

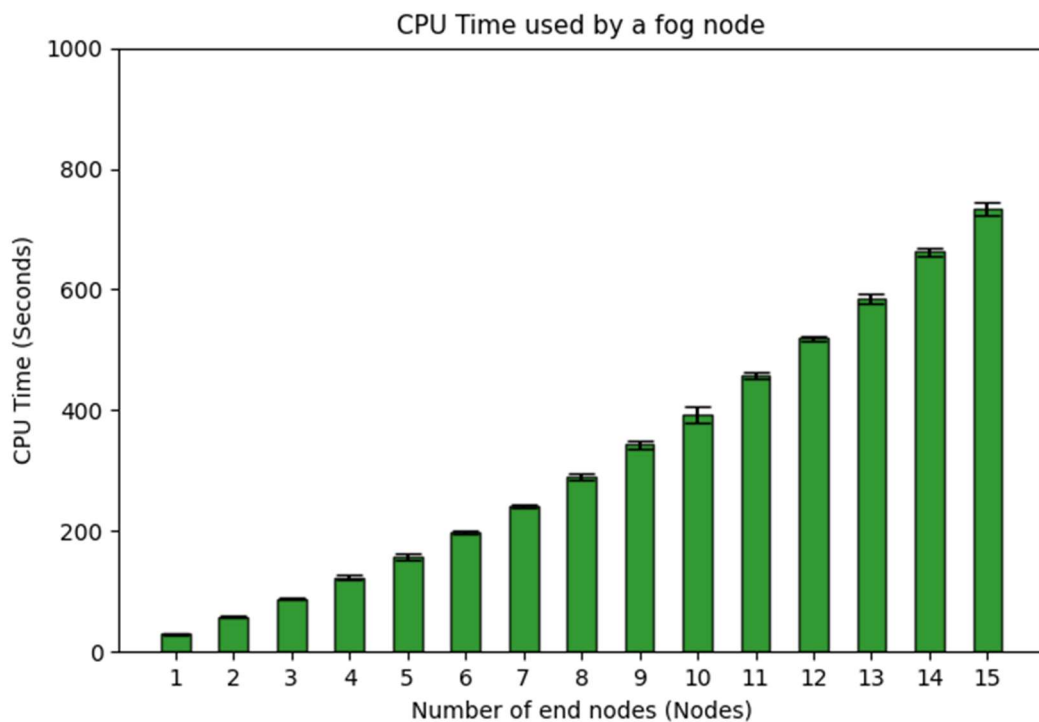


Figure 6.11: CPU time used by a fog node when performing trust management for 1 to 15 trustee end nodes

Figure 6.11 and Table 6.11 show the total CPU time required by the fog node to manage trust as the number of end nodes increases from 1 to 15 end nodes. For a single end node, the CPU time is 29 seconds. When the number of end nodes increases to two, the CPU time nearly doubles to 57.7 seconds. The CPU time grows significantly after 10 end nodes, reaching 734 seconds at 15 end nodes, approximately 24 times higher than at 1 end node.

Figure 6.12 and Table 6.12 present the RAM usage of the fog node. RAM usage rises rapidly from 14.8 MB with 1 end node to 44.9 MB with 15 end nodes. Similarly, HDD space usage, in Figure 6.13 and Table 6.13, grows from 0.363 MB to 4.97 MB as the number of end nodes grows from 1 to 15. The sharp increase in CPU time, RAM, and HDD usage suggests non-linear overhead caused by repeated trust calculations. Although the fog node is considerably more capable than resource-constrained end devices, it may have fewer CPU, RAM, and HDD resources available for its primary tasks when estimating the trust of a large number of end nodes.

In contrast, the total energy consumption of the fog node, as shown in Figure 6.14 and Table 6.14, increases only moderately compared with the dramatic rise in CPU time. With 1 end node, energy consumption is 2.26 Wh. The consumption increases very slightly to 2.36 Wh at 2 end nodes, then rises to 3.75 Wh at 15 end nodes, an increase of 1.39 Wh. The relatively small increase in energy consumption suggests that the main concern is computational overhead rather than energy demand. Energy consumption does not directly correlate with CPU time. The reason is that the fog node, represented by a Raspberry Pi 4, already consumes a baseline level of energy for normal operation. As a result, the additional CPU time required for trust management does not greatly increase the total energy consumption.

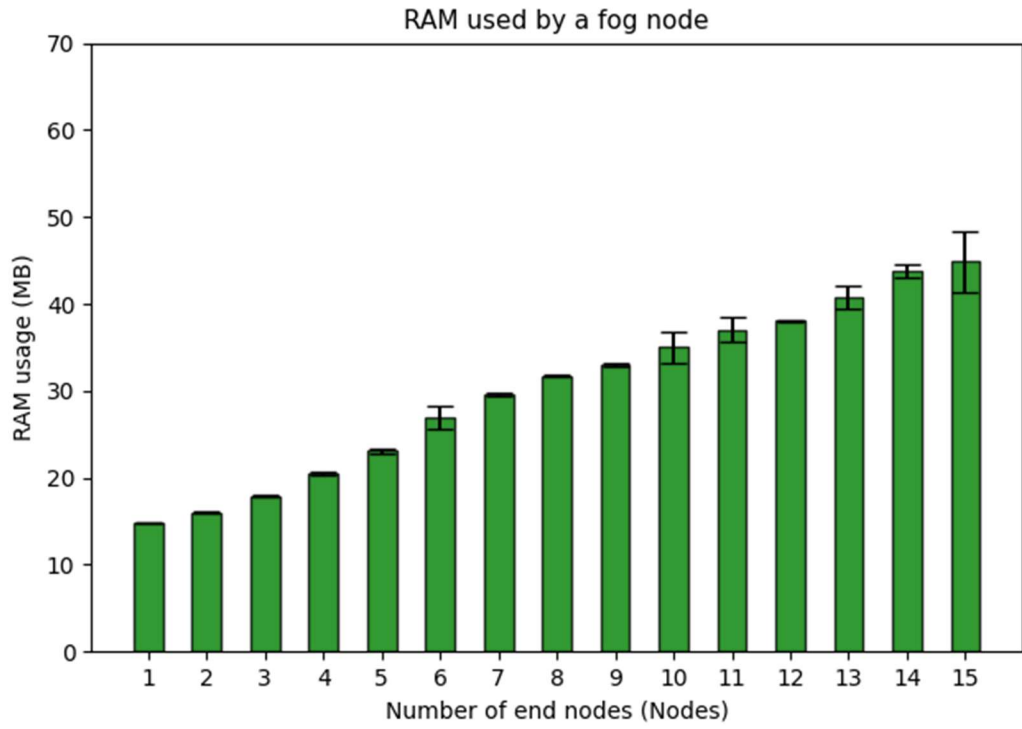


Figure 6.12: RAM used by a fog node when performing trust management for 1 to 15 trustee end nodes

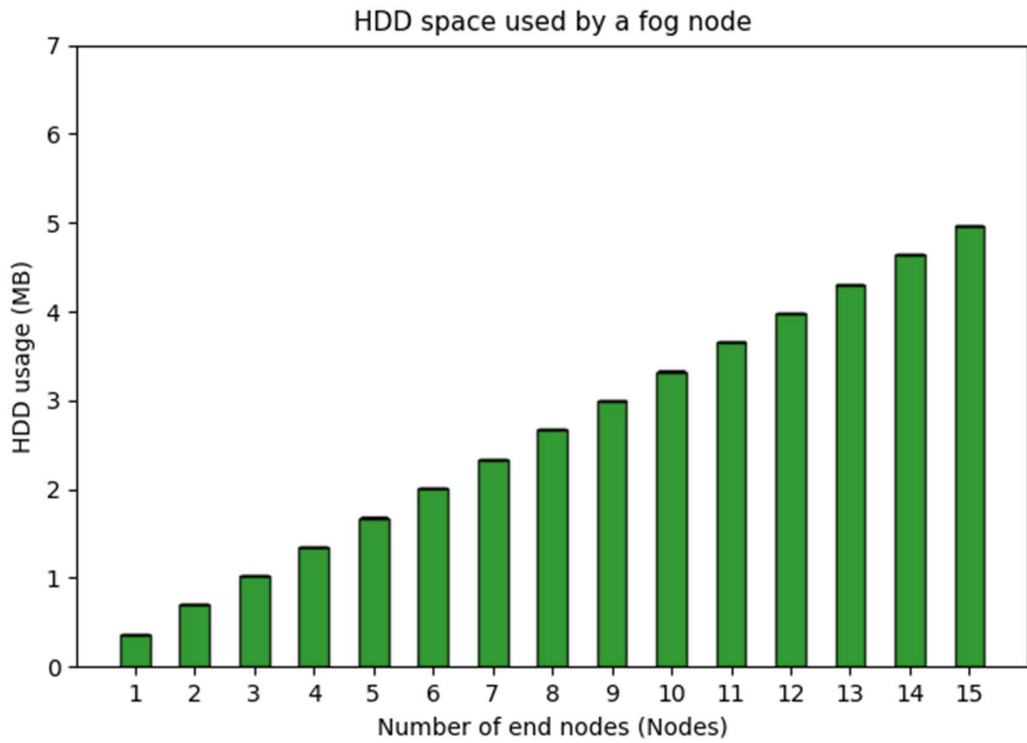


Figure 6.13: HDD used by a fog node when performing trust management for 1 to 15 trustee end nodes

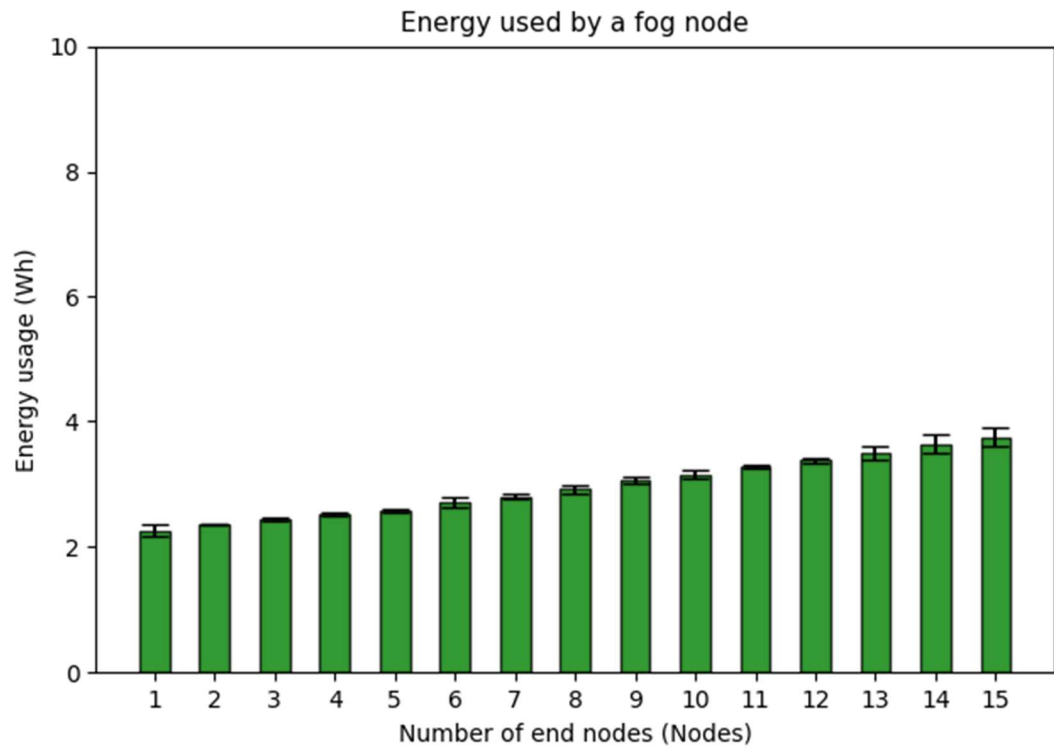


Figure 6.14: Energy used by a fog node when performing trust management for 1 to 15 trustee end nodes.

Table 6.11: CPU time used by a fog node when performing trust management for 1 to 15 trustee end nodes

No. end nodes (Nodes)	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
CPU Time (Seconds)	29	57.7	88.1	123	158	198	241	291	344	393	458	519	585	662	734
SD	0.297	0.571	0.882	1.86	2.31	0.762	1.14	2.15	2.94	5.86	2.21	1.89	3.52	3.09	4.19
CI (95%)	(28.3, 29.8)	(56.3, 59.2)	(85.9, 90.3)	(118, 128)	(152, 164)	(196, 199)	(238, 244)	(286, 296)	(336, 351)	(379, 408)	(452, 463)	(514, 524)	(577, 594)	(655, 670)	(724, 745)

Table 6.12: RAM used by a fog node when performing trust management for 1 to 15 trustee end nodes

No. end nodes (Nodes)	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
RAM usage (MB)	14.8	16	17.9	20.5	23.1	26.9	29.6	31.8	32.9	35.1	37.1	38.1	40.8	43.8	44.9
SD	0.00391	0.0421	0.0549	0.059	0.121	0.522	0.065	0.0678	0.0576	0.723	0.59	0.0193	0.552	0.29	1.42
CI (95%)	(14.8, 14.8)	(15.9, 16.1)	(17.8, 18.1)	(20.3, 20.6)	(22.8, 23.4)	(25.6, 28.2)	(29.4, 29.8)	(31.6, 31.9)	(32.8, 33.1)	(33.3, 36.8)	(35.6, 38.5)	(38, 38.1)	(39.4, 42.2)	(43.1, 44.5)	(41.3, 48.4)

Table 6.13: HDD used by a fog node when performing trust management for 1 to 15 trustee end nodes

No. end nodes (Nodes)	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
HDD usage (MB)	0.363	0.695	1.02	1.36	1.68	2.01	2.34	2.67	2.99	3.32	3.65	3.98	4.31	4.64	4.97

Table 6.14: Energy used by a fog node when performing trust management for 1 to 15 trustee end nodes

No. end nodes (Nodes)	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Energy usage (Wh)	2.26	2.36	2.43	2.52	2.58	2.71	2.8	2.92	3.05	3.15	3.27	3.38	3.49	3.64	3.75
SD	0.037	0.00227	0.0103	0.0148	0.00971	0.031	0.0121	0.028	0.0197	0.0273	0.012	0.0172	0.045	0.061	0.0594
CI (95%)	(2.17, 2.35)	(2.35, 2.37)	(2.41, 2.46)	(2.48, 2.55)	(2.56, 2.61)	(2.63, 2.78)	(2.77, 2.83)	(2.85, 2.99)	(3.01, 3.1)	(3.08, 3.21)	(3.24, 3.3)	(3.34, 3.43)	(3.37, 3.6)	(3.49, 3.79)	(3.6, 3.9)

6.5 Experiment 2: Resource consumption under different trust models and trust metric configurations

The second experiment examines the impact of different trust models and trust metric configurations on resource consumption. It compares local trust management on an end node using a fuzzy-logic model versus delegation, examines local trust management on fog nodes using weighted-sum and fuzzy-logic models, and analyses how different trust metrics affect resource usage on fog nodes. The experimental setup is divided into two parts due to the limited number of physical Raspberry Pi 4 and Raspberry Zero devices.

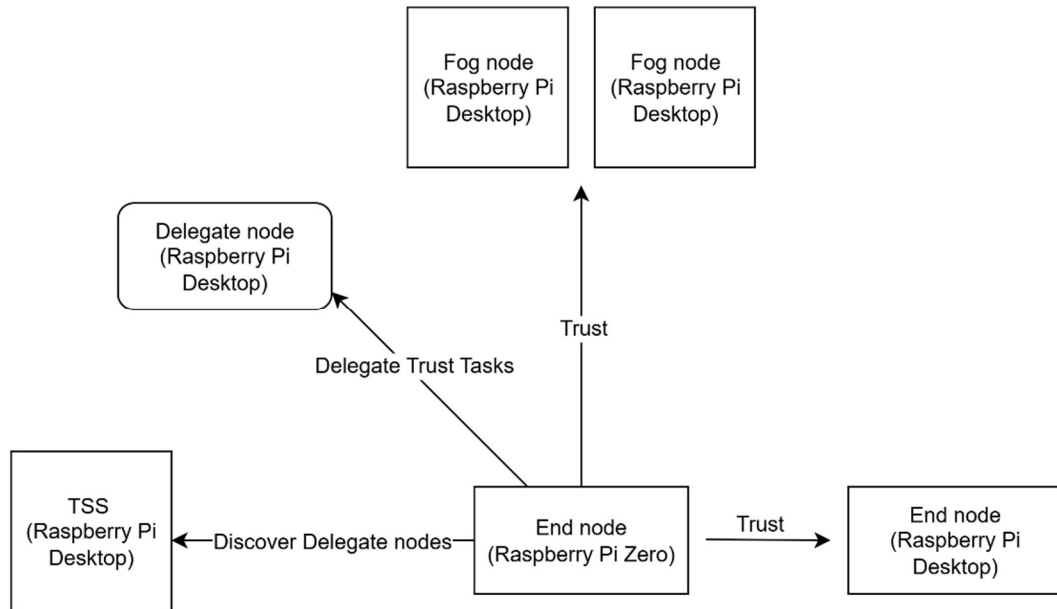


Figure 6.15: Experiment 2 setup for evaluating an end node

First, Section 6.5.1 aims to measure the resources consumed by an end node when performing local trust computation using fuzzy logic versus when delegating trust computation. In the experiment, the number of target peer end nodes and fog nodes ranges from 1 to 10. The experimental setup can be shown in Figure 6.15. A Raspberry Pi Zero serves as the end node, while the target end nodes and fog nodes are simulated using Raspberry Pi Desktop instances running on Oracle VirtualBox. The TSS and the delegate are deployed on separate virtual machines using Raspberry Pi Desktop.

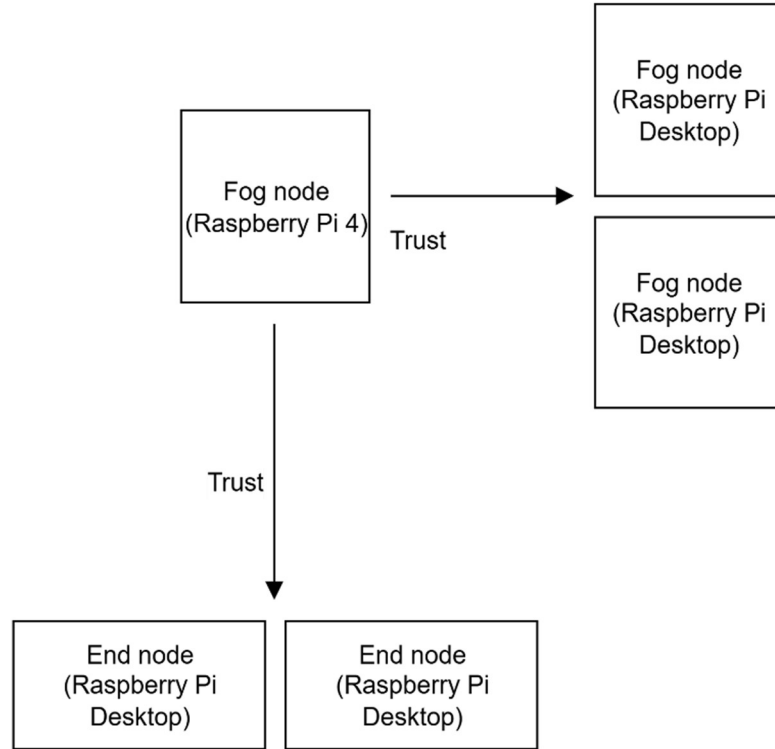


Figure 6.16: Experiment 2 setup for evaluating a fog node

Second, Section 6.5.2 and Section 6.5.3 evaluate the resource consumption of a fog node when assessing the trustworthiness of both end nodes and peer fog nodes locally. The experimental setup can be shown in Figure 6.16. In these experiments, a Raspberry Pi 4 is used as a fog node. The target end nodes and fog nodes are simulated on a Raspberry Pi Desktop, with their numbers varied from 1 to 10.

For trust evaluation, network latency, cooperativeness and reputability are used as trust metrics when assessing end nodes. In contrast, network latency and availability are used when assessing fog nodes. To simulate trust evidence collection, availability is determined by sending ping requests and recording the number of echo replies received. Network latency is measured using the round-trip time of these ping messages. Cooperativeness is simulated by assigning common friends and calculating the ratio of common friends shared between two nodes.

6.5.1 Resource consumption on an end node: Local trust management using a fuzzy logic trust model versus delegating trust management towards other end nodes and fog nodes

In this section, the resource consumption of an end node when performing trust management locally using a fuzzy-logic trust model is compared with delegating trust management towards targeted end nodes and fog nodes, with the number of each ranging from 1 to 10.

When handling trust locally, the end node performs all trust management tasks itself, including collecting trust evidence, updating the trust value, and making a decision by comparing the trust value against a threshold. In the delegation scenario, after selecting a delegate from the TSS, the end node collects evidence, stores it in its database, and then sends the trust evidence to the delegate. When a trust value is required, the end node sends a trust value request message with a threshold value for decision-making to the delegate to obtain the trust result towards the fog node. For both approaches, these processes are repeated sequentially every 10 seconds, for a total of 10 cycles.

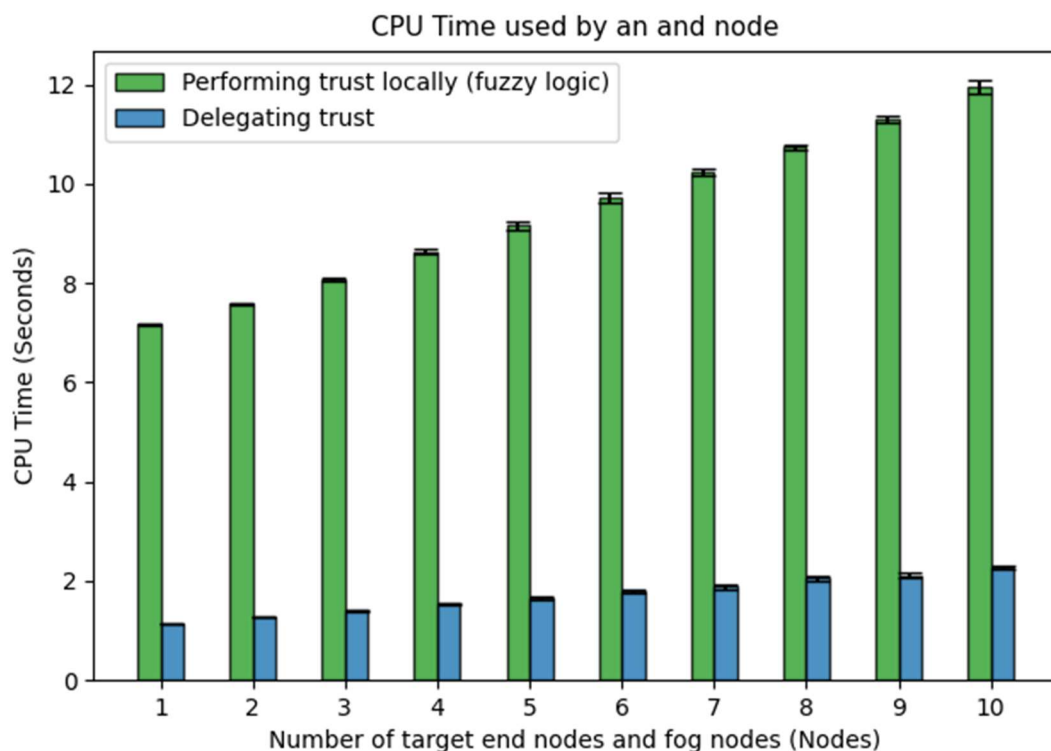


Figure 6.17: CPU time used by an end node when performing trust computation using fuzzy logic locally versus delegating trust computation

Table 6.15: CPU time used by an end node when performing trust computation using fuzzy logic locally versus delegating trust computation

No. target end & fog nodes (Nodes)	1	2	3	4	5	6	7	8	9	10
CPU Time for local trust (Seconds)	7.18	7.58	8.09	8.64	9.17	9.73	10.2	10.7	11.3	11.9
SD	0.021	0.029	0.055	0.069	0.113	0.149	0.084	0.076	0.092	0.196
CI (95%)	(7.16, 7.19)	(7.56, 7.6)	(8.05, 8.12)	(8.59, 8.69)	(9.09, 9.25)	(9.62, 9.84)	(10.2, 10.3)	(10.7, 10.8)	(11.2, 11.4)	(11.8, 12.1)
CPU time for delegation (Seconds)	1.14	1.28	1.4	1.54	1.65	1.79	1.88	2.05	2.12	2.28
SD	0.011	0.016	0.03	0.034	0.053	0.04	0.064	0.09	0.054	0.047
CI (95%)	(1.13, 1.15)	(1.26, 1.29)	(1.38, 1.43)	(1.51, 1.56)	(1.61, 1.68)	(1.76, 1.82)	(1.84, 1.93)	(1.99, 2.11)	(2.08, 2.15)	(2.25, 2.32)

Figure 6.17 and Table 6.15 present the CPU time required by the end node to perform trust management locally using the fuzzy-logic approach, compared with delegating trust management. Performing trust locally by fuzzy logic consumes substantially more CPU time than delegating trust. With 1 end node and 1 fog node, delegation takes 1.14 seconds, approximately 6.3 times less than the 7.18 seconds needed for local execution. With 10 end nodes and 10 fog nodes, performing trust locally consumes about 5.2 times more CPU time when compared to delegating trust. The result is as expected because fuzzy logic involves more computationally complex operations than the weighted-sum trust model. Therefore, delegation is effective in reducing processing burden on resource-constrained end nodes and conserving their limited processing resources for primary tasks. However, delegating trust requires less CPU time than local execution, but its CPU usage grows more sharply as the number of end and fog nodes increases. Delegating trust management increases from 1.14 seconds to 2.28 seconds between 1 and 10 nodes, a 100% increase, whereas performing trust locally rises from 7.18 seconds to 11.9 seconds, representing a 53.60% increase. This observation suggests that although delegation is more efficient than local execution, its advantage decreases as the system scales.

The amount of RAM used by the end node is presented in Figure 6.18 and Table 6.16. Delegating trust requires less RAM than performing trust locally. At 1 target end node and 1 fog node, delegating trust consumes 19.8 MB of RAM, compared to 55.7 MB when trust is performed locally, which is approximately 2.8 times less. As the number of target end nodes and fog nodes increases to 10, RAM usage for delegation rises to 21.3 MB, while local execution reaches 62.3 MB, which is 65.8% lower. Additionally, RAM consumption for both approaches rises slightly as the number of target end nodes and fog nodes grows from 1 to 10, rising by 7.78% for delegating trust and 11.9% for local trust management. This slight increase suggests that RAM usage is affected more by the trust evaluation method than by the number of nodes. The reason is that fuzzy logic requires more RAM than the weighted-sum approach because it stores membership functions and inference rules during evaluation.

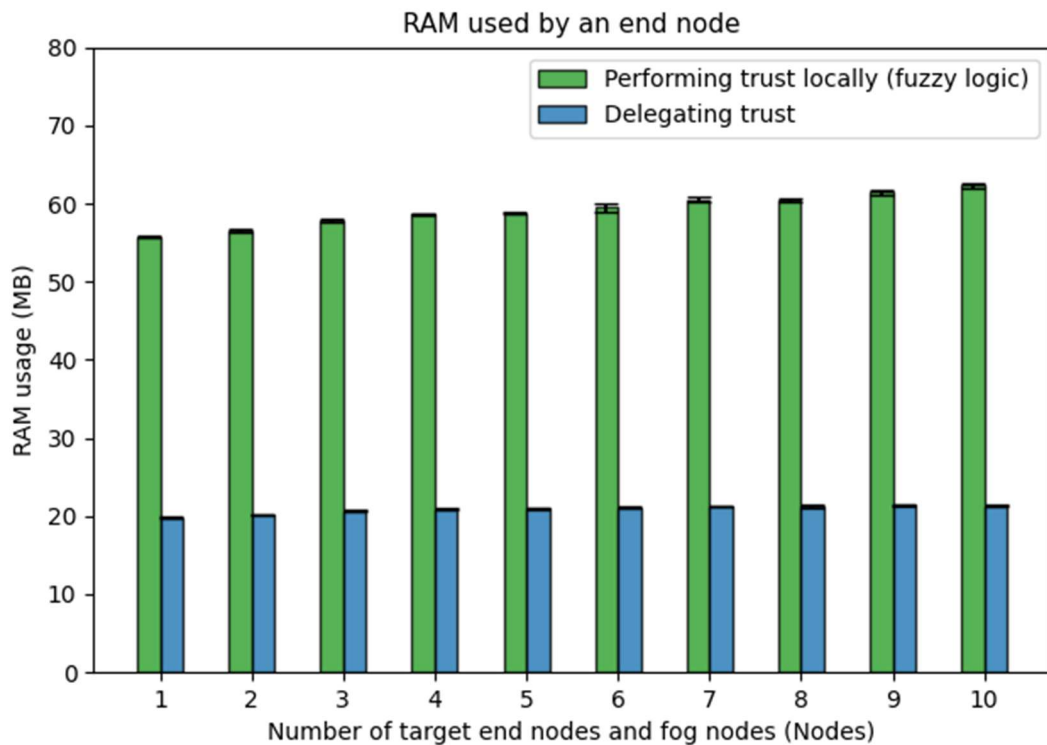


Figure 6.18: RAM used by an end node when performing trust computation using fuzzy logic locally versus delegating trust computation

Table 6.16: RAM used by an end node when performing trust computation using fuzzy logic locally versus delegating trust computation

No. target end & fog nodes (Nodes)	1	2	3	4	5	6	7	8	9	10
RAM for local trust (MB)	55.7	56.5	57.8	58.6	58.8	59.5	60.5	60.5	61.4	62.3
SD	0.142	0.237	0.336	0.256	0.204	0.722	0.469	0.375	0.493	0.381
CI (95%)	(55.6, 55.8)	(56.3, 56.7)	(57.6, 58.1)	(58.4, 58.8)	(58.6, 58.9)	(59, 60)	(60.2, 60.8)	(60.2, 60.7)	(61.1, 61.8)	(62, 62.6)
RAM for delegation (MB)	19.8	20.2	20.7	20.9	20.9	21.1	21.2	21.2	21.4	21.3
SD	0.11	0.06	0.08	0.1	0.17	0.15	0.1	0.19	0.18	0.21
CI (95%)	(19.7, 19.9)	(20.2, 20.3)	(20.6, 20.7)	(20.8, 21)	(20.8, 21.1)	(21, 21.2)	(21.2, 21.3)	(21.1, 21.4)	(21.3, 21.6)	(21.2, 21.5)

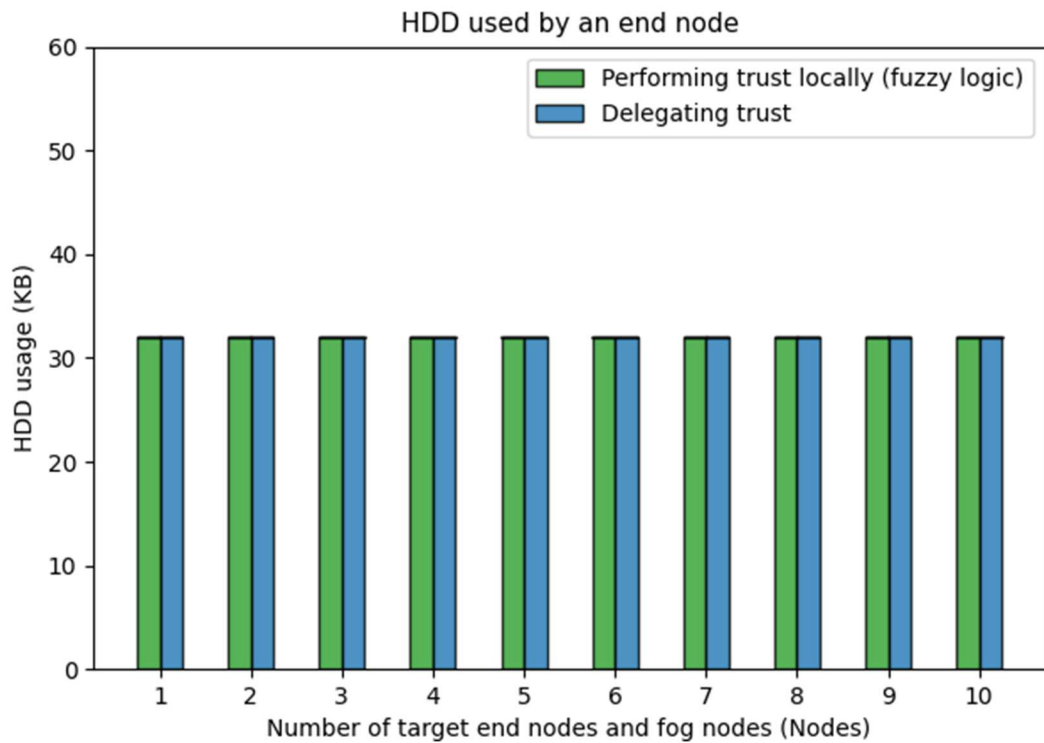


Figure 6.19: HDD used by an end node when performing trust computation using fuzzy logic locally versus delegating trust computation

Table 6.17: HDD used by an end node when performing trust computation using fuzzy logic locally versus delegating trust computation

No. target end & fog nodes (Nodes)	1	2	3	4	5	6	7	8	9	10
HDD for local trust (MB)	32	32	32	32	32	32	32	32	32	32
HDD for delegation (MB)	32	32	32	32	32	32	32	32	32	32

The amount of HDD space used for trust value and trust evidence storage remains constant throughout the experiment, as shown in Figure 6.19 and Table 6.17. The end node consumes 32 KB of HDD space for both local and delegating trust. This is because the trust evidence needs to be stored locally in both cases, resulting in the same HDD storage demand. Therefore, the storage demand at the end node is determined by the evidence collection role rather than by the trust computation models.

Figure 6.20 and Table 6.18 illustrate the energy consumption. Delegation results in significantly lower energy usage. For 1 target end node and 1 fog node, delegating trust consumes 1.15 mWh, whereas local execution requires 2.88, which is 150.4% or over 2.5 times less energy. Although both approaches show fluctuations in energy consumption, the overall trend is upward as the number of target end nodes and fog nodes grows. Interestingly, performing trust locally increases more sharply, 55.6% increases, while delegating trust increases by 53.9% as the number of target end nodes and fog nodes grows from 1 to 10. Similar to CPU time and RAM usage, delegating fuzzy logic trust computation significantly reduces energy consumption at the end node. This is particularly important in scenarios where end nodes are battery-powered devices, such as electric vehicles (EVs), that operate under limited energy. These results support that complex trust computation should be delegated to the fog layer. As a result, constrained end nodes can conserve their own resources and energy for essential main tasks.

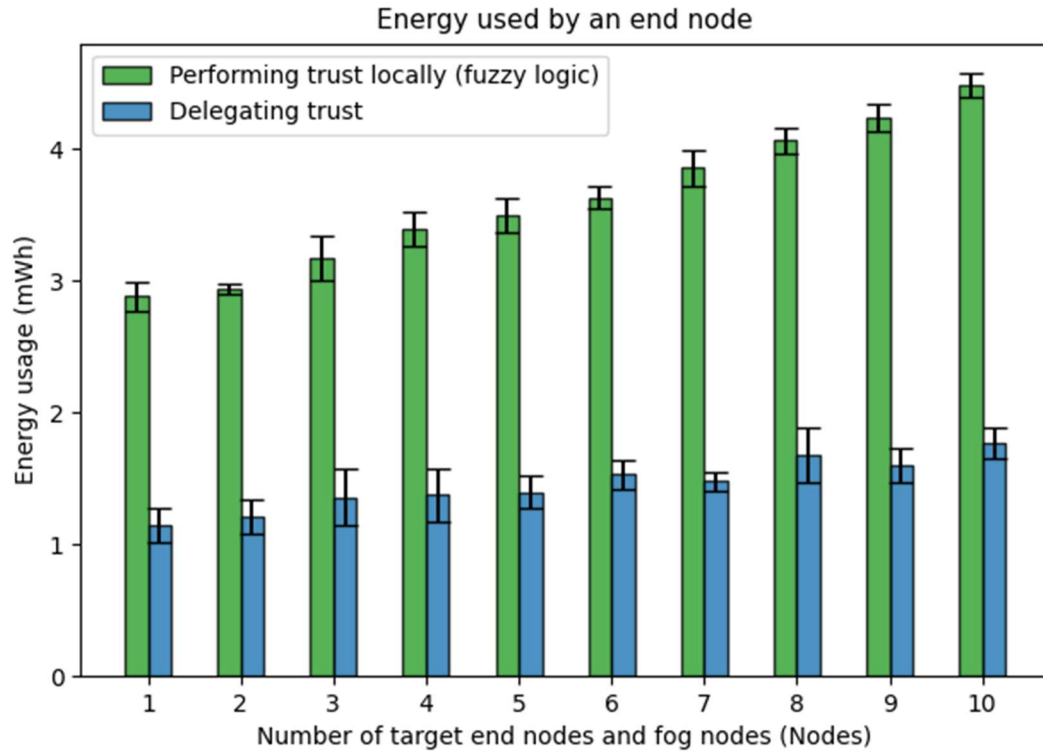


Figure 6.20: Energy used by an end node when performing trust computation using fuzzy logic locally versus delegating trust computation

Table 6.18: Energy used by an end node when performing trust computation using fuzzy logic locally versus delegating trust computation

No. target end & fog nodes (Nodes)	1	2	3	4	5	6	7	8	9	10
Energy for local trust (mWh)	2.88	2.94	3.18	3.39	3.49	3.63	3.86	4.07	4.24	4.48
SD	0.157	0.055	0.239	0.182	0.183	0.115	0.196	0.132	0.145	0.13
CI (95%)	(2.77, 2.99)	(2.91, 2.98)	(3.01, 3.35)	(3.26, 3.52)	(3.36, 3.62)	(3.55, 3.72)	(3.72, 4)	(3.97, 4.16)	(4.14, 4.34)	(4.39, 4.57)
Energy for delegation (mWh)	1.15	1.22	1.36	1.37	1.4	1.53	1.48	1.68	1.6	1.77
SD	0.181	0.182	0.294	0.274	0.175	0.16	0.097	0.289	0.177	0.161
CI (95%)	(1.02, 1.28)	(1.08, 1.35)	(1.15, 1.57)	(1.18, 1.57)	(1.27, 1.52)	(1.42, 1.64)	(1.41, 1.55)	(1.47, 1.89)	(1.47, 1.73)	(1.66, 1.89)

6.5.2 Resource consumption on a fog node: Comparing weighted sum and fuzzy logic-based trust models for evaluating end nodes and peer fog nodes

The primary objective of this experiment is to study the impact of performing trust management operations on a high-resource fog node and to assess the additional computational overhead introduced by the fuzzy logic. The fog node, a more powerful device, is responsible for evaluating the trustworthiness of both end nodes and peer fog nodes simultaneously. It locally performs all trust management tasks towards each target node separately every 10 seconds and 10 times.

As shown in Figure 6.21 and Table 6.19, the fuzzy logic method consistently required more CPU time than the weighted sum approach, with a noticeable increase as the number of nodes grew. For 1 to 10 nodes, the CPU time for fuzzy logic increases from 2.21 to 3.32 seconds, a 50.2% increase, while the weighted sum approach increases from 1.52 to 1.90 seconds, a 25% increase. When comparing the two methods, fuzzy logic required 45.4% more CPU time than the weighted sum for 1 end node and 1 fog node, and this gap rises to 74.7% 10 end nodes and fog nodes.

The amount of RAM in Figure 6.22 and Table 6.20 illustrate that the fuzzy logic approach requires more RAM than the weighted sum throughout the experiment. However, the fuzzy logic approach does not require much additional RAM compared to CPU time as the number of target nodes increases, only 5.25% more RAM than the weighted sum with 1 end node and fog node, and 7.87% more with 10 nodes. As the number of end nodes and fog nodes increases from 1 to 10, RAM usage rises slightly for both approaches, by 6.54% for fuzzy logic and 3.96% for weighted sum. Although the fuzzy-logic approach requires significantly more CPU time than the weighted-sum approach, the RAM requirement is slightly higher, suggesting that the main resource challenge is processing power rather than memory. This finding helps identify that CPU resources must be allocated to fog nodes in real-world deployments. In particular, fog node hardware should be designed with stronger processing capabilities rather than greater RAM capacity when supporting fuzzy-logic trust evaluation.

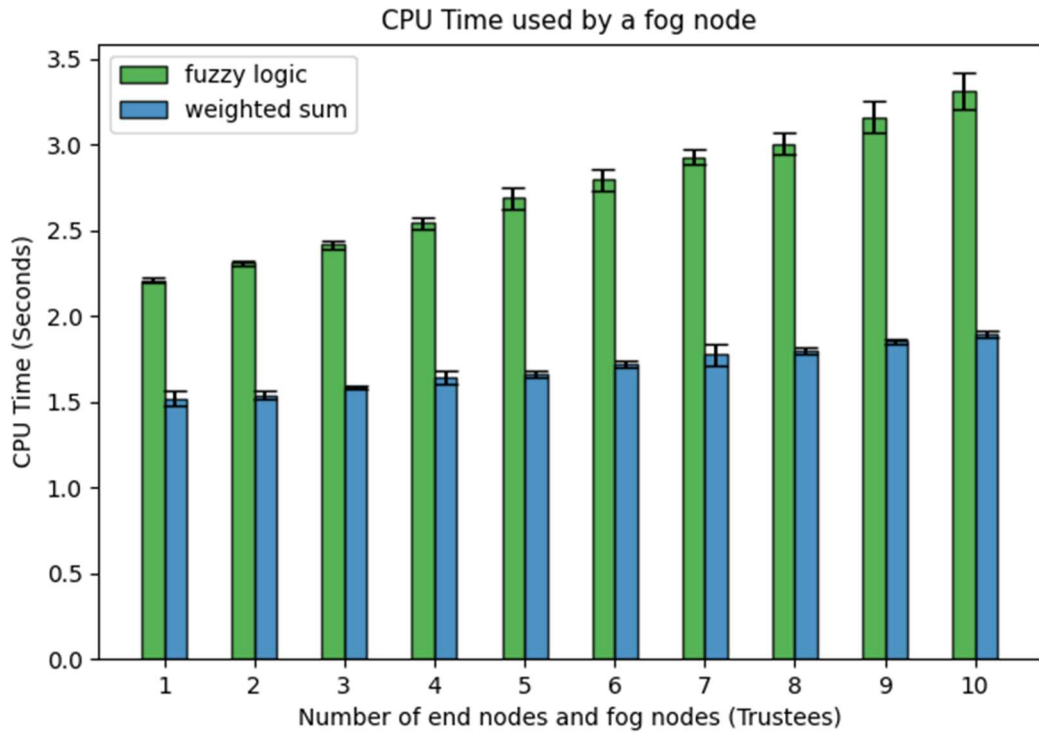


Figure 6.21: CPU time used by a fog node when performing trust computation locally using weighted sum versus fuzzy logic

Table 6.19: CPU time used by a fog node when performing trust computation locally using weighted sum versus fuzzy logic

No. end & fog nodes (Nodes)	1	2	3	4	5	6	7	8	9	10
CPU time for fuzzy logic (Seconds)	2.21	2.31	2.42	2.54	2.69	2.8	2.93	3	3.16	3.31
SD	0.016	0.012	0.021	0.035	0.061	0.06	0.046	0.063	0.089	0.104
CI (95%)	(2.2, 2.22)	(2.3, 2.32)	(2.4, 2.43)	(2.52, 2.57)	(2.65, 2.73)	(2.75, 2.84)	(2.9, 2.96)	(2.96, 3.05)	(3.1, 3.23)	(3.24, 3.39)
CPU time for weighted sum (Seconds)	1.52	1.54	1.58	1.64	1.67	1.72	1.78	1.8	1.85	1.9
SD	0.043	0.02	0.007	0.04	0.019	0.017	0.064	0.021	0.013	0.021
CI (95%)	(1.49, 1.55)	(1.53, 1.56)	(1.58, 1.59)	(1.61, 1.67)	(1.65, 1.68)	(1.71, 1.73)	(1.73, 1.82)	(1.78, 1.81)	(1.84, 1.86)	(1.88, 1.91)

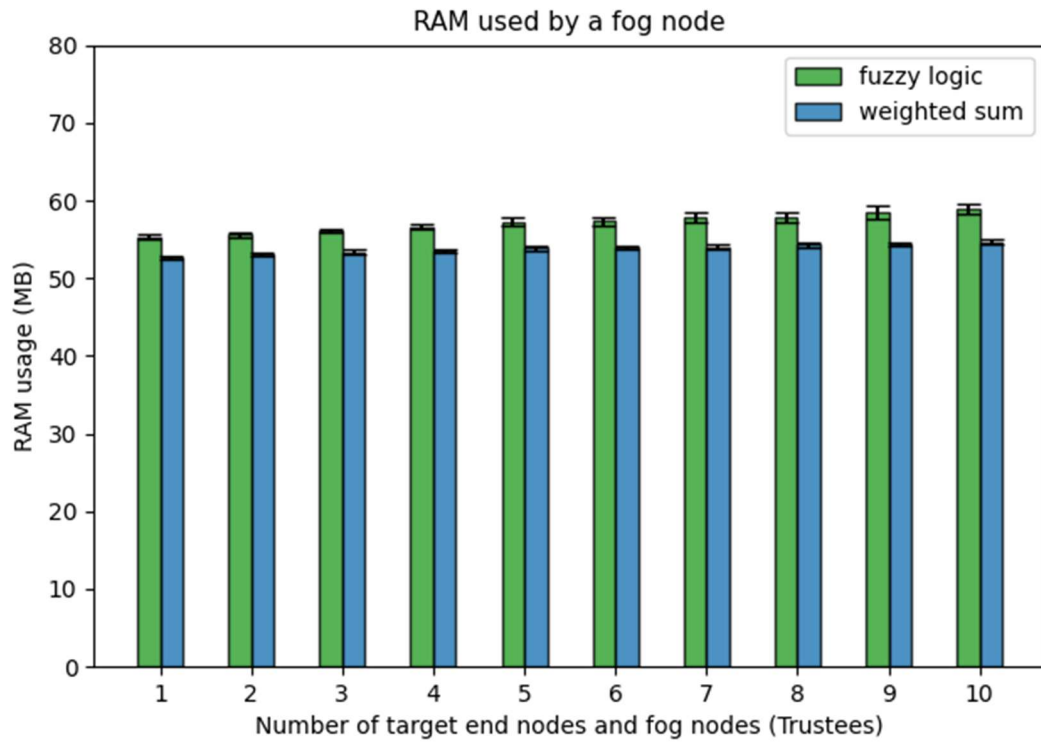


Figure 6.22: RAM used by a fog node when performing trust computation locally using weighted sum versus fuzzy logic

Table 6.20: RAM used by a fog node when performing trust computation locally using weighted sum versus fuzzy logic

No. end & fog nodes (Nodes)	1	2	3	4	5	6	7	8	9	10
RAM for fuzzy logic (MB)	55.3	55.6	56.1	56.6	57.2	57.3	57.8	57.8	58.5	58.9
SD	0.248	0.308	0.28	0.365	0.536	0.607	0.655	0.657	0.933	0.642
CI (95%)	(55.2, 55.5)	(55.4, 55.9)	(55.9, 56.3)	(56.3, 56.9)	(56.9, 57.6)	(56.9, 57.8)	(57.3, 58.3)	(57.4, 58.3)	(57.8, 59.1)	(58.5, 59.4)
RAM for weighted sum (MB)	52.6	53.1	53.4	53.5	53.9	54	54	54.3	54.5	54.6
SD	0.22	0.246	0.315	0.27	0.294	0.289	0.23	0.281	0.223	0.3
CI (95%)	(52.4, 52.7)	(52.9, 53.3)	(53.2, 53.6)	(53.3, 53.7)	(53.6, 54.1)	(53.8, 54.2)	(53.9, 54.2)	(54.1, 54.5)	(54.3, 54.6)	(54.4, 54.9)

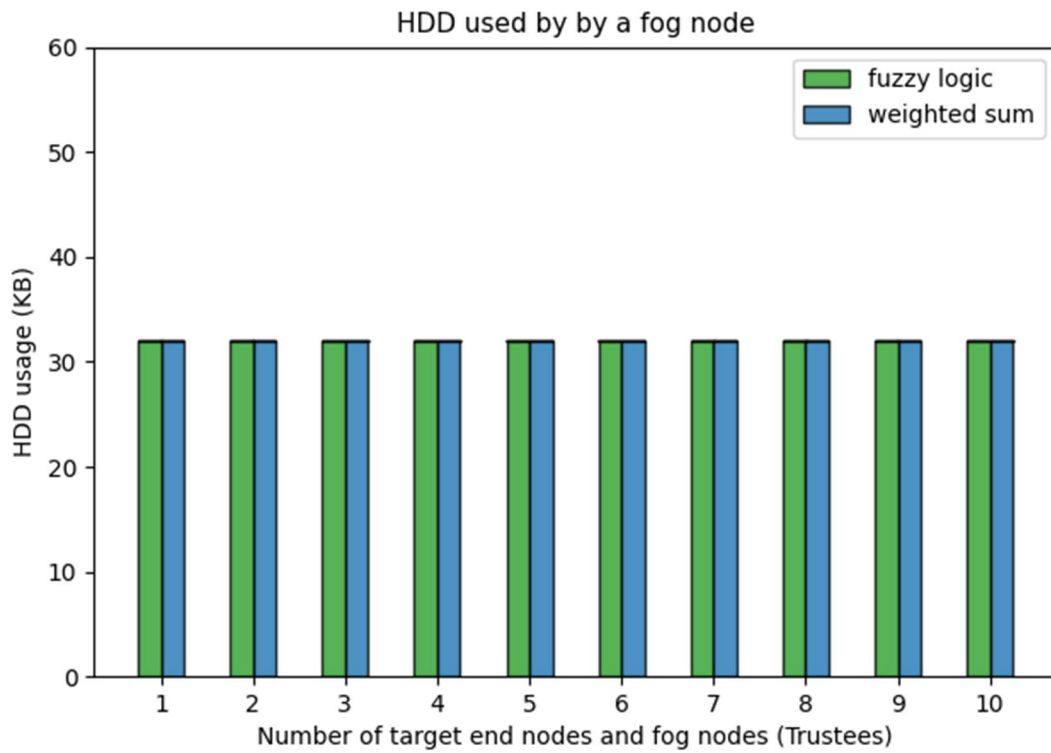


Figure 6.23: HDD used by a fog node when performing trust computation locally using weighted sum versus fuzzy logic

Table 6.21: HDD used by a fog node when performing trust computation locally using weighted sum versus fuzzy logic

No. end & fog nodes (nodes)	1	2	3	4	5	6	7	8	9	10
HDD for Fuzzy logic (KB)	32	32	32	32	32	32	32	32	32	32
HDD for Weighted sum (KB)	32	32	32	32	32	32	32	32	32	32

Regarding HDD usage, the fog node uses 32 KB of HDD space to store trust evidence and trust values for both approaches towards end nodes and other fog nodes throughout the experiment, as illustrated in Figure 6.23 and Table 6.21.

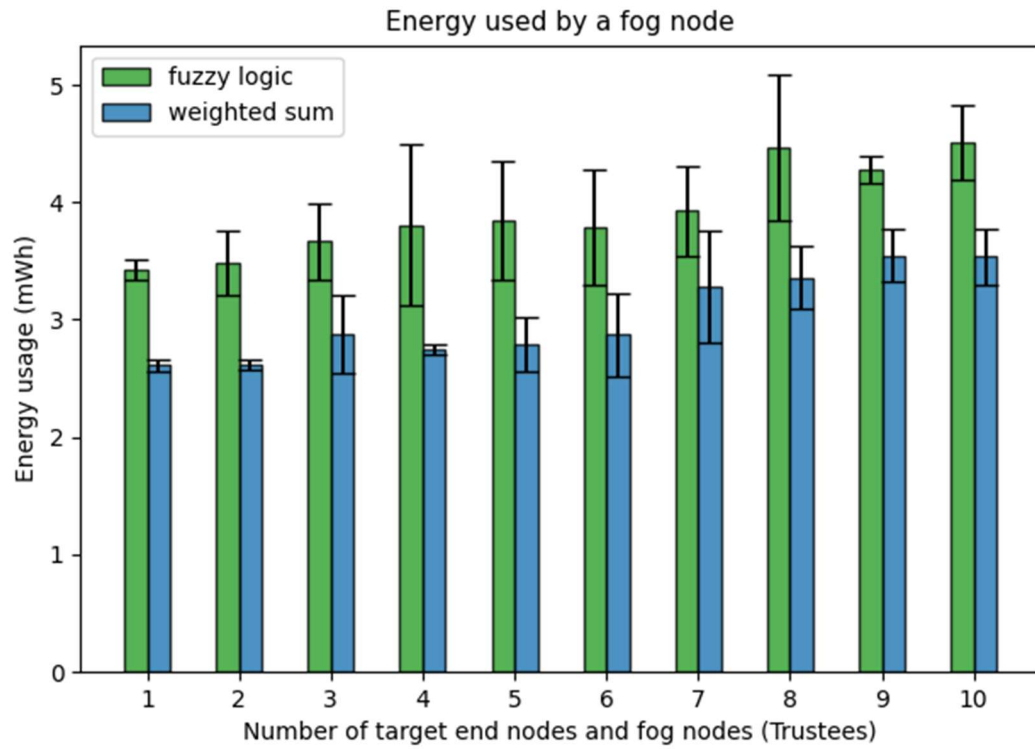


Figure 6.24: Energy used by a fog node when performing trust computation locally using weighted sum versus fuzzy logic

Table 6.22: Energy used by a fog node when performing trust computation locally using weighted sum versus fuzzy logic

No. end & fog nodes (Nodes)	1	2	3	4	5	6	7	8	9	10
Energy for fuzzy logic (mWh)	3.43	3.49	3.67	3.81	3.84	3.79	3.92	4.47	4.28	4.51
SD	0.085	0.272	0.32	0.689	0.507	0.494	0.38	0.618	0.112	0.324
CI (95%)	(3.37, 3.49)	(3.29, 3.68)	(3.44, 3.89)	(3.32, 4.3)	(3.48, 4.21)	(3.44, 4.14)	(3.65, 4.2)	(4.02, 4.91)	(4.2, 4.36)	(4.28, 4.74)
Energy for weighted sum (mWh)	2.61	2.62	2.87	2.75	2.79	2.87	3.28	3.36	3.54	3.54
SD	0.052	0.044	0.331	0.044	0.229	0.353	0.474	0.267	0.222	0.239
CI (95%)	(2.57, 2.65)	(2.59, 2.65)	(2.64, 3.11)	(2.72, 2.78)	(2.62, 2.95)	(2.62, 3.13)	(2.94, 3.62)	(3.17, 3.55)	(3.39, 3.7)	(3.37, 3.71)

Figure 6.24 and Table 6.22 present energy used by the fog node to perform trust management locally by using weighted sum and fuzzy logic-based trust models towards end nodes and other fog nodes simultaneously. While some fluctuation is observed due to measurement precision, there is a clear upward trend in energy consumption as the number of end nodes and fog nodes increases. Moreover, it is clear that the fuzzy logic approach consumes more energy than the weighted sum method. This is consistent with the CPU time results. The higher computational complexity of fuzzy logic leads to increased processing activity and higher energy usage. From a practical perspective, this finding shows that energy demand is also an important consideration when deploying fuzzy-logic trust evaluation on fog nodes.

6.5.3 Resource consumption on a fog node: Comparison of trust evaluation towards end nodes versus towards peer fog nodes with different trust properties

This section compares the resources consumed by a fog node when performing trust management towards end nodes versus towards peer fog nodes with different trust properties. The hypothesis is that if the fog node performs trust management towards different entities with different trust metrics, it will consume different amounts of resources. The CPU time used by the fog node is presented in Figure 6.25 and Table 6.23. CPU time increases for both the end node and the fog node evaluations as the number of target nodes rises from 1 to 10, indicating that trust management becomes more computationally demanding at larger scales on the fog node. However, CPU time is slightly higher when the fog node evaluates end nodes than when it evaluates peer fog nodes. The fog node requires 2.17 seconds for evaluating 1 end node, which is 1.4% more than the 2.14 seconds needed to evaluate 1 fog node. Notably, the rate of increase is higher when assessing end nodes. With 10 end nodes, the fog node spends 0.41 seconds more CPU time than with 10 fog nodes, an increase of 16.7%. Because different trust metrics are used to evaluate end nodes and fog nodes, this result highlights the impact of trust metrics on fog node resources. In other words, computational overhead depends not only on the number of trustees being evaluated but also on the number and complexity of trust metrics used in the evaluation process.

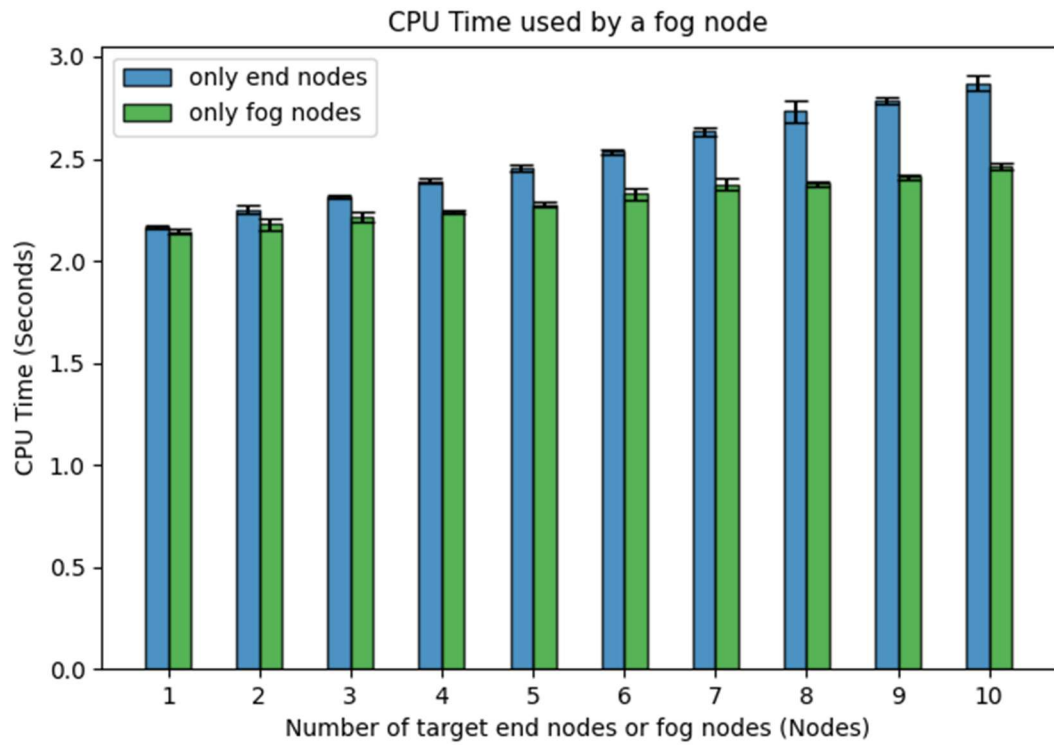


Figure 6.25: CPU time used by a fog node on evaluating trust of end nodes versus evaluating trust of fog nodes with different trust properties

Table 6.23: CPU time used by a fog node on evaluating trust of end nodes versus evaluating trust of fog nodes with different trust properties

No. end nodes or fog nodes	1	2	3	4	5	6	7	8	9	10
CPU time for end nodes (Seconds)	2.17	2.25	2.31	2.39	2.46	2.54	2.63	2.73	2.79	2.87
SD	0.008	0.031	0.016	0.019	0.02	0.02	0.028	0.073	0.025	0.052
CI (95%)	(2.16, 2.17)	(2.23, 2.27)	(2.3, 2.33)	(2.38, 2.4)	(2.44, 2.47)	(2.52, 2.55)	(2.61, 2.66)	(2.68, 2.79)	(2.77, 2.8)	(2.83, 2.91)
CPU time for fog nodes (Seconds)	2.14	2.18	2.21	2.24	2.27	2.33	2.37	2.38	2.41	2.46
SD	0.017	0.042	0.03	0.012	0.016	0.042	0.039	0.015	0.018	0.025
CI (95%)	(2.13, 2.15)	(2.15, 2.21)	(2.19, 2.24)	(2.23, 2.25)	(2.26, 2.29)	(2.3, 2.36)	(2.35, 2.4)	(2.37, 2.39)	(2.4, 2.42)	(2.45, 2.48)

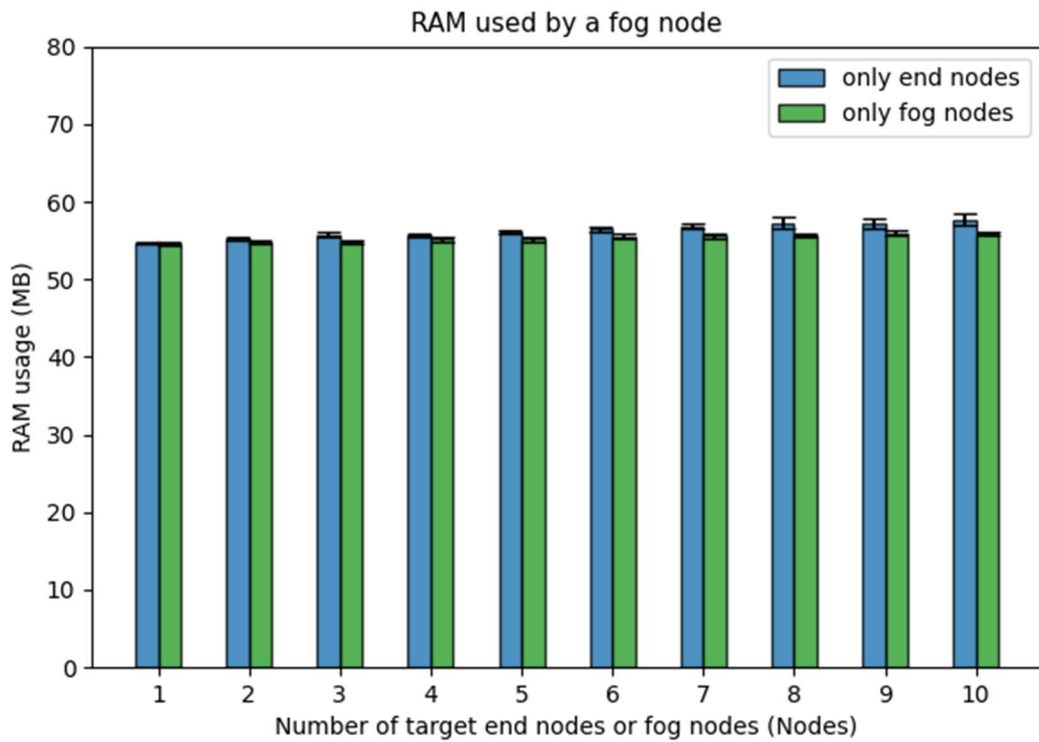


Figure 6.26: RAM used by a fog node on evaluating trust of end nodes versus evaluating trust of fog nodes with different trust properties

Table 6.24: RAM used by a fog node on evaluating trust of end nodes versus evaluating trust of fog nodes with different trust properties

No. end nodes or fog nodes	1	2	3	4	5	6	7	8	9	10
RAM for end nodes (MB)	54.7	55.2	55.7	55.7	56.1	56.5	56.8	57.3	57.2	57.7
SD	0.203	0.282	0.306	0.276	0.231	0.266	0.349	0.712	0.607	0.783
CI (95%)	(54.6, 54.8)	(55, 55.4)	(55.5, 56)	(55.5, 55.9)	(56, 56.3)	(56.3, 56.7)	(56.6, 57.1)	(56.8, 57.8)	(56.8, 57.7)	(57.2, 58.3)
RAM for fog nodes (MB)	54.6	54.8	54.8	55.1	55.2	55.5	55.6	55.7	56	55.9
SD	0.277	0.271	0.222	0.228	0.309	0.297	0.324	0.281	0.252	0.306
CI (95%)	(54.4, 54.8)	(54.6, 55)	(54.7, 55)	(55, 55.3)	(55, 55.4)	(55.3, 55.7)	(55.4, 55.8)	(55.5, 55.9)	(55.8, 56.1)	(55.7, 56.1)

A similar pattern can be observed in the RAM results shown in Figure 6.26 and Table 6.24. Evaluating end nodes consistently required more memory than evaluating fog nodes. An additional 0.14 MB is needed for a single end node and 1.82 MB more for 10 end nodes. As the number of end nodes increases from 1 to 10, the total additional RAM required is 3.01 MB, whereas evaluating 1 to 10 fog nodes requires only an additional 1.34 MB. This suggests that the additional trust metrics used for end node evaluation increase not only processing costs but also the amount of memory required to maintain trust data during evaluation. However, the overall increase remains negligible, indicating that the additional RAM overhead is not a major concern in this scenario. Figure 6.27 and Table 6.25 indicate that the HDD space used by the fog node to store evidence and trust values remains constant at 32 KB throughout the experiments, regardless of whether it is managing trust for end nodes or for fog nodes.

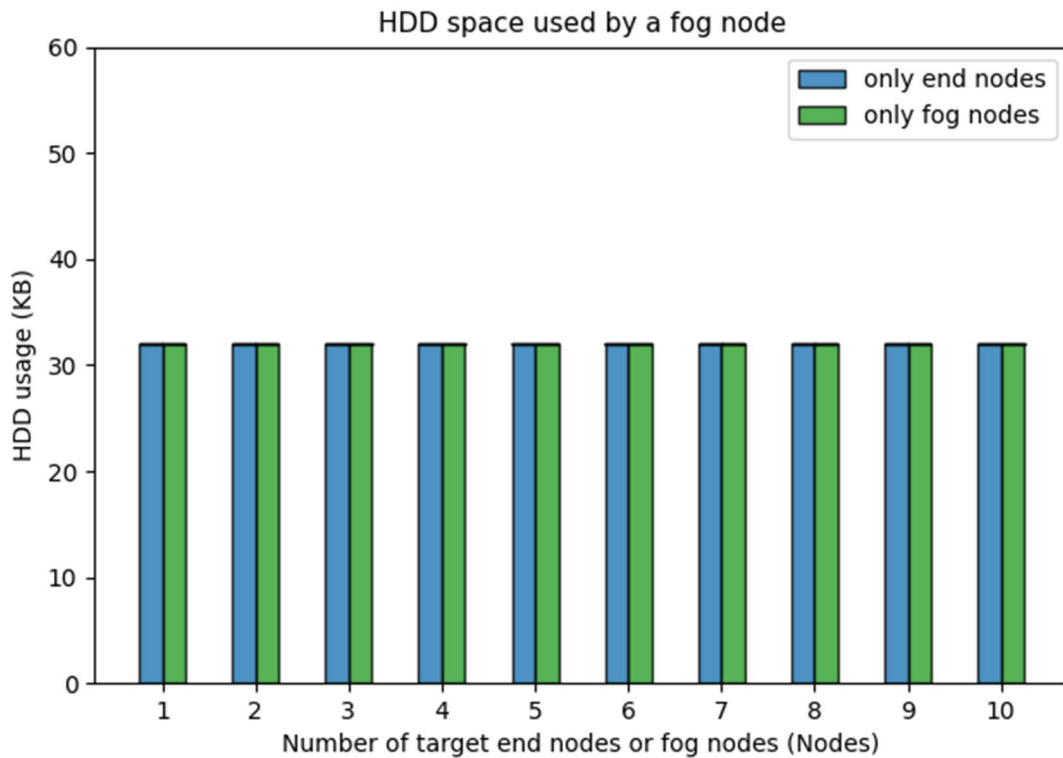


Figure 6.27: HDD used by a fog node on evaluating trust of end nodes versus evaluating trust of fog nodes with different trust properties

Table 6.25: HDD used by a fog node on evaluating trust of end nodes versus evaluating trust of fog nodes with different trust properties

No. end nodes or fog nodes	1	2	3	4	5	6	7	8	9	10
HDD usage for end nodes (MB)	32	32	32	32	32	32	32	32	32	32
HDD usage for fog nodes (MB)	32	32	32	32	32	32	32	32	32	32

The energy consumption of the fog node is presented in Figure 6.28 and Table 6.26. Although the results show some fluctuations due to the limited precision of the voltage and current measurement tool, the overall trend indicates that evaluating end nodes consumes more energy than evaluating fog nodes. Additionally, energy consumption for both cases increases as the number of end nodes or fog nodes rises. This pattern is consistent with the earlier CPU-time and RAM results and suggests that the higher processing and memory demands of end node evaluation lead to higher energy consumption. Furthermore, the results indicate that energy overhead also depends on the type and complexity of the trust metrics used in the evaluation process.

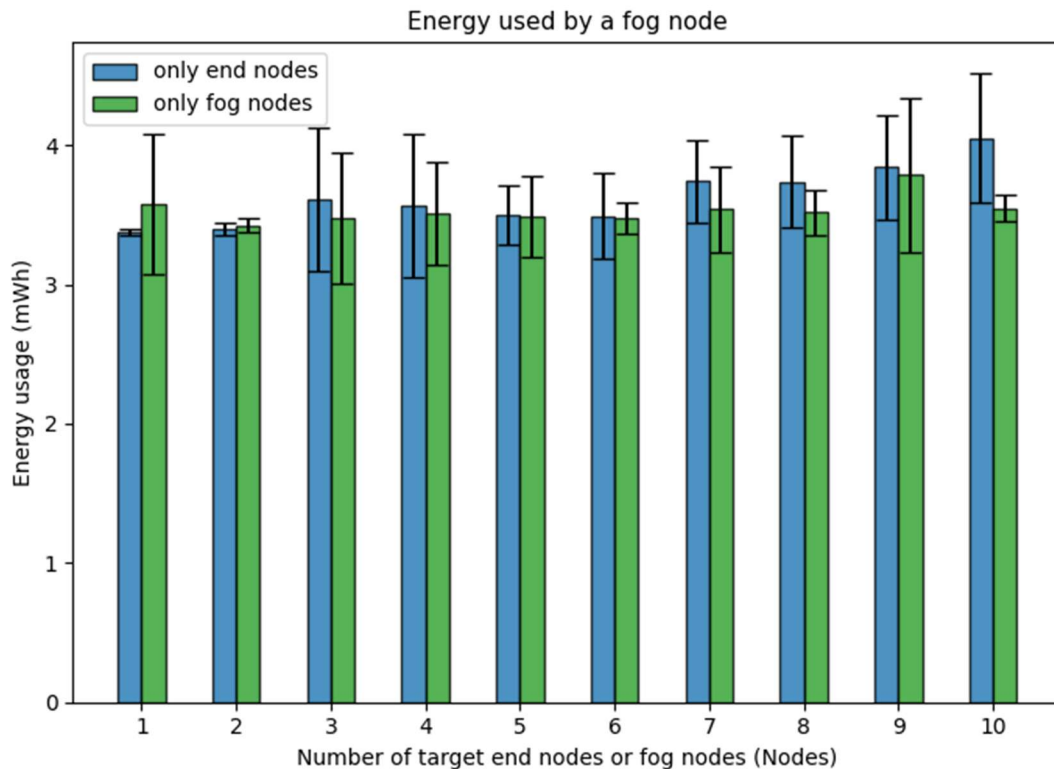


Figure 6.28: Energy used by a fog node on evaluating trust of end nodes versus evaluating trust of fog nodes with different trust properties

Table 6.26: Energy used by a fog node on evaluating trust of end nodes versus evaluating trust of fog nodes with different trust properties

No. end nodes or fog nodes	1	2	3	4	5	6	7	8	9	10
Energy for end nodes (mWh)	3.37	3.4	3.61	3.57	3.49	3.49	3.74	3.74	3.84	4.05
SD	0.022	0.048	0.513	0.515	0.213	0.307	0.299	0.332	0.372	0.465
CI (95%)	(3.36, 3.39)	(3.36, 3.43)	(3.24, 3.98)	(3.2, 3.94)	(3.34, 3.65)	(3.27, 3.71)	(3.53, 3.95)	(3.5, 3.98)	(3.57, 4.11)	(3.72, 4.38)
Energy for fog nodes (mWh)	3.58	3.42	3.48	3.51	3.48	3.48	3.54	3.52	3.79	3.55
SD	0.505	0.047	0.469	0.365	0.292	0.113	0.309	0.165	0.552	0.092
CI (95%)	(3.21, 3.94)	(3.39, 3.46)	(3.14, 3.81)	(3.25, 3.77)	(3.28, 3.69)	(3.4, 3.56)	(3.32, 3.76)	(3.4, 3.64)	(3.39, 4.18)	(3.48, 3.61)

6.6 Experiment 3: Resource consumption under different evidence collection and trust model configurations

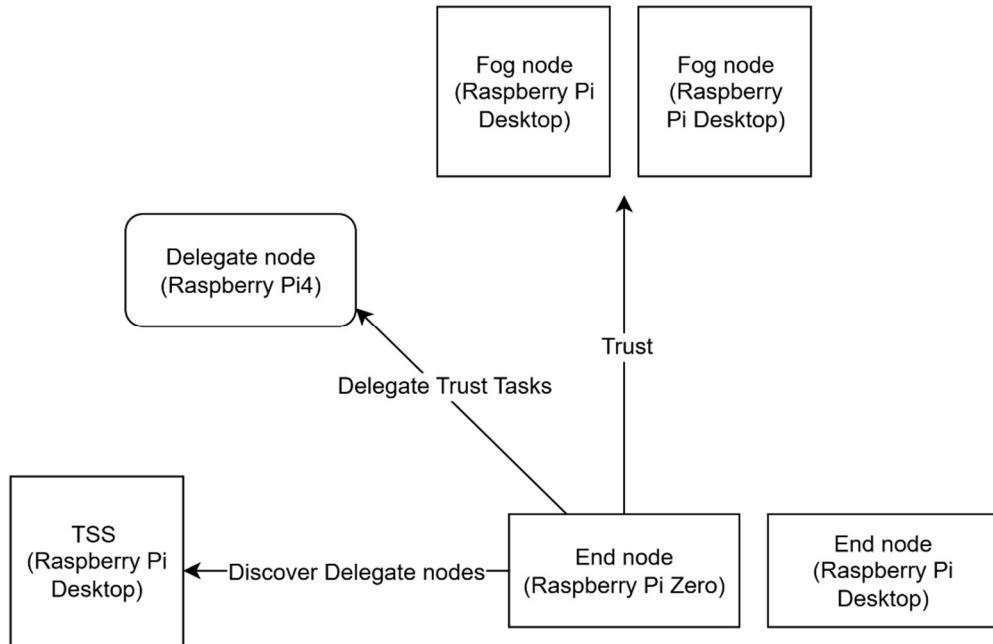


Figure 6.29: Experiment 3 setup

The experimental setup shown in Figure 6.29 is intended to evaluate the resource consumption associated with evidence collection on both the end node and the delegate. It also compares the costs of the weighted-sum and fuzzy-logic trust models on both the end node and the delegate. For trust evaluation, distance, response time, and availability are used to assess the fog nodes. In this setup, the end node is implemented using a Raspberry Pi Zero, while additional end nodes are simulated using Raspberry Pi Desktop instances. The delegate node is represented by a Raspberry Pi 4. The fog nodes and the TSS are emulated using Raspberry Pi Desktop on Oracle VirtualBox. The details of each experimental setting will be described within the relevant sections.

6.6.1 Resource consumption on an end node performing trust management delegation with self-evidence collection and delegated evidence collection

This section investigates whether evidence collection creates a significant burden on a constrained end node. The resource consumption of the end node is compared under two delegation conditions. First, the end node collects trust evidence itself. Second, the delegate is responsible for collecting it. In other words, the end node delegates trust computer and trust repositor roles in the first case, and it delegates all trust management roles in the second case. To clearly evaluate the impact of evidence collection, the experiment assumes that fog nodes or trustees behave consistently with different entities, and the delegate resides on the same network as the end node.

In performing trust management with self-evidence collection, the end node gathers trust evidence, including distance, response time, and availability and transmits them to the delegate every 10 seconds. In delegated evidence collection, the delegate takes full responsibility for collecting all trust evidence on behalf of the end node. The delegate then performs the trust calculation using the weighted-sum trust model. These processes run at 10-second intervals. The end node periodically requests the trust evaluation result from the delegate every 10 seconds.

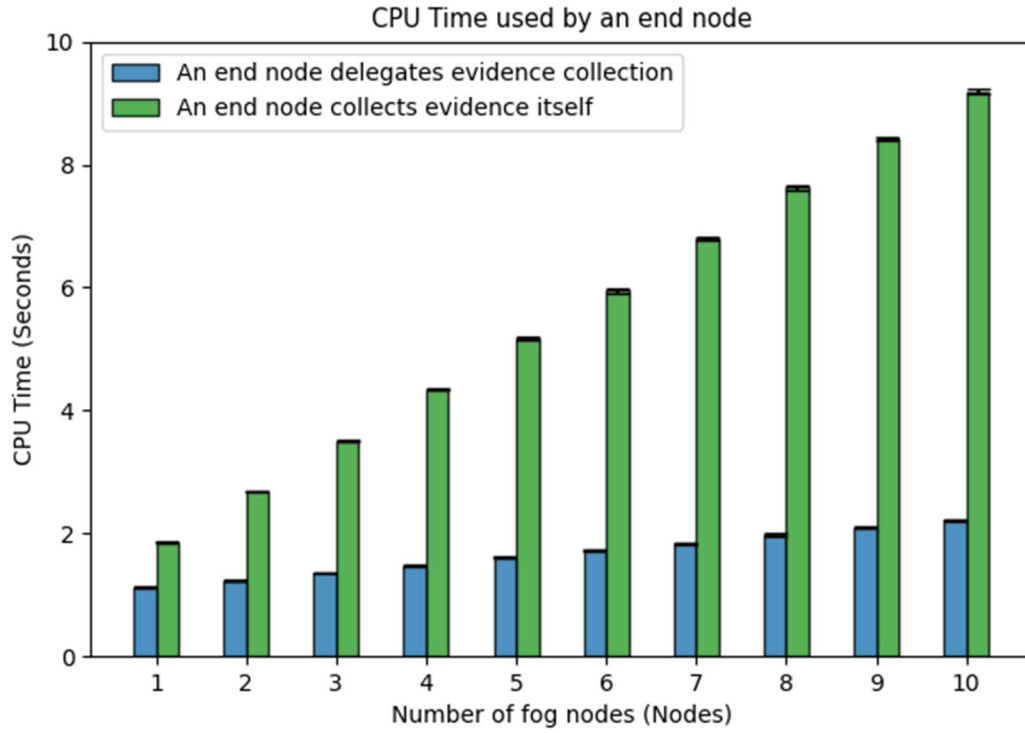


Figure 6.30: CPU time used by an end node performing trust management delegation with self-evidence collection and delegated evidence collection

Table 6.27: CPU time used by an end node performing trust management delegation with self-evidence collection and delegated evidence collection

No. fog nodes (Nodes)	1	2	3	4	5	6	7	8	9	10
CPU time for delegating collection (Seconds)	1.12	1.23	1.36	1.48	1.6	1.72	1.83	1.98	2.1	2.21
SD	0.0137	0.00632	0.0129	0.0143	0.0217	0.0184	0.0255	0.0267	0.0235	0.0303
CI (95%)	(1.11, 1.13)	(1.23, 1.24)	(1.35, 1.37)	(1.46, 1.49)	(1.59, 1.62)	(1.71, 1.74)	(1.82, 1.85)	(1.96, 2)	(2.09, 2.12)	(2.19, 2.24)
CPU time for self-collection (Seconds)	1.85	2.68	3.51	4.34	5.17	5.95	6.79	7.62	8.41	9.19
SD	0.017	0.014	0.0241	0.0254	0.0406	0.0495	0.0363	0.0422	0.0394	0.0617
CI (95%)	(1.84, 1.86)	(2.67, 2.69)	(3.49, 3.52)	(4.32, 4.36)	(5.14, 5.2)	(5.91, 5.98)	(6.77, 6.82)	(7.59, 7.65)	(8.38, 8.44)	(9.15, 9.24)

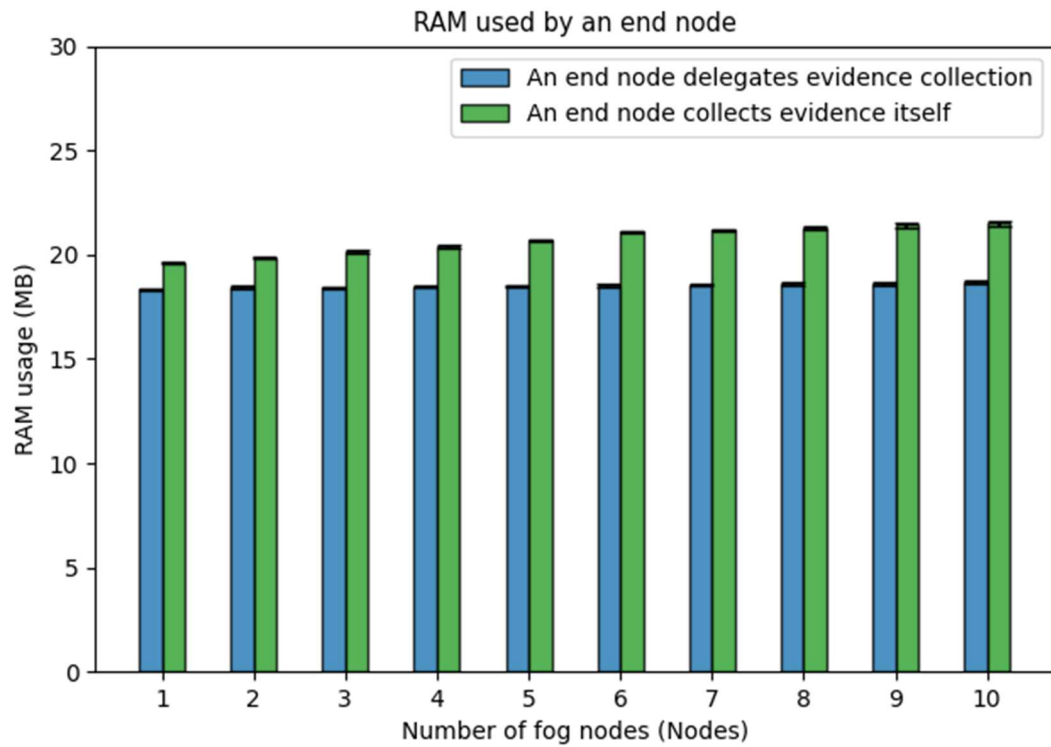


Figure 6.31: RAM used by an end node performing trust management delegation with self-evidence collection and delegated evidence collection

Table 6.28: RAM used by an end node performing trust management delegation with self-evidence collection and delegated evidence collection

No. fog nodes (Nodes)	1	2	3	4	5	6	7	8	9	10
RAM usage for delegating collection (MB)	18.3	18.4	18.4	18.4	18.5	18.5	18.6	18.6	18.6	18.7
SD	0.057	0.0688	0.094	0.0568	0.0731	0.104	0.0788	0.091	0.0882	0.0866
CI (95%)	(18.3, 18.4)	(18.4, 18.5)	(18.3, 18.5)	(18.4, 18.5)	(18.5, 18.6)	(18.4, 18.6)	(18.5, 18.6)	(18.5, 18.7)	(18.5, 18.7)	(18.6, 18.7)
RAM usage for self-collection (MB)	19.6	19.9	20.2	20.4	20.7	21.1	21.2	21.3	21.4	21.5
SD	0.0767	0.0754	0.0841	0.0964	0.0769	0.0865	0.0753	0.115	0.191	0.131
CI (95%)	(19.6, 19.7)	(19.8, 19.9)	(20.1, 20.2)	(20.3, 20.5)	(20.6, 20.8)	(21, 21.1)	(21.1, 21.2)	(21.2, 21.4)	(21.3, 21.5)	(21.4, 21.6)

Figure 6.30 and Table 6.27 present the CPU time required by the end node for trust management delegation, comparing self-evidence collection with delegating evidence collection. CPU time increases in both cases as the number of fog nodes grows, but the increase is much steeper when the end node collects evidence itself. When delegating evidence collection, the end node spends 1.12 seconds with 1 fog node and 2.21 seconds with 10 fog nodes, a 97.3% increase (or approximately 2 times higher). In contrast, the CPU time for self-evidence collection rises from 1.85 seconds at 1 fog node to 9.19 seconds at 10 fog nodes, an increase of 396% (roughly 5 times higher). The sharp increase in CPU time under self-evidence collection suggests that evidence collection becomes more expensive as the number of fog nodes increases. However, delegating evidence collection substantially limits this growth, indicating that it is a more scalable approach for resource-constrained end nodes. When the two approaches are compared, delegation reduces CPU time by approximately 39.5%, from 1.85 seconds to 1.12 seconds, for 1 fog node. A greater reduction is observed with 10 fog nodes, where CPU time decreases by approximately 76.0%, from 9.19 seconds to 2.21 seconds. This difference indicates that delegation becomes increasingly advantageous as the number of fog nodes grows.

Figure 6.31 and Table 6.28 show a RAM usage pattern. Although collecting evidence locally significantly increases CPU time, RAM consumption increases only slightly. The figure suggests that the evidence collection process does not require a substantial additional memory demand. When the end node collects evidence itself, RAM usage increases from 19.6 MB for 1 fog node to 21.5 MB for 10 fog nodes, a 9.58% increase. Similarly, when evidence collection is delegated, RAM usage rises from 18.3 MB to 18.7 MB as the number of fog nodes increases from 1 to 10, an increase by 1.8%. The difference in RAM usage between the two approaches is very small, at 1.28 MB for 1 fog node and 2.83 MB for 10 fog nodes. Evidence collection mainly increases CPU time rather than RAM usage. Although the end node must collect evidence towards more fog nodes, this evidence collection process adds only a small memory overhead.

As shown in Figure 6.32 and Table 6.29, the end node uses 20 KB of HDD space for evidence storage in both approaches. Although the end node performs evidence collection itself, it deletes the evidence from its storage immediately after sending it

to the delegate. However, it could require more HDD space to store evidence if the end node needed to buffer evidence locally and send it in batches rather than immediately. Because the results remained constant across all 10 repetitions of the experiment, there is no standard deviation or 95% confidence interval to report for this measurement.

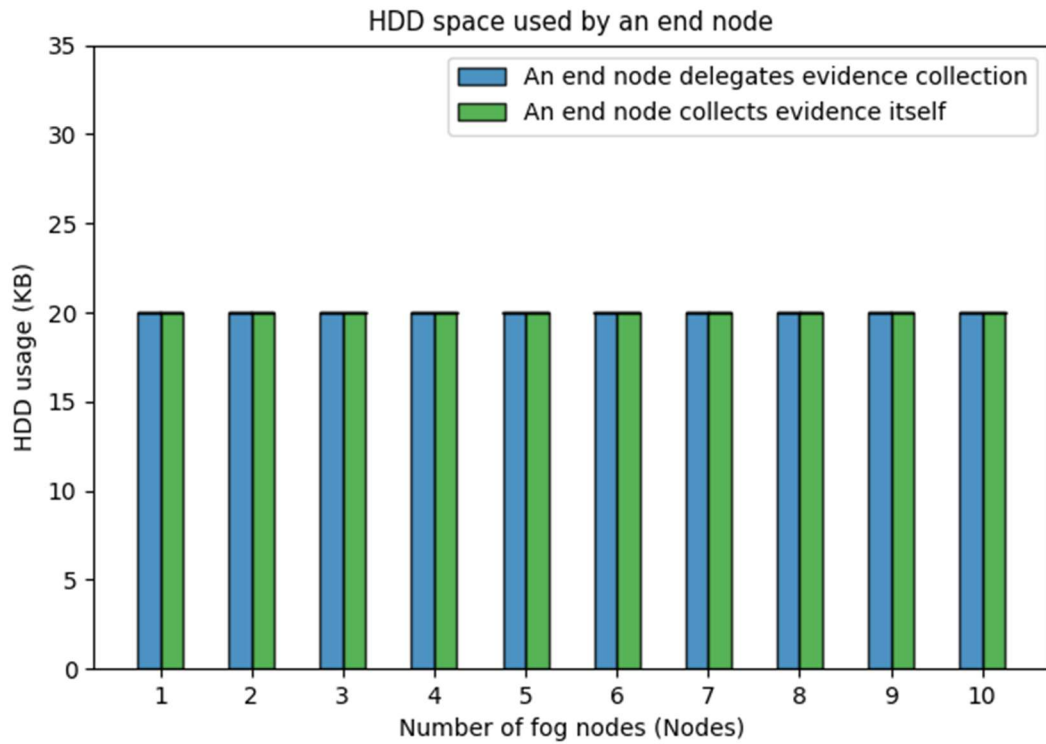


Figure 6.32: HDD used by an end node performing trust management delegation with self-evidence collection and delegated evidence collection

Table 6.29: HDD used by an end node performing trust management delegation with self-evidence collection and delegated evidence collection

No. fog nodes (Nodes)	1	2	3	4	5	6	7	8	9	10
HDD usage for delegating collection (KB)	20	20	20	20	20	20	20	20	20	20
HDD usage for self-collection (KB)	20	20	20	20	20	20	20	20	20	20

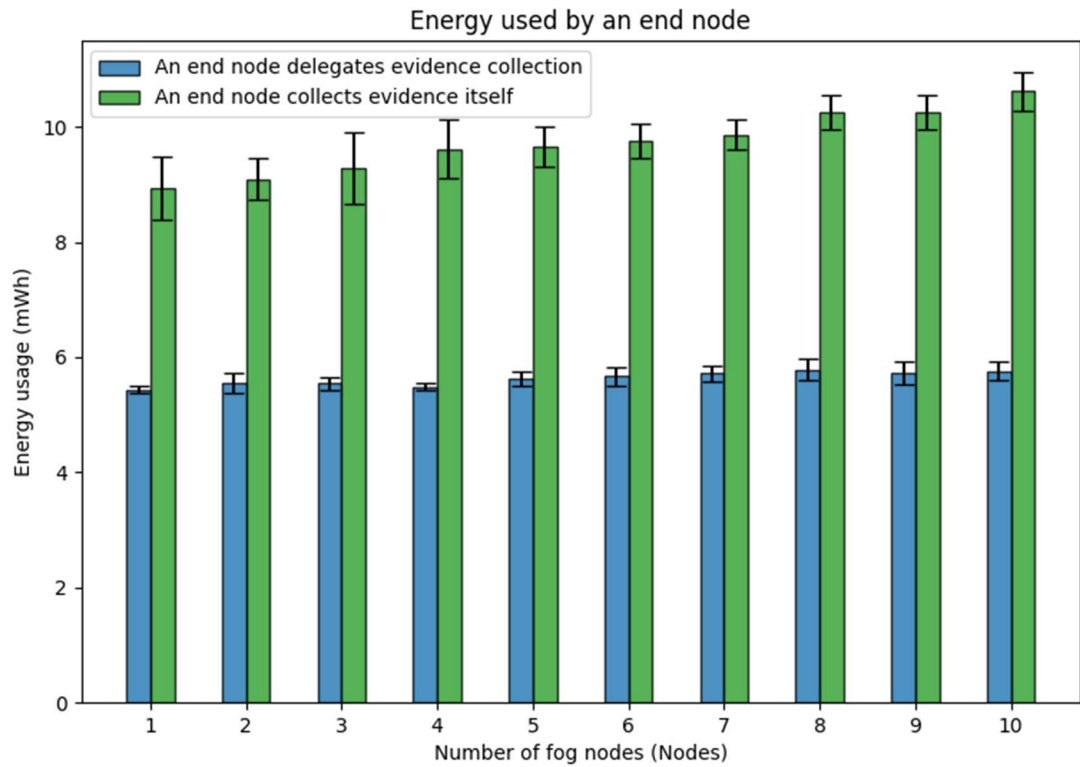


Figure 6.33: Energy used by an end node performing trust management delegation with self-evidence collection and delegated evidence collection

Table 6.30: Energy used by an end node performing trust management delegation with self-evidence collection and delegated evidence collection

No. fog nodes (Nodes)	1	2	3	4	5	6	7	8	9	10
Energy usage for delegating collection (mWh)	5.43	5.55	5.54	5.49	5.63	5.66	5.72	5.78	5.72	5.76
SD	0.0943	0.243	0.151	0.0893	0.179	0.222	0.193	0.253	0.283	0.228
CI (95%)	(5.37, 5.5)	(5.38, 5.73)	(5.43, 5.65)	(5.42, 5.55)	(5.5, 5.76)	(5.51, 5.82)	(5.58, 5.86)	(5.6, 5.97)	(5.52, 5.93)	(5.6, 5.92)
Energy usage self-collection (mWh)	8.94	9.1	9.29	9.61	9.66	9.75	9.87	10.3	10.3	10.6
SD	0.769	0.497	0.869	0.713	0.486	0.415	0.365	0.419	0.402	0.469
CI (95%)	(8.39, 9.49)	(8.74, 9.45)	(8.67, 9.92)	(9.1, 10.1)	(9.31, 10.0)	(9.45, 10.1)	(9.61, 10.1)	(9.95, 10.6)	(10.0, 10.5)	(10.3, 10.1)

Figure 6.33 and Table 6.30 show that self-evidence collection also consumes more energy than delegated evidence collection. Additionally, energy usage increases steadily for both approaches as the number of fog nodes rises. When collecting evidence for a fog node, the end node requires 64.3% more energy than when delegating evidence collection. This percentage grows to 84.3% with 10 fog nodes. Notably, the self-evidence collection scenario has a high variance in energy consumption because some evidence has a significant effect on energy consumption. For example, the response time measured for a fog node can vary significantly depending on network conditions, e.g., 10 ms and 2000 ms. When response times are longer, the end node must wait longer for a reply, resulting in higher energy consumption. Moreover, the higher energy cost is consistent with the earlier CPU time results. Increased local processing and longer evidence collection times directly increase power consumption.

Overall, the results show that self-evidence collection mainly increases CPU time and energy consumption at the resource-constrained end node, while its effect on RAM usage is relatively small. As the number of fog nodes increases in larger systems, the burden of collecting trust evidence becomes more significant for additional processing and energy requirements. However, the delegation can reduce both CPU time and energy consumption at the end node. Therefore, the findings suggest that self-evidence collection is the dominant source of overhead at the end node and that delegating evidence collection can maintain resource efficiency in large-scale fog systems.

6.6.2 Resource consumption on a delegate performing trust management with and without evidence collection on behalf of multiple end nodes

In contrast to the previous section, this section investigates resource usage on the delegate side when it performs trust management on behalf of 1 to 10 end nodes with and without evidence collection. Even if the delegate does not collect evidence, it must still receive trust evidence from the end nodes, which incurs additional communication overhead. The experimental setup is similar to that in the previous section. The number of fog nodes (trustees) is fixed at three, and the number of end nodes ranges from 1 to 10. When operating without evidence collection, the delegate receives trust evidence directly from the end nodes. Trust updates are processed using the weighted-sum trust

model upon receipt of new evidence. Subsequently, the end nodes send requests to the delegate to get trust values. All processes are run every 10 seconds.

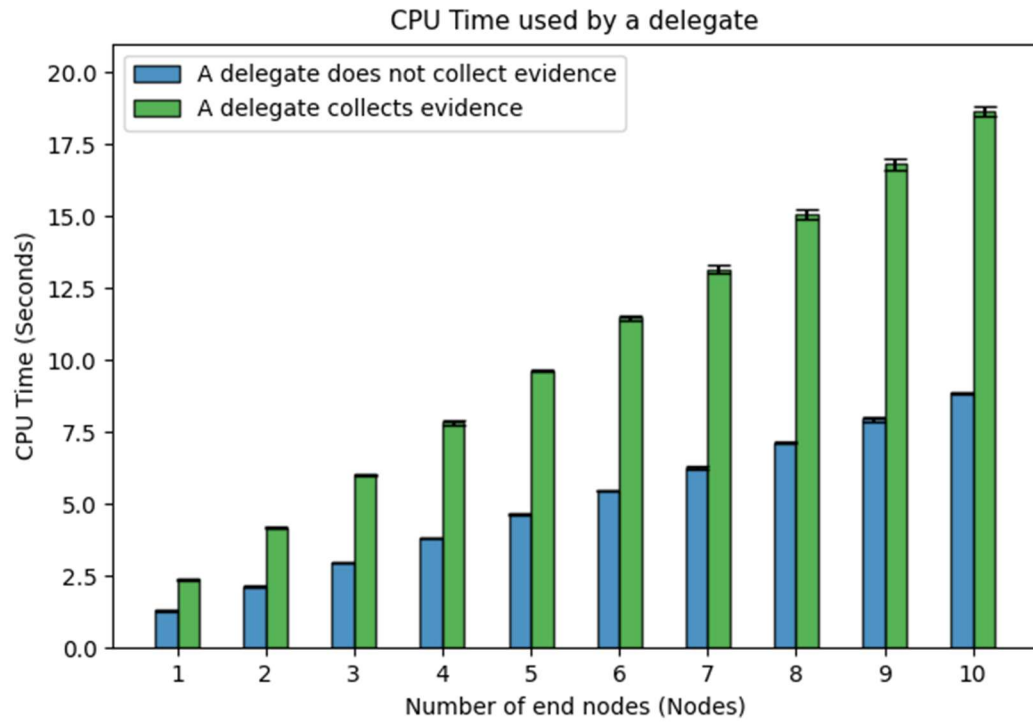


Figure 6.34: CPU time used by a delegate when performing trust on behalf of 1 to 10 delegators with and without evidence collection

Table 6.31: CPU time used by a delegate when performing trust on behalf of 1 to 10 delegators with and without evidence collection

No. end nodes (Nodes)	1	2	3	4	5	6	7	8	9	10
CPU time for without evidence collection (Seconds)	1.3	2.14	2.98	3.82	4.65	5.46	6.26	7.14	7.94	8.85
SD	0.0123	0.0238	0.0227	0.0175	0.0273	0.0347	0.102	0.0331	0.147	0.0396
CI (95%)	(1.29, 1.31)	(2.12, 2.16)	(2.96, 2.99)	(3.81, 3.83)	(4.63, 4.67)	(5.44, 5.49)	(6.19, 6.34)	(7.12, 7.16)	(7.84, 8.05)	(8.82, 8.88)
CPU time for with evidence collection (Seconds)	2.39	4.18	6.02	7.84	9.64	11.5	13.1	15.1	16.8	18.6
SD	0.0307	0.0585	0.0399	0.0998	0.0562	0.144	0.185	0.258	0.289	0.237
CI (95%)	(2.37, 2.41)	(4.14, 4.23)	(5.99, 6.05)	(7.77, 7.91)	(9.6, 9.68)	(11.4, 11.6)	(13, 13.3)	(14.9, 15.3)	(16.6, 17)	(18.5, 18.8)

Figure 6.34 and Table 6.31 demonstrate that collecting evidence requires much higher CPU time on the delegate than in scenarios where the delegate does not perform evidence collection. The delegate requires nearly twice as much CPU time to collect evidence as not to collect it. At 1 end node, evidence collection requires 1.09 seconds more CPU time, an 83.9% increase. At 5 and 10 end nodes, this difference grows to 107% (about 2.1 times) and 111% (about 2.1 times), respectively. Compared to the previous section, delegating evidence collection decreases CPU time on the end node, evidence collection still creates processing costs for the delegate. This highlights a trade-off in delegation. Offloading workload from the end node improves its efficiency, but it increases additional computational cost on the delegate. CPU usage for both approaches increases significantly as the number of end nodes grows. As the number of end nodes increases from 1 to 10, CPU time for the evidence collection scenario rises by 678%, which is about 7.8 times higher (from 2.39 seconds to 18.6 seconds). For the other scenario, CPU time increases by 581%, or approximately 6.8 times higher (from 1.3 seconds to 8.85 seconds).

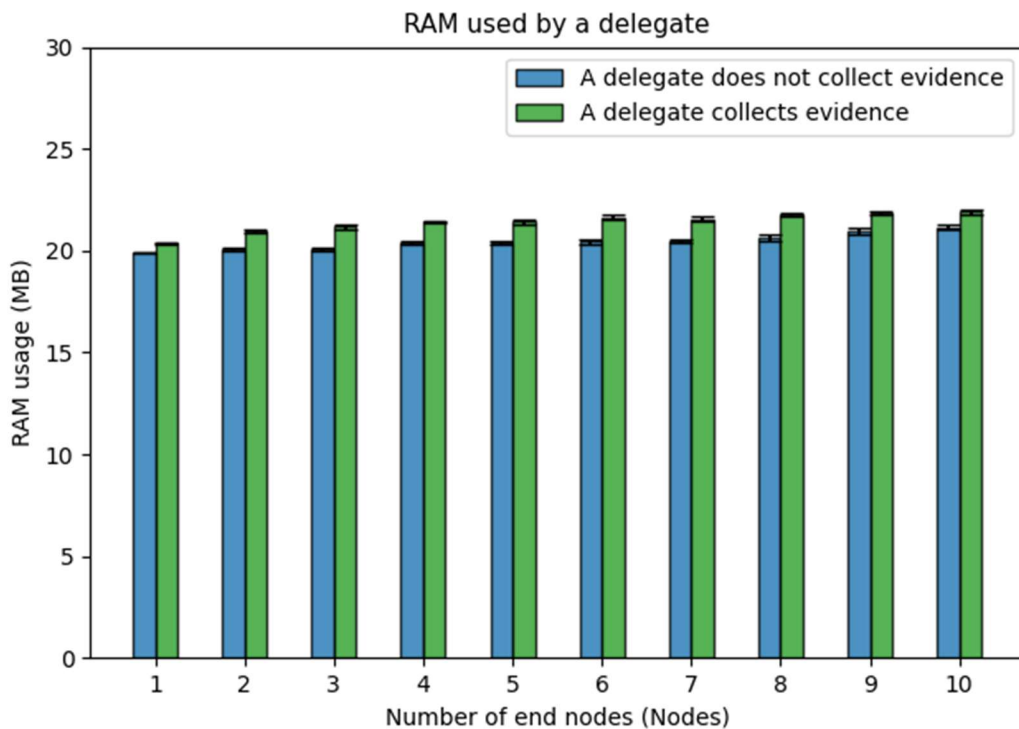


Figure 6.35: RAM used by a delegate when performing trust on behalf of 1 to 10 delegators with and without evidence collection

Table 6.32: RAM used by a delegate when performing trust on behalf of 1 to 10 delegators with and without evidence collection

No. end nodes (Nodes)	1	2	3	4	5	6	7	8	9	10
RAM usage for without evidence collection (MB)	19.9	20	20.1	20.4	20.4	20.4	20.5	20.6	21	21.1
SD	0.0474	0.134	0.106	0.0899	0.134	0.138	0.0784	0.226	0.182	0.157
CI (95%)	(19.9, 19.9)	(20, 20.1)	(20, 20.1)	(20.3, 20.5)	(20.3, 20.5)	(20.3, 20.5)	(20.4, 20.5)	(20.5, 20.8)	(20.8, 21.1)	(21, 21.3)
RAM usage for with evidence collection (MB)	20.4	21	21.2	21.4	21.4	21.6	21.6	21.8	21.9	21.9
SD	0.055	0.0747	0.145	0.111	0.183	0.134	0.147	0.115	0.134	0.17
CI (95%)	(20.3, 20.4)	(20.9, 21)	(21.1, 21.3)	(21.3, 21.5)	(21.3, 21.6)	(21.5, 21.7)	(21.5, 21.7)	(21.7, 21.8)	(21.8, 22)	(21.8, 22)

The amount of RAM used by the delegate is presented in Figure 6.35 and Table 6.32. In contrast to the sharp increase in CPU time, RAM usage rises only marginally in both scenarios. This indicates that trust management, whether with or without evidence collection, requires little additional memory as the number of end nodes or delegators increases. As the number of end nodes increases from 1 to 10, RAM usage increases by just 2.26% when collecting evidence and by 3.6% when evidence collection is not performed. In addition, the RAM requirement is slightly higher when evidence collection is included, with increases of 2.51% for 1 end node and 3.79% for 10 end nodes. This RAM result implies that evidence collection introduces only a small memory overhead, which is consistent with the RAM consumption on the end node in the previous section. The reason is that the evidence collection task mainly involves repeated processing and communication, rather than storing large amounts of data in memory. Therefore, memory capacity is not the main cost of evidence collection.

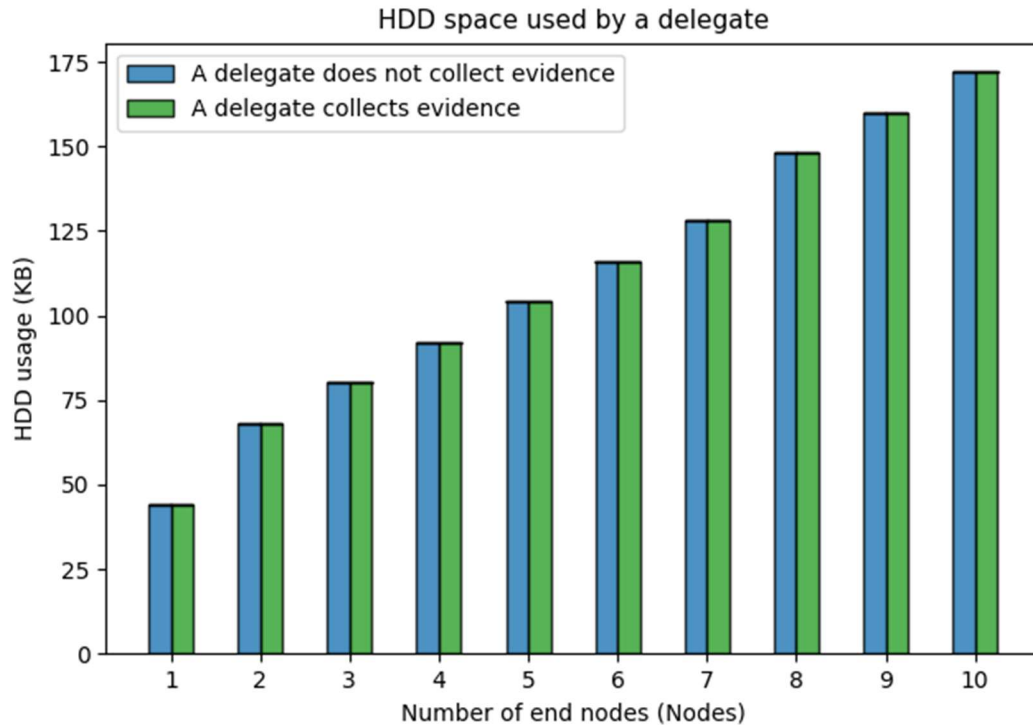


Figure 6.36: HDD used by a delegate when performing trust on behalf of 1 to 10 delegators with and without evidence collection

Table 6.33: HDD used by a delegate when performing trust on behalf of 1 to 10 delegators with and without evidence collection

No. end nodes (Nodes)	1	2	3	4	5	6	7	8	9	10
HDD usage for without evidence collection (KB)	44	68	80	92	104	116	128	148	160	172
HDD usage for with evidence collection (KB)	44	68	80	92	104	116	128	148	160	172

Both approaches require the same amount of HDD space to store trust values and evidence, as shown in Figure 6.36 and Table 6.33. HDD usage starts at 44 KB with 1 end node and increases to 172 KB with 10 end nodes. Because HDD space consumption remains constant across all experiment repetitions, standard deviations and 95% confidence intervals are not reported. The HDD results also indicate that storage demand is not the main cost for evidence collection on the delegate node in this experiment.

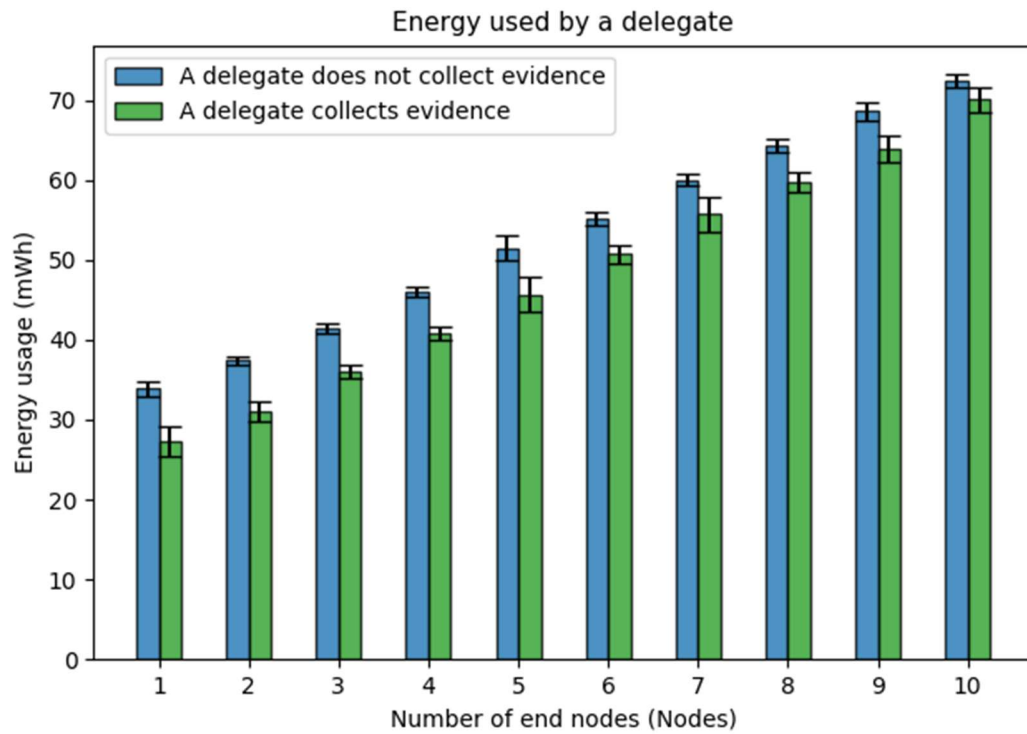


Figure 6.37: Energy used by a delegate when performing trust on behalf of 1 to 10 delegators with and without evidence collection

Table 6.34: Energy used by a delegate when performing trust on behalf of 1 to 10 delegators with and without evidence collection

No. end nodes (Nodes)	1	2	3	4	5	6	7	8	9	10
Energy for without evidence collection (mWh)	33.9	37.4	41.4	46	51.4	55.2	60	64.3	68.6	72.5
SD	1.21	0.7	0.907	0.935	2.15	1.19	1.11	1.24	1.58	1.07
CI (95%)	(33, 34.7)	(36.9, 37.9)	(40.7, 42)	(45.4, 46.7)	(49.9, 53)	(54.4, 56.1)	(59.3, 60.8)	(63.4, 65.2)	(67.5, 69.8)	(71.7, 73.2)
Energy for with evidence collection (mWh)	27.3	31	36	40.8	45.7	50.7	55.7	59.8	63.9	70.1
SD	2.66	1.81	1.18	1.23	3.02	1.55	3.07	1.76	2.35	2.13
CI (95%)	(25.4, 29.2)	(29.8, 32.3)	(35.2, 36.8)	(40, 41.7)	(43.5, 47.8)	(49.6, 51.8)	(53.5, 57.9)	(58.5, 61)	(62.3, 65.6)	(68.6, 71.6)

Figure 6.37 and Table 6.34 show an interesting result for energy consumption. The delegate consumes more energy when performing trust management without evidence collection than when collecting evidence. Although the delegate does not collect trust evidence, it still consumes energy to receive evidence from end nodes. It takes a longer time to receive the evidence from the end node than to collect it, leading to increased overall energy use. For example, with 1 end node, the delegate consumes 24.13% more energy in the scenario without evidence collection. However, this percentage difference decreases as the number of end nodes increases, 12.61% at 5 end nodes and 3.37% at 10 end nodes. Overall, the findings show that evidence collection does not significantly affect energy consumption on the delegate, even though it requires substantial CPU time, because communication with end nodes is a more important factor in energy use than evidence collection.

6.6.3 Resource consumption on an end node: Comparing local trust management using weighted sum versus fuzzy logic

Most end nodes can perform trust management using a weighted-sum trust model, but they are not well-suited to the more computationally intensive fuzzy-logic model. This experiment evaluates and compares the resource requirements of an end node when employing both weighted-sum and fuzzy-logic trust models. In the experimental setup, the end node performs local trust management towards fog nodes (Trustee) using both approaches, with the number of fog nodes varied from 1 to 10. The end node collects trust evidence for each fog node and stores it locally. Subsequently, it performs trust updates and makes trust-based decisions. These processes are executed every 10 seconds and repeated for 10 cycles to make the result value visible.

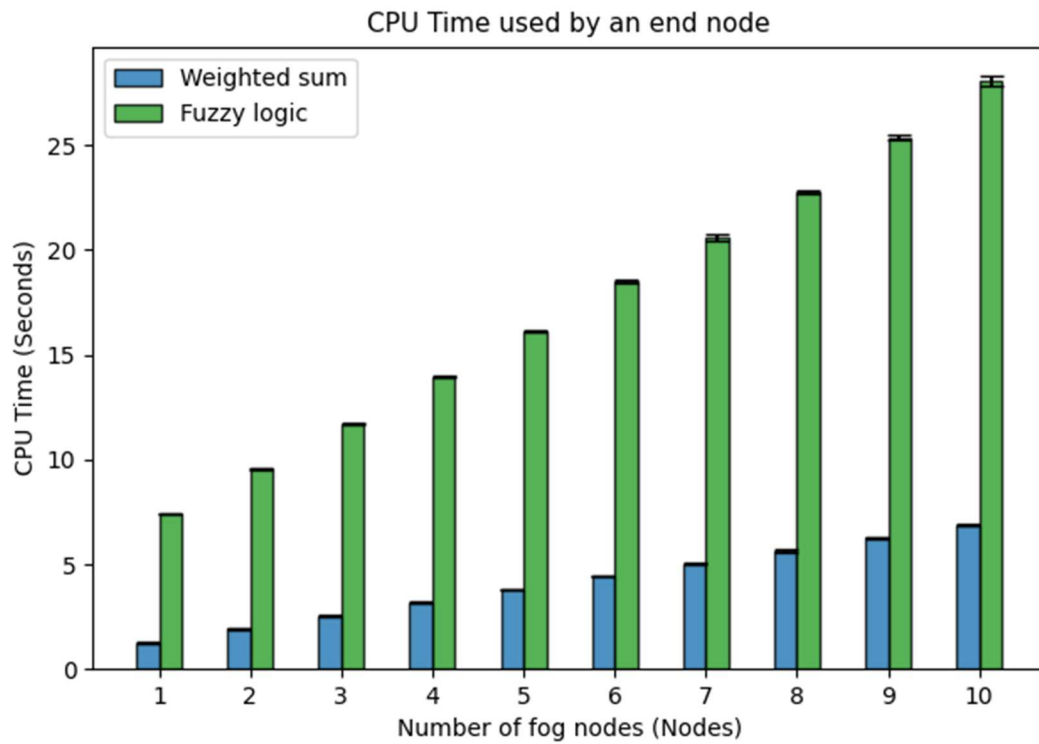


Figure 6.38: CPU time used by an end node when performing trust computation locally using weighted sum versus fuzzy logic

Table 6.35: CPU time used by an end node when performing trust computation locally using weighted sum versus fuzzy logic

No. fog nodes (Nodes)	1	2	3	4	5	6	7	8	9	10
CPU time for weighted sum (Seconds)	1.26	1.91	2.54	3.17	3.81	4.43	5.03	5.65	6.25	6.88
SD	0.0107	0.025	0.0302	0.0341	0.0367	0.0442	0.0362	0.0651	0.0583	0.0513
CI (95%)	(1.25, 1.26)	(1.89, 1.92)	(2.52, 2.56)	(3.15, 3.2)	(3.78, 3.84)	(4.4, 4.46)	(5.01, 5.06)	(5.6, 5.7)	(6.21, 6.29)	(6.84, 6.92)
CPU time for fuzzy logic (Seconds)	7.4	9.56	11.7	14	16.1	18.5	20.6	22.8	25.4	28.1
SD	0.0324	0.0757	0.0604	0.0998	0.0748	0.114	0.23	0.157	0.197	0.316
CI (95%)	(7.38, 7.43)	(9.5, 9.61)	(11.6, 11.7)	(13.9, 14)	(16.1, 16.2)	(18.4, 18.6)	(20.4, 20.8)	(22.7, 22.9)	(25.2, 25.5)	(27.9, 28.3)

Figure 6.38 and Table 6.35 compare the CPU time required by the end node to perform trust management locally using weighted-sum and fuzzy-logic trust models. CPU time increases for both trust models as the number of fog nodes grows. However, the fuzzy logic approach consumes substantially more CPU time than the weighted sum. At one fog node, the fuzzy logic model requires 7.4 seconds, and the weighted sum needs 1.26 seconds, around 5.9 times higher. As the number of fog nodes rises, the percentage of difference decreases slightly. At five fog nodes, fuzzy logic requires 16.1 seconds, while weighted sum needs 3.81 seconds, 323.95% higher, or around 4.2 times. At ten fog nodes, fuzzy logic requires 28.1 seconds, while weighted sum requires 6.88 seconds, 307% higher, or about 4.1 times. The CPU time result highlights that the fuzzy-logic approach is an intensive computational task for the end node throughout the experiment. The CPU time in this experiment is consistent with the experiment in Section 6.5.2, where the fog node requires more CPU time for fuzzy logic than the weighted-sum model. However, the difference between the two approaches in this experiment is larger when they are executed on the end node. This reason can be explained by differences in device capabilities. The end node is represented by a Raspberry Pi Zero, while the fog node is implemented on a more capable Raspberry Pi 4, which provides higher processing power for more complex trust evaluation tasks.

The RAM results shown in Figure 6.39 and Table 6.36 indicate that the fuzzy logic model requires substantially more memory than the weighted sum model. However, the memory overhead of fuzzy logic remains consistently high throughout the experiment. With 1 fog node, the fuzzy logic model uses 54.3 MB, while the weighted sum model requires only 14.9 MB, about 3.6 times more, or a 264% increase. As the number of fog nodes increases, RAM usage rises slightly for both methods. At 10 fog nodes, fuzzy logic consumes 58.8 MB compared to 15.9 MB for the weighted sum, making fuzzy logic 3.7 times higher, or 268% more. The much greater RAM and CPU time required by the fuzzy logic approach is because it uses more complex mathematical computations and the need to store fuzzy rules and control variables in memory, unlike the more lightweight calculations of the weighted sum method.

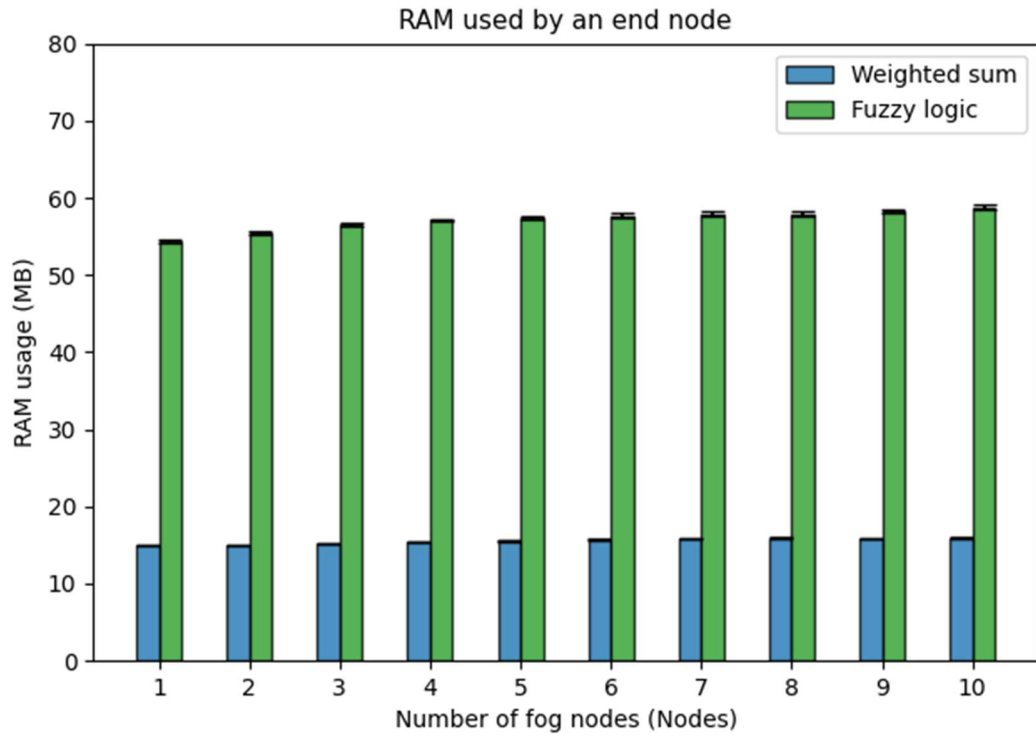


Figure 6.39: RAM used by an end node when performing trust computation locally using weighted sum versus fuzzy logic

Table 6.36: RAM used by an end node when performing trust computation locally using weighted sum versus fuzzy logic

No. fog nodes (Nodes)	1	2	3	4	5	6	7	8	9	10
RAM for weighted sum (MB)	14.9	15	15.2	15.4	15.5	15.7	15.8	15.9	15.9	15.9
SD	0.0361	0.055	0.0976	0.0663	0.113	0.0576	0.0504	0.119	0.0533	0.0609
CI (95%)	(14.9, 14.9)	(15, 15.1)	(15.1, 15.3)	(15.3, 15.4)	(15.4, 15.6)	(15.7, 15.8)	(15.8, 15.9)	(15.8, 16)	(15.8, 15.9)	(15.9, 16)
RAM for fuzzy logic (MB)	54.3	55.4	56.5	57.1	57.3	57.7	57.9	57.9	58.3	58.8
SD	0.231	0.291	0.261	0.165	0.286	0.417	0.387	0.446	0.252	0.364
CI (95%)	(54.2, 54.5)	(55.2, 55.6)	(56.3, 56.7)	(57, 57.2)	(57.1, 57.5)	(57.4, 58)	(57.6, 58.2)	(57.6, 58.2)	(58.1, 58.5)	(58.5, 59)

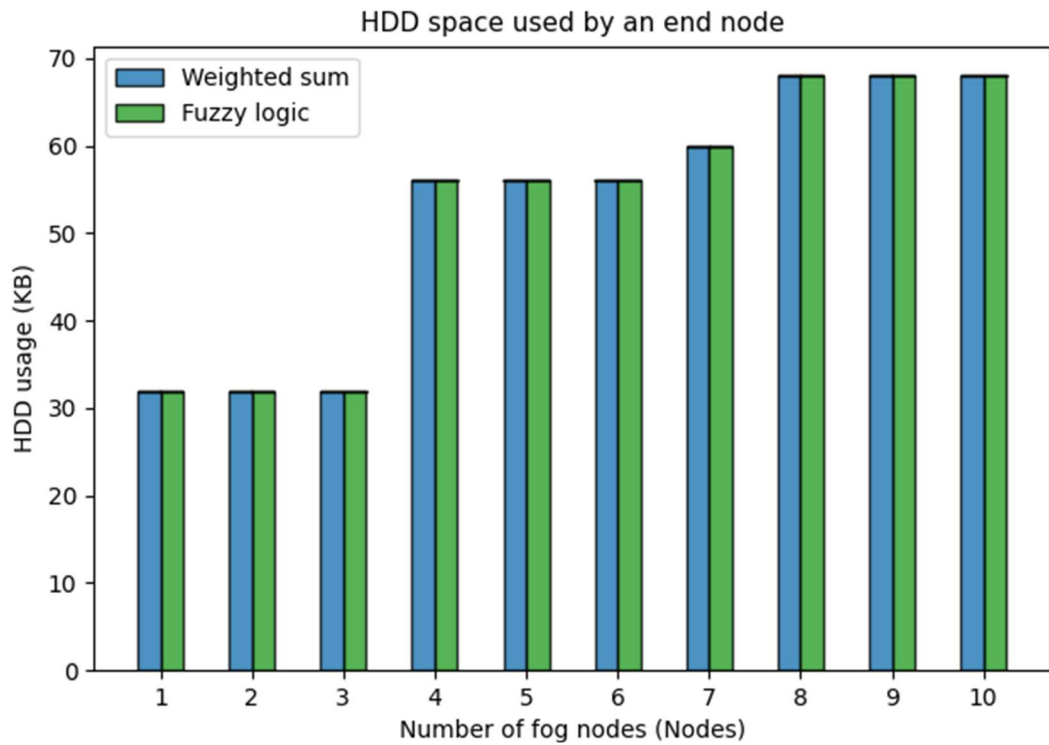


Figure 6.40: HDD used by an end node when performing trust computation locally using weighted sum versus fuzzy logic

Table 6.37: HDD used by an end node when performing trust computation locally using weighted sum versus fuzzy logic

No. fog nodes (Nodes)	1	2	3	4	5	6	7	8	9	10
HDD usage for weighted sum (KB)	32	32	32	56	56	56	56	68	68	68
HDD usage for fuzzy logic (KB)	32	32	32	56	56	56	56	68	68	68

Figure 6.40 and Table 6.37 compare the HDD usage (in KB) required to store the trust value and evidence for fuzzy-logic and weighted-sum trust models. Both approaches require the same amount of HDD, which increases with the number of fog nodes. It starts at 32 KB for 1 fog node and reaches 68 KB for 10 fog nodes. Remarkably, HDD usage remains constant within certain ranges of the number of fog nodes, such as 1, 2, and 3. This condition occurs because SQLite pre-allocates space in the database file to a specific size and extends the file size once data exceeds predetermined thresholds.

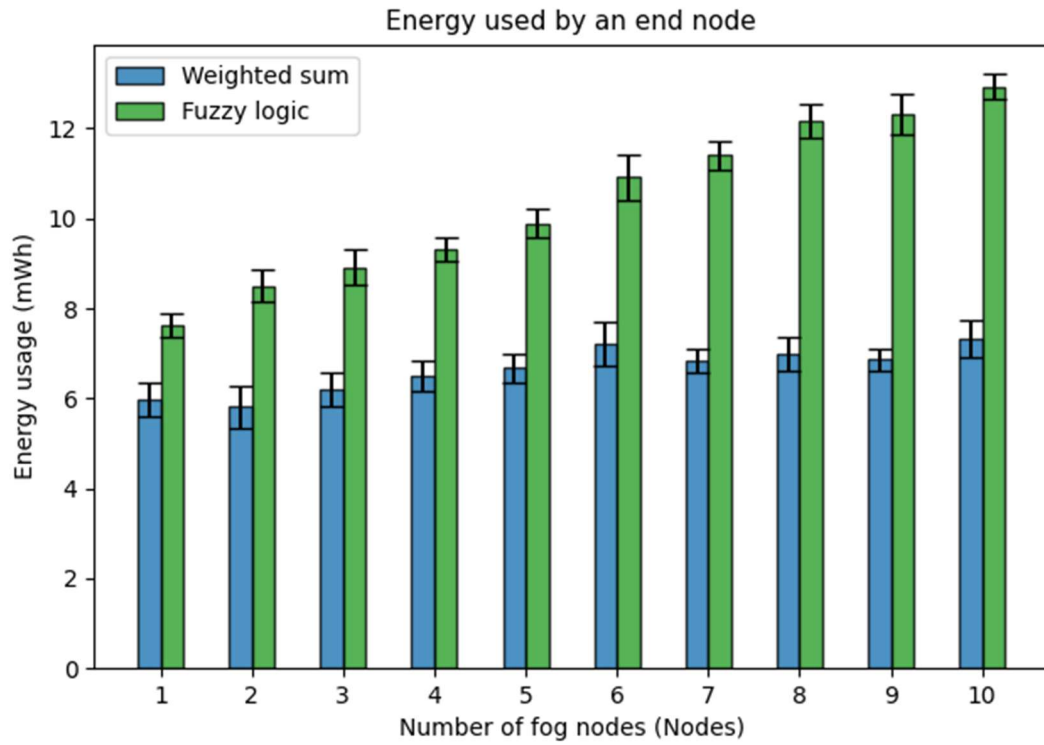


Figure 6.41: Energy used by an end node when performing trust computation locally using weighted sum versus fuzzy logic

Table 6.38: Energy used by an end node when performing trust computation locally using weighted sum versus fuzzy logic

No. fog nodes (Nodes)	1	2	3	4	5	6	7	8	9	10
Energy for weighted sum (mWh)	5.98	5.81	6.19	6.5	6.68	7.2	6.84	6.98	6.86	7.33
SD	0.517	0.65	0.514	0.447	0.449	0.685	0.362	0.52	0.351	0.564
CI (95%)	(5.61, 6.35)	(5.34, 6.27)	(5.82, 6.56)	(6.18, 6.82)	(6.36, 7)	(6.71, 7.69)	(6.58, 7.1)	(6.6, 7.35)	(6.61, 7.11)	(6.93, 7.73)
Energy for fuzzy logic (mWh)	7.63	8.5	8.92	9.33	9.89	10.9	11.4	12.2	12.3	12.9
SD	0.358	0.496	0.554	0.372	0.45	0.719	0.446	0.521	0.64	0.391
CI (95%)	(7.38, 7.89)	(8.14, 8.85)	(8.52, 9.32)	(9.06, 9.59)	(9.57, 10.2)	(10.4, 11.4)	(11.1, 11.7)	(11.8, 12.5)	(11.9, 12.8)	(12.7, 13.2)

Figure 6.41 and Table 6.38 present the total energy consumption required by the end node to perform trust management using weighted sum and fuzzy-logic trust models. Both approaches show an upward trend in energy usage as the number of fog nodes increases. However, the fuzzy-logic approach consistently consumes more energy than the weighted sum model. For one fog node, the weighted-sum model consumes 5.98 mWh, while the fuzzy-logic model requires 7.63 mWh, a 27.6% increase. As the number of fog nodes grows, this difference goes up. At 10 fog nodes, the fuzzy logic approach uses 12.9 mWh compared to 7.33 mWh for the weighted sum, a 76.4% increase. The higher energy usage correlates with the CPU time and RAM usage required by the fuzzy logic model. This suggests that battery-powered end nodes may not be suitable for executing fuzzy-logic trust evaluation locally, particularly in larger fog environments with a large number of trustee fog nodes.

6.6.4 Resource consumption analysis on a delegate when performing trust on behalf of 1 to 15 delegators using weighted sum versus fuzzy logic

This section shifts the focus from the end node, as the delegator, to the delegate side. It analyses the resource consumption of the delegate node when it performs trust management using weighted-sum and fuzzy-logic trust models on behalf of end nodes. The experimental setup is the same as in the previous section (Section 6.6.3). However, the number of fog nodes is fixed at 3, and the number of end nodes varies from 1 to 10.

Upon receiving delegation requests from each end node, the delegate performs all trust management roles on behalf of each end node towards the three fog nodes concurrently. For each run, the delegate collects and stores trust evidence, updates trust values and responds to trust queries from end nodes. All experimental processes are executed every 10 seconds for 10 times to obtain observable results.

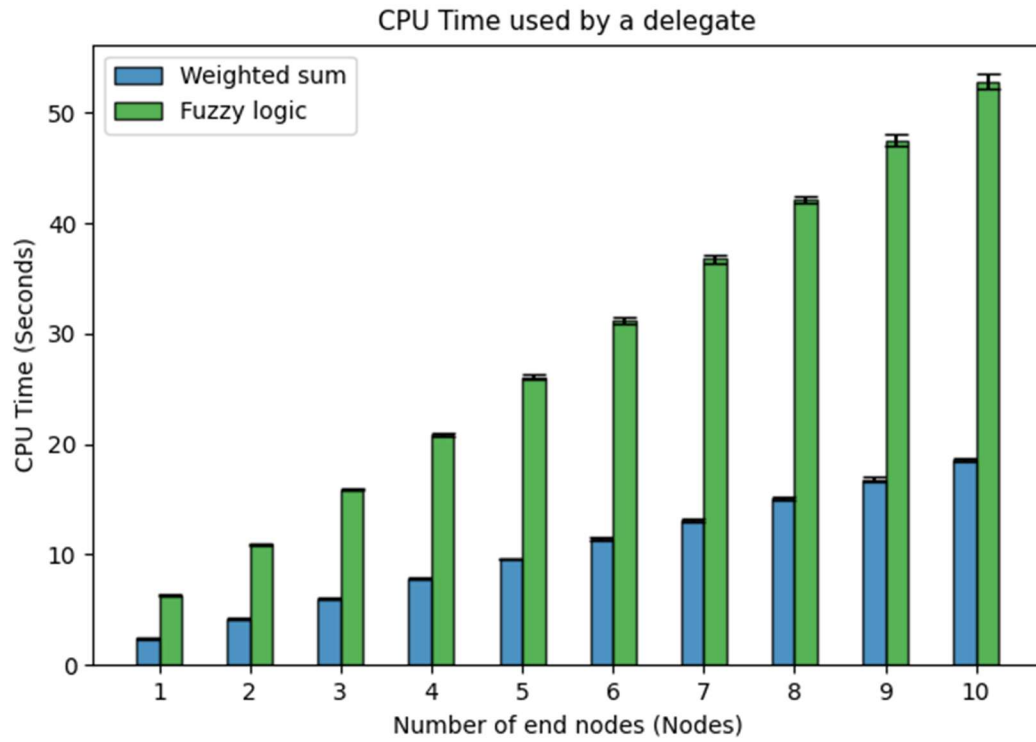


Figure 6.42: CPU time used by a delegate when performing trust on behalf of 1 to 15 delegators using weighted sum versus fuzzy logic

Table 6.39: CPU time used by a delegate when performing trust on behalf of 1 to 15 delegators using weighted sum versus fuzzy logic

No. end nodes (Nodes)	1	2	3	4	5	6	7	8	9	10
CPU time for weighted sum (Seconds)	2.39	4.18	6.02	7.84	9.64	11.5	13.1	15.1	16.8	18.6
SD	0.0307	0.0585	0.0399	0.0998	0.0562	0.144	0.185	0.258	0.289	0.237
CI (95%)	(2.37, 2.41)	(4.14, 4.23)	(5.99, 6.05)	(7.77, 7.91)	(9.6, 9.68)	(11.4, 11.6)	(13, 13.3)	(14.9, 15.3)	(16.6, 17)	(18.5, 18.8)
CPU time for fuzzy logic (Seconds)	6.33	11	15.9	20.9	26.1	31.2	36.8	42.2	47.5	52.9
SD	0.0755	0.115	0.11	0.186	0.275	0.435	0.574	0.518	0.654	0.917
CI (95%)	(6.27, 6.38)	(10.9, 11.1)	(15.8, 16)	(20.7, 21)	(25.9, 26.3)	(30.9, 31.5)	(36.4, 37.2)	(41.8, 42.5)	(47.1, 48)	(52.2, 53.5)

Figure 6.42 and Table 6.39 present the CPU time required by the delegate when performing trust management using weighted-sum and fuzzy-logic approaches. The results show that the fuzzy logic approach requires substantially more CPU time than the weighted sum. At one end node, the delegate node spends 2.39 seconds using the weighted sum model compared to 6.33 seconds with fuzzy logic, 164% more CPU time, or about 2.7 times higher. The computational demands increase as the number of end nodes rises. At 10 end nodes, the fuzzy-logic model requires 183% more CPU time than the weighted-sum approach. These results indicate that fuzzy logic remains significantly more computationally intensive on the delegate than the weighted-sum approach. The delegation protocol does not eliminate the computational overhead of fuzzy logic on the end node in Section 6.6.3, but it transfers the overhead to a more capable node. Furthermore, the delegate needs to spend resources to communicate with the end node. However, the delegate with more resources can handle this additional cost more efficiently than a constrained end node does, making the delegation approach beneficial for constrained end nodes when handling computationally intensive trust models.

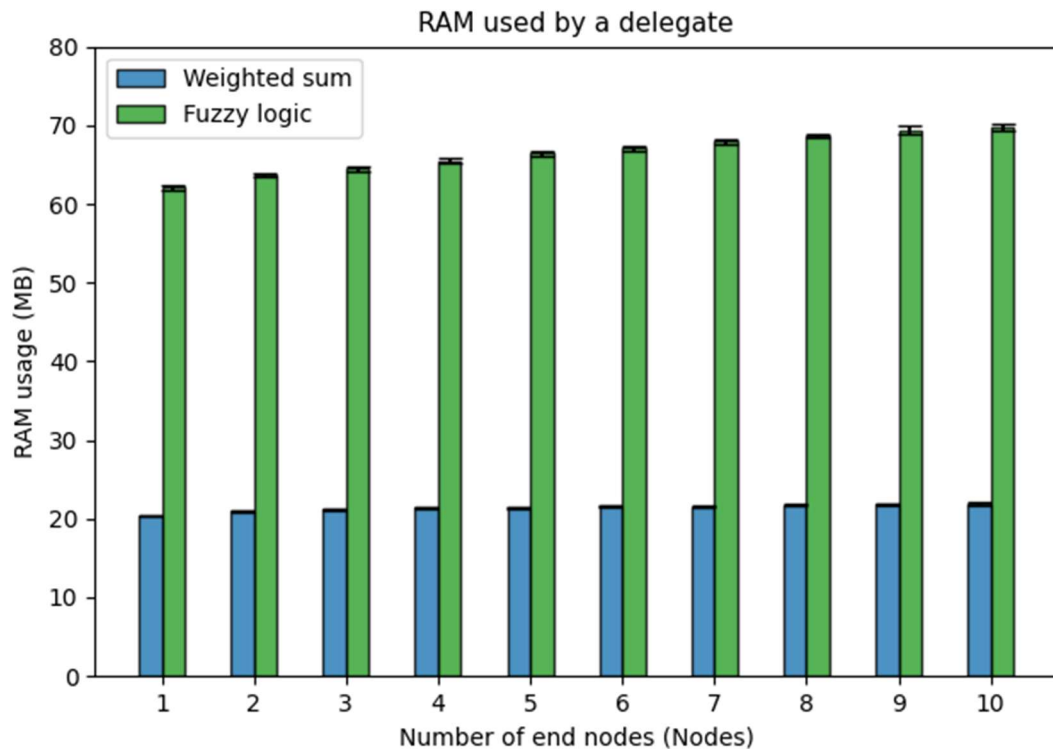


Figure 6.43: RAM used by a delegate when performing trust on behalf of 1 to 15 delegators using weighted sum versus fuzzy logic

Table 6.40: RAM used by a delegate when performing trust on behalf of 1 to 15 delegators using weighted sum versus fuzzy logic

No. end nodes (Nodes)	1	2	3	4	5	6	7	8	9	10
RAM usage for weighted sum (MB)	20.4	21	21.2	21.4	21.4	21.6	21.6	21.8	21.9	21.9
SD	0.055	0.0747	0.145	0.111	0.183	0.134	0.147	0.115	0.134	0.17
CI (95%)	(20.3, 20.4)	(20.9, 21)	(21.1, 21.3)	(21.3, 21.5)	(21.3, 21.6)	(21.5, 21.7)	(21.5, 21.7)	(21.7, 21.8)	(21.8, 22)	(21.8, 22)
RAM usage for fuzzy logic (MB)	62.1	63.6	64.5	65.5	66.4	67.1	67.9	68.7	69.4	69.7
SD	0.338	0.24	0.372	0.467	0.508	0.346	0.525	0.36	0.781	0.54
CI (95%)	(61.8, 62.3)	(63.5, 63.8)	(64.2, 64.7)	(65.1, 65.8)	(66, 66.8)	(66.8, 67.3)	(67.5, 68.3)	(68.5, 69)	(68.9, 70)	(69.3, 70.1)

As shown in Figure 6.43 and Table 6.40, the fuzzy-logic approach also requires substantially more RAM than the weighted sum method on the delegate. For 1 end node, the fuzzy-logic approach uses 204% more RAM, and this difference increases to 218% when the number of end nodes rises to 10 nodes. These results indicate that the higher memory overhead of fuzzy logic is not limited to the end node in Section 6.6.3, but also remains significant at the delegate. Moreover, both methods show an increase in RAM usage as the number of end nodes increases. However, the increase remains much smaller than the rise in CPU time. As the number of end nodes increases from 1 to 10, RAM usage rises by 7.56% for the weighted sum and by 12.3% for fuzzy logic. This indicates that the increase in the number of end nodes is not the main effect of RAM consumption on the delegate in the fuzzy-logic model.

HDD usage is the same for both trust models, as depicted in Figure 6.44 and Table 6.41. The HDD usage requires 44 KB for one end node and grows to 172 KB for ten end nodes. Because the amount of HDD space remains the same for every experimental repetition, the standard deviation and 95% confidence intervals are not reported.

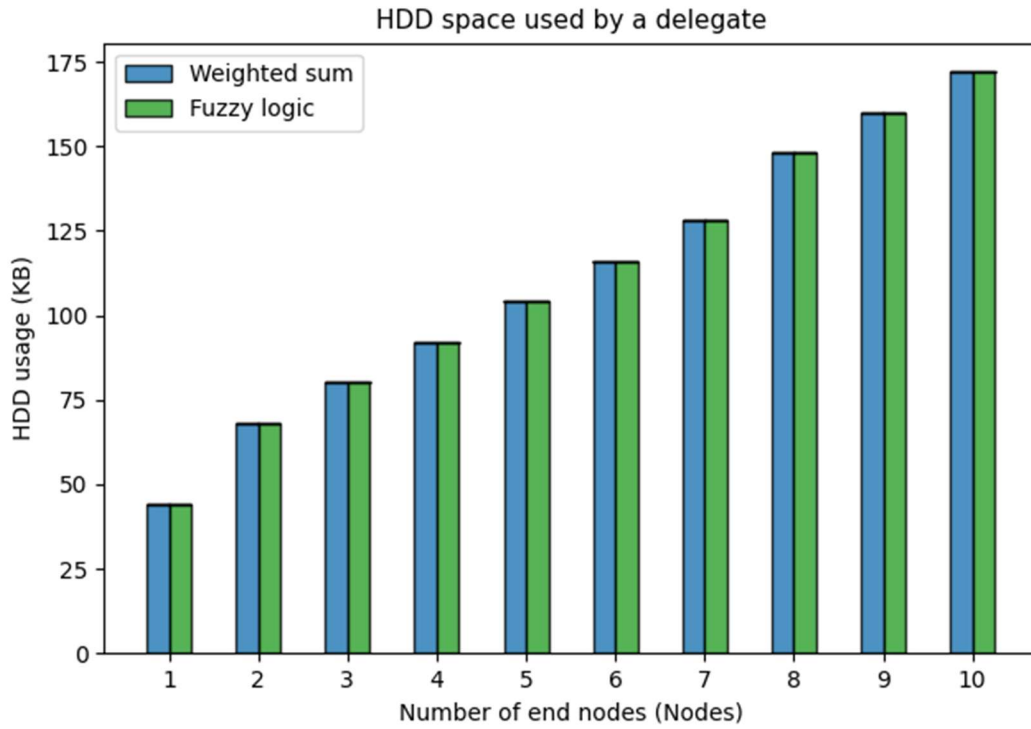


Figure 6.44: HDD used by a delegate when performing trust on behalf of 1 to 15 delegators using weighted sum versus fuzzy logic

Table 6.41: HDD used by a delegate when performing trust on behalf of 1 to 15 delegators using weighted sum versus fuzzy logic

No. end nodes (Nodes)	1	2	3	4	5	6	7	8	9	10
HDD usage for weighted sum (KB)	44	68	80	92	104	116	128	148	160	172
HDD usage for fuzzy logic (KB)	44	68	80	92	104	116	128	148	160	172

Figure 6.45 and Table 6.42 illustrate the energy consumption of the delegate. The fuzzy logic approach consistently consumes more energy than the weighted sum. At one end node, the weighted-sum approach uses 27.3 mWh, while the fuzzy-logic approach requires 28.5 mWh, a 4.36% difference. At ten end nodes, the difference grows to 21.7%, 85.3 mWh for fuzzy logic and 70.1 mWh for weighted sum. However, the energy difference between the two approaches is smaller than that observed at the end node in Section 6.6.3. This suggests that the delegate, implemented on a Raspberry Pi 4, can perform the fuzzy-logic trust model more efficiently in terms of energy than the end node. Overall, the results in this section suggest that delegate hardware

intended to support computationally intensive trust models should be provisioned with high processing power, memory and energy capacity in real-world deployment.

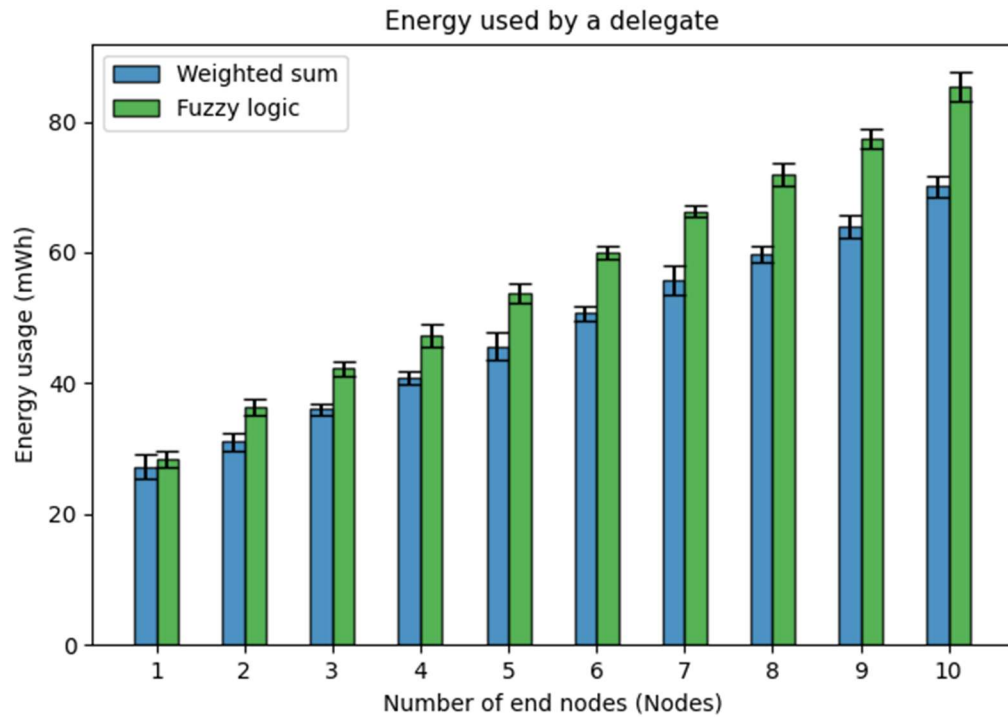


Figure 6.45: Energy used by a delegate when performing trust on behalf of 1 to 15 delegators using weighted sum versus fuzzy logic

Table 6.42: Energy used by a delegate when performing trust on behalf of 1 to 15 delegators using weighted sum versus fuzzy logic

No. end nodes (Nodes)	1	2	3	4	5	6	7	8	9	10
Energy usage for weighted sum (mWh)	27.3	31	36	40.8	45.7	50.7	55.7	59.8	63.9	70.1
SD	2.66	1.81	1.18	1.23	3.02	1.55	3.07	1.76	2.35	2.13
CI (95%)	(25.4, 29.2)	(29.8, 32.3)	(35.2, 36.8)	(40, 41.7)	(43.5, 47.8)	(49.6, 51.8)	(53.5, 57.9)	(58.5, 61)	(62.3, 65.6)	(68.6, 71.6)
Energy usage for fuzzy logic (mWh)	28.5	36.4	42.3	47.3	53.8	60	66.3	71.9	77.4	85.3
SD	1.78	1.73	1.59	2.36	1.96	1.55	1.36	2.58	2.23	3.14
CI (95%)	(27.2, 29.7)	(35.1, 37.6)	(41.2, 43.5)	(45.6, 49)	(52.4, 55.2)	(58.9, 61.1)	(65.4, 67.3)	(70.1, 73.8)	(75.8, 79)	(83.1, 87.6)

6.7 Comparison with traditional trust management systems and task-offloading schemes

This section compares the trust management delegation protocol with traditional trust management systems and task-offloading schemes. Both similarities and differences are discussed.

6.7.1 Comparison with trust management systems

The proposed protocol architecture differs from traditional trust management systems in IoT and cloud environments in several key respects, as shown in Table 6.43. Traditional trust management systems in the cloud [74] and IoT [27] environments usually focus on evaluating node behaviour and detecting malicious devices without considering the resource limitations. The proposed protocol considers the differences in device capabilities and enables resource-constrained devices to delegate trust management functions that they cannot support. The functions to be delegated are also determined based on the capabilities of the available delegate nodes, rather than assuming uniform resource availability across all devices. Moreover, by using nearby nodes, the protocol is particularly well-suited for fog environments where low latency is a primary requirement.

The proposed protocol provides seamless, low-latency mobility support through delegation handovers. This differs from cloud-based trust management approaches, which rely on a centralised remote cloud server for all trust-related tasks [74]. In contrast, mobility is not a concern for IoT-based trust management approaches because devices in IoT environments typically perform trust management locally and do not require handover support [27].

The proposed protocol uses CoAP, a lightweight protocol specifically designed for constrained IoT devices. This is a difference from cloud-based trust management, which does not consider lightweight communication protocols [77]. However, IoT systems already tend to use lightweight protocols that are well-suited to constrained environments [35].

The role-based access control model of the proposed protocol ensures that entities are only granted access to trust data that correspond to their assigned delegation roles. Although trust management in cloud environments supports access control mechanisms. These approaches are typically managed centrally by the cloud server rather than by the individual devices [75]. On the other hand, IoT-based trust management approaches do not consider access control over trust data because these approaches store trust data locally [35].

Table 6.43: Comparison of the trust delegation protocol against other trust management systems

Aspect	IoT trust management	Cloud-based trust management	Proposed protocol
Evaluation location	Local at each device	Centralised at cloud servers	Local/delegated to nearby nodes
Mobility support	Not required	No handover mechanism	Support delegation handover
Communication protocol	Lightweight protocols	No consideration of lightweight protocols	Lightweight CoAP
Access control over trust data	No consideration of trust data restriction	Managed by the cloud	Role-based access control

6.7.2 Comparison with task-offloading schemes

Table 6.44 presents a comparison of the proposed delegation protocol with different task-offloading schemes in fog computing. The proposed protocol is similar to these schemes in that it enables resource-constrained devices to leverage the computational resources available in their environment to perform tasks they cannot efficiently handle locally. However, existing task-offloading schemes focus on transferring generic application tasks, rather than accounting for the specific requirements of individual tasks. However, differences among task types may require specific device capabilities to execute them effectively. The proposed trust management delegation mechanism focuses on trust management tasks and allows decomposing them into four roles, each of which can be offloaded to an appropriate delegate node.

Most prior task-offloading studies have not considered lightweight communication. However, messages exchanged between delegators and delegates in the proposed protocol are designed to minimise communication overhead and memory usage by using COAP and JSON. Moreover, it supports mobility in dynamic environments, where trust management task delegation can be transferred to another node.

Table 6.44: Comparative analysis of different task offloading algorithms

Reference	Leverages available resources	What is offloaded	Task decomposition	Lightweight message	Mobility
Shahryari et al. [81]	✓	Delay and computation-intensive tasks	✗	✗	✗
Singh and Singh [82]	✓	Computation-intensive tasks	✗	✗	✗
Yadav and Sangwan [84]	✓	Computation tasks	✗	✗	✗
Alenizi and Rana [85]	✓	Generic tasks	✗	✗	✗
Vakilian et al. [86]	✓	Computation tasks	✗	✗	✗
Aleteemat and Shurman [83]	✓	Computation-intensive tasks	✗	✗	✗
Proposed mechanism	✓	Trust management tasks	Four delegation roles	CoAP with JSON	✓

6.8 Discussion

This section gathers the experimental findings to address the three experimental evaluation questions introduced at the beginning of this chapter.

EQ1: Does the trust management delegation protocol reduce CPU time, memory usage, storage space, and energy consumption on resource-constrained devices when delegating trust management compared to performing local trust management?

Discussion: The experiments demonstrate that the delegation protocol substantially reduces resource consumption on constrained devices, even as the system scale grows.

CPU time is reduced by 39–84% compared with local trust computation. For instance, in Section 6.5.1, Delegation reduces the CPU time used by an end node from 7.18 seconds to 1.14 seconds, representing an 84% reduction. Similarly, in Section 6.6.1, performing trust management delegation with self-evidence collection requires 1.85 seconds of CPU time for one trustee, whereas delegating evidence collection requires 1.12 seconds, a 39% decrease. RAM usage also decreases significantly with delegation. In Section 6.5.1, RAM consumption on an end node drops from 55.7 MB for local management using the fuzzy-logic model to 19.8 MB when trust management is delegated, which is approximately 2.8 times lower. In Section 6.6.1, RAM usage for self-evidence collection decreases from 21.5 MB to 18.7 MB when evidence collection is delegated.

HDD usage is minimised with delegation. In Section 6.4.1, HDD usage for local trust management increases sharply from 0.03 MB to 0.93 MB, but it remains consistently low at 0.01 MB in delegation as more evidence is collected. Energy consumption is significantly reduced. In Section 6.6.1, delegating evidence collection uses 64.3% less energy for one trustee compared to self-evidence collection. This energy savings increases to 84.3% for ten trustees.

EQ2: What is the impact of different trust computation models (weighted sum versus fuzzy logic) on the delegation protocol's performance and resource overhead?

Discussion: The results indicate that the delegation protocol enables constrained devices to leverage sophisticated trust models that would be computationally intensive tasks. This allows for the selection of alternative trust models based on application requirements rather than being constrained by the computational limitations of end devices. The results of all experiments indicate that the fuzzy logic model consumes significantly more resources than the weighted sum approaches. However, the results also indicate that resource usage is significantly reduced through delegation. For example, in Section 6.5.1, the delegation of fuzzy-logic trust management reduces CPU time from 7.18 seconds to 1.14 seconds, a reduction of approximately 6.3 times. RAM usage is reduced from 55.7 MB to 19.8 MB, a 2.8-fold decrease. Additionally, energy consumption drops from 2.88 mWh to 1.15 mWh.

EQ3: What is the resource overhead on delegate nodes when performing trust management on behalf of multiple constrained devices?

Discussion: Although the delegation protocol reduces resource consumption on resource-constrained devices, it shifts the workload to delegate nodes. As a result, there is increased overhead on delegate nodes, especially as the number of delegators rises. For example, in Section 6.4.2, CPU time of the delegate increases sharply from 33.6 seconds at one delegator to 871 seconds at 15 delegators, a 26-fold rise. In Section 6.6.2, CPU time grows from 2.39 seconds to 18.6 seconds across the same device range. However, the experimental evidence in some settings demonstrates that the overhead patterns are predictable. RAM usage shows a steady increase, while storage consumption grows linearly. For instance, in Section 6.6.4, RAM usage climbs slightly from 20.4 MB to 21.90 MB on delegate nodes. HDD usage also increases systematically from 0.38 MB to 5.02 MB.

6.9 Conclusion

The chapter presented the experimental evaluation of the proof-of-concept trust management delegation protocol implementation. The results clearly show that performing trust management locally imposes a substantial impact on resource consumption on resource-constrained devices. The resource demands increase markedly as the number of entities grows. However, the evaluation demonstrates that the proposed trust management delegation approach effectively addresses these limitations by offloading intensive trust management tasks from resource-constrained end devices to more capable delegate nodes. This delegation approach can reduce CPU time, RAM usage, storage and energy requirements on resource-limited end devices, while maintaining efficient resource consumption on delegates even as system scale increases.

Chapter 7: Conclusion and future work

This chapter presents a summary of the research by providing a concise explanation of how the research questions were addressed, the thesis contributions, the research limitations, future research directions, and the thesis conclusions.

7.1 Research questions

- **RQ1:** What are trust management requirements for managing trust between devices participating in fog computing systems?

The key trust management requirements for managing trust between devices participating in fog computing systems have been analysed in Chapter 3. The trust management architecture must accommodate all device types in the system and consider varying capabilities, hardware configurations, and operating systems. Moreover, the architecture should utilise powerful fog nodes to efficiently manage trust management tasks. Furthermore, the architecture must be designed to maintain trust between every pair of interacting entities with a wide range of trust dimensions. Furthermore, the trust architecture must support dynamic fog environments with mobility support.

- **RQ2:** How can we design an architecture for trust management in fog computing that addresses the identified requirements?

The design of the trust management delegation architecture, as described in Chapter 4, addresses the requirements identified for heterogeneous and resource-constrained fog computing environments. This architecture design considers the diverse capabilities and resource limitations of participating devices. It introduces four main delegation roles, including evidence collector, trust computer, trust decision maker, and trust repositor. These roles enable entities to delegate trust management responsibilities according to their individual capabilities. The architecture is flexible to support a wide range of trust aspects and to maintain various trust relationships among all interacting entities.

Furthermore, the architecture includes a handover mechanism that allows devices to transfer trust management delegation from previous delegates to new nearby delegates without additional overhead on the constrained delegator. In addition, access control mechanisms are implemented to restrict entities from reading, inserting, or deleting trust data and evidence within the delegation protocol.

- **RQ3:** How can we show that the architecture is effective in supporting the different trust management scenarios?
 - How can we provide a proof-of-concept implementation?

The proof-of-concept delegation protocol has been implemented in Python to support various device types, as explained in Chapter 5. The messages exchanged between entities for the delegation are transmitted over the CoAP protocol, implemented by the aiocoap library. SQLite was used as the database engine for storing trust values and evidence.

- How can we show that the implementation accommodates participating devices to carry out their trust management?

In Chapter 6, the experimental testbed is implemented using both physical hardware and virtual machines. The Raspberry Pi Zero is used as a resource-constrained end node, while the Raspberry Pi 4 acts as a more capable fog or delegate node. To simulate larger-scale scenarios, we utilise Raspberry Pi Desktop instances run on Oracle VirtualBox. Performance metrics, including CPU time, RAM, and HDD usage, are measured with dedicated Python scripts. Energy consumption is measured using an INA219 sensor attached to the Raspberry Pi Zero. The results are recorded in files and analysed using Python, with the Matplotlib library used to generate graphs.

The experimental results demonstrate that the trust management delegation protocol accommodates devices with diverse capabilities and significantly reduces the CPU time, RAM usage, HDD space, and energy consumption required to perform trust management. This shows that the trust management delegation protocol enables resource-constrained devices to effectively participate in trust management.

7.2 Contributions

This thesis makes several contributions to research on trust management in fog computing. First, the research identifies the main challenges affecting effective trust management in fog computing systems. Fog computing consists of a diverse range of devices, from high-end servers to resource-constrained IoT devices. Resource-limited devices often lack sufficient resources to perform trust evaluations, especially computationally intensive trust models. Moreover, most existing trust management architectures [33], [34] are not flexible enough to support leveraging available resources from other nodes in the system for handling trust management tasks.

Second, this research identifies a list of trust management requirements that reflects the varying capabilities of participating devices. Previous studies have examined general requirements for trust management in fog and IoT systems, such as security, scalability, and efficiency [62], [67]. However, they do not analyse these requirements in the context of real-world fog computing scenarios. In contrast, this work derives trust management requirements through an analysis of practical fog computing application scenarios. The study also emphasises the importance of utilising available resources within the fog infrastructure, such as fog nodes, to assist resource-constrained devices in handling trust management tasks.

Third, this research introduces a trust management delegation protocol that enables devices with limited resources to delegate trust management tasks to more capable nodes within the system. In addition, a proof-of-concept implementation is provided and evaluated using real devices across multiple scenarios. Experimental results demonstrate that the proposed delegation protocol effectively accommodates different devices and reduces resource consumption on constrained nodes.

7.3 Limitations and future work

The proposed trust management delegation protocol accommodates different devices for trust management in fog systems. However, several limitations remain in the current design and implementation. First, although the delegation protocol supports device mobility, more sophisticated mechanisms and predictive mobility management are required to analyse mobility patterns and enable predictive handovers in highly

dynamic environments. Second, the protocol assumes that its main components, such as the TSS and delegates, are trustworthy. In practice, they may be compromised or behave maliciously. Further work should investigate mechanisms to assess the trustworthiness of delegates within the protocol. Moreover, future research should identify the key attacks and threat models relevant to trust management delegation, such as impersonation and replay attacks, and apply formal threat modelling methods to evaluate the protocol against these threats and identify potential vulnerabilities in its design and implementation. Third, the data exchanged between devices during trust management delegation is not encrypted. Lightweight cryptographic protocols should be employed to secure communications and message exchange between entities. Finally, the use of IP addresses as device identifiers should be replaced with more secure mechanisms, such as digital certificates, to protect against identity spoofing and impersonation attacks. In particular, post-quantum cryptography (PQC) could be applied to support secure identity management and secure communication in trust management. However, the trade-offs between security and additional resource requirements need to be evaluated carefully.

Future research directions could also explore new areas to extend trust management beyond its current scope. One direction is to study how a trust management task can be split across multiple delegate nodes, for example, separating an intensive trust computation across different delegates. This could lead to further research on load balancing, fault tolerance, and coordination among fog computing systems, particularly when one node fails or becomes overloaded. Another future direction is the integration of blockchain technology to maintain trust evidence and trust values in a tamper-resistant and immutable manner. Such an approach could improve the accountability and auditability of trust data and help protect it against unauthorised modification. Furthermore, the emergence of quantum computing presents new opportunities for trust management. Quantum computation could be used to process complex trust models more efficiently, enhancing the overall efficiency of trust management systems in large-scale environments.

7.4 Thesis conclusion

Prior research on trust management in fog computing, cloud, and the Internet of Things has primarily focused on proposing trust evaluation models for particular trust relationships, and they often assume a single trust architecture across all devices [34], [77]. This thesis addresses trust management from the perspective of heterogeneous device capabilities and the need to delegate trust management tasks from constrained devices to capable nodes. The study investigates the differences in device type in terms of resources and capabilities across fog computing application scenarios, including smart homes, smart e-health, and smart traffic management. This was followed by an analysis of trust relationships and trust aspects between the devices participating in these scenarios. The investigation revealed that devices vary considerably in computational power, memory capacity, and communication protocols. Thus, it was necessary to establish a trust management mechanism that can accommodate the variations in device types and capabilities.

The research then presents a design for a trust management delegation protocol that enables the delegation of trust management tasks from resource-constrained devices to higher-resource nodes in fog environments. Existing task-offloading schemes focus on general computational workloads [84] and do not address the specific requirements of trust management tasks. The proposed delegation mechanism is specifically designed for trust management to enhance the flexibility of resource-constrained devices to participate in trust management. In addition, the protocol incorporates trust data access control and mobility management to support delegation handover in dynamic scenarios, such as smart traffic management, which existing trust management approaches in the fog computing literature have not addressed.

A proof-of-concept implementation was evaluated from multiple perspectives. The evaluation distinguishes this work from prior approaches by demonstrating that the proposed protocol is not only conceptually different but also practically effective in addressing the resource limitations that most existing approaches leave unresolved. The overall results show that the trust management delegation protocol reduced resource consumption in terms of CPU time, RAM usage, HDD usage, and energy consumption when compared to performing trust management locally.

Bibliography

- [1] M. Aazam, S. Zeadally, and K. A. Harras, “Offloading in fog computing for IoT: Review, enabling technologies, and research opportunities,” *Future Generation Computer Systems*, vol. 87, pp. 278–289, 2018, doi: 10.1016/j.future.2018.04.057.
- [2] S. Yi, C. Li, and Q. Li, “A survey of fog computing: Concepts, applications and issues,” in *Proceedings of the 2015 Workshop on Mobile Big Data (Mobidata '15)*, pp. 37–42, 2015, doi: 10.1145/2757384.2757397.
- [3] M. Debe, K. Salah, M. H. Rehman, and D. Svetinovic, “Towards a blockchain-based decentralized reputation system for public fog nodes,” in *2019 IEEE/ACS 16th International Conference on Computer Systems and Applications (AICCSA)*, pp. 1–6, 2019, doi: 10.1109/AICCSA47632.2019.9035252.
- [4] W. Ben Daoud, M. S. Obaidat, A. Meddeb-Makhlouf, F. Zarai, and K. F. Hsiao, “TACRM: trust access control and resource management mechanism in fog computing,” *Human-centric Computing and Information Sciences*, vol. 9, no. 1, 2019, doi: 10.1186/s13673-019-0188-3.
- [5] T. Wang *et al.*, “Fog-based evaluation approach for trustworthy communication in sensor-cloud system,” *IEEE Communications Letters*, vol. 21, no. 11, pp. 2532–2535, 2017, doi: 10.1109/LCOMM.2017.2740279.
- [6] A. Thida and T. Shwe, “Process provenance-based trust management in collaborative fog environment,” in *2020 IEEE Conference on Computer Applications (ICCA)*, pp. 1–5, 2020, doi: 10.1109/icca49400.2020.9022846.
- [7] R. Mahmud, R. Kotagiri, and R. Buyya, “Fog computing: A taxonomy, survey and future directions,” in *Internet of Everything: Algorithms, Methodologies, Technologies and Perspectives*, pp. 1–28, 2018, doi: 10.1007/978-981-10-5861-5_5.
- [8] T. Grandison and M. Sloman, “A survey of trust in Internet applications,” *IEEE Communications Surveys & Tutorials*, vol. 3, no. 4, pp. 2–16, Fourth Quarter 2000, doi: 10.1109/COMST.2000.5340804.

- [9] F. Azzedin and M. Ghaleb, “Internet-of-Things and information fusion: Trust perspective survey,” *Sensors*, vol. 19, no. 8, 2019, doi: 10.3390/s19081929.
- [10] L. J. Hoffman, K. Lawson-Jenkins, and J. Blum, “Trust beyond security: An expanded trust model,” *Communications of the ACM*, vol. 49, no. 7, pp. 94–101, Jul. 2006, doi: 10.1145/1139922.1139924.
- [11] N. B. Truong, H. Lee, B. Askwith, and G. M. Lee, “Toward a trust evaluation mechanism in the social Internet of Things,” *Sensors*, vol. 17, no. 6, Jun. 2017, doi: 10.3390/s17061346.
- [12] S. Ruohomaa and L. Kutvonen, “Trust management survey,” in *Proceedings of the 3rd International Conference on Trust Management (iTrust)*, Springer, Berlin, Heidelberg, pp. 77–92, 2005, doi: 10.1007/11429760_6.
- [13] H. Mousa, S. Ben Mokhtar, O. Hasan, O. Younes, M. Hadhoud, and L. Brunie, “Trust management and reputation systems in mobile participatory sensing applications: A survey,” *Computer Networks*, vol. 90, pp. 49–73, 2015, doi: 10.1016/j.comnet.2015.07.011.
- [14] Q. H. Vu, M. Lupu, and B. C. Ooi, “Trust and reputation,” in *Peer-to-Peer Computing*, Berlin, Heidelberg: Springer Berlin Heidelberg, 2010, pp. 183–214, doi: 10.1007/978-3-642-03514-2_7.
- [15] K. Hoffman, C. Nita-Rotaru, and K. Hoffman, “A survey of attack and defense techniques for reputation systems,” *ACM Computing Surveys*, vol. 42, no. 1, pp. 1–31, 2009, doi: 10.1145/1592451.1592452.
- [16] E. Aivaloglou, S. Gritzalis, and C. Skianis, “Trust establishment in sensor networks: Behaviour-based, certificate-based and a combinational approach,” in *2008 International Symposium on Applications and the Internet*, pp. 249–252, 2008, doi: 10.1109/SAINT.2008.82.
- [17] J. Guo, I. R. Chen, and J. J. P. Tsai, “A survey of trust computation models for service management in internet of things systems,” *Computer Communications*, vol. 97, pp. 1–14, 2017, doi: 10.1016/j.comcom.2016.10.012.

- [18] F. Boustanifar, "A trust-based offloading for mobile M2M communications," in *2016 IEEE 13th International Conference on Ubiquitous Intelligence and Computing, Advanced and Trusted Computing, Scalable Computing and Communications, Cloud and Big Data Computing, Internet of People, and Smart World Congress (UIC/ATC/ScalCom/CBDCCom/IoP/SmartWorld)*, pp. 18–22, 2016, doi: 10.1109/UIC-ATC-ScalCom-CBDCCom-IoP-SmartWorld.2016.171.
- [19] A. H. Networks, J. Cho, A. Swami, and I. Chen, "A survey on trust management for mobile ad hoc networks," *IEEE Communications Surveys & Tutorials*, vol. 13, no. 4, pp. 562–583, 2011, doi: 10.1109/SURV.2011.092110.00088.
- [20] J. H. Cho, K. Chan, and S. Adali, "A survey on trust modeling," *ACM Computing Surveys*, vol. 48, no. 2, pp. 1–40, 2015, doi: 10.1145/2815595.
- [21] F. Hosseinpour, A. S. Siddiqui, J. Plosila, and H. Tenhunen, "A security framework for fog networks based on role-based access control and trust models," in *Trust Management XII*, Springer International Publishing, 2018, doi: 10.1007/978-3-319-94845-4_15.
- [22] R. Latif, M. U. Ahmed, S. Tahir, S. Latif, W. Iqbal, and A. Ahmad, "A novel trust management model for edge computing," *Complex & Intelligent Systems*, vol. 8, no. 5, pp. 3747–3763, Oct. 2022, doi: 10.1007/s40747-021-00518-3.
- [23] F. H. Rahman, T. W. Au, S. H. Shah Newaz, and W. S. Suhaili, "Trustworthiness in fog: A fuzzy approach," in *Proceedings of the 2017 VI International Conference on Network, Communication and Computing*, pp. 207–211, 2017, doi: 10.1145/3171592.3171606.
- [24] V. Karagiannis, P. A. Frangoudis, S. Dustdar, and S. Schulte, "Context-aware routing in fog computing systems," *IEEE Transactions on Cloud Computing*, vol. 11, no. 1, pp. 532–549, Jan. 2023, doi: 10.1109/TCC.2021.3102996.
- [25] P. Kochovski, S. Gec, V. Stankovski, M. Bajec, and P. D. Drobintsev, "Trust management in a blockchain based fog computing platform with trustless smart oracles," *Future Generation Computer Systems*, vol. 101, pp. 747–759, 2019, doi: 10.1016/j.future.2019.07.030.

- [26] Y. Hussain *et al.*, “Context-aware trust and reputation model for fog-based IoT,” *IEEE Access*, vol. 8, pp. 31622–31632, 2020, doi: 10.1109/ACCESS.2020.2972968.
- [27] F. Bao and I. R. Chen, “Dynamic trust management for internet of things applications,” in *Proceedings of the 2012 International Workshop on Self-Aware Internet of Things (Self-IoT '12)*, pp. 1–6, 2012, doi: 10.1145/2378023.2378025.
- [28] E. Alemneh, S. M. Senouci, P. Brunet, and T. Tegegne, “A two-way trust management system for fog computing,” *Future Generation Computer Systems*, vol. 106, pp. 206–220, 2020, doi: 10.1016/j.future.2019.12.045.
- [29] L. Atzori, A. Iera, G. Morabito, and M. Nitti, “The social Internet of Things (SIoT) – when social networks meet the Internet of Things: Concept, architecture and network characterization,” *Computer Networks*, vol. 56, no. 16, pp. 3594–3608, Nov. 2012, doi: 10.1016/j.comnet.2012.07.010.
- [30] S. Alam, S. Zardari, S. Noor, S. Ahmed, and H. Mouratidis, “Trust management in social Internet of Things (SIoT): A survey,” *IEEE Access*, vol. 10, pp. 108924–108954, 2022, doi: 10.1109/ACCESS.2022.3213699.
- [31] S. S. Roy, B. J. R. Sahu, and S. Dash, “Enhanced trust management for building trustworthy social Internet of Things network,” *IET Networks*, vol. 14, no. 1, pp. 1–19, Jan. 2025, doi: 10.1049/ntw2.12111.
- [32] R. Mayrhofer, J. Vander Stoep, C. Brubaker, and N. Kravovich, “The Android platform security model,” *ACM Transactions on Privacy and Security*, vol. 24, no. 3, pp. 1–35, Aug. 2021, doi: 10.1145/3448609.
- [33] A. K. Junejo, N. Komninos, M. Sathiyarayanan, and B. S. Chowdhry, “Trustee: A trust management system for fog-enabled cyber physical systems,” *IEEE Transactions on Emerging Topics in Computing*, vol. 9, no. 4, pp. 1736–1747, 2021, doi: 10.1109/TETC.2019.2957394.
- [34] M. Al-Khafajiy *et al.*, “COMITMENT: A fog computing trust management approach,” *Journal of Parallel and Distributed Computing*, vol. 137, pp. 1–16, 2020, doi: 10.1016/j.jpdc.2019.10.006.

- [35] S. Dhelim, N. Aung, M. T. Kechadi, H. Ning, L. Chen, and A. Lakas, "Trust2Vec: Large-scale IoT trust management system based on signed network embeddings," *IEEE Internet of Things Journal*, vol. 10, no. 1, pp. 553–562, Jan. 2023, doi: 10.1109/JIOT.2022.3201772.
- [36] C. Zhang, S. Lan, L. Wang, L. Liu, and J. Ren, "Trust attacks and defense in the social Internet of Things: Taxonomy and simulation-based evaluation," *Sensors*, vol. 25, no. 24, p. 7513, Dec. 2025, doi: 10.3390/s25247513.
- [37] Y. Alghofaili and M. A. Rassam, "A dynamic trust-related attack detection model for IoT devices and services based on the deep long short-term memory technique," *Sensors*, vol. 23, no. 8, p. 3814, Apr. 2023, doi: 10.3390/s23083814.
- [38] I. R. Chen, J. Guo, and F. Bao, "Trust management for SOA-based IoT and its application to service composition," *IEEE Transactions on Services Computing*, vol. 9, no. 3, pp. 482–495, 2016, doi: 10.1109/TSC.2014.2365797.
- [39] Q. Yaseen, M. Aldwairi, Y. Jararweh, M. Al-Ayyoub, and B. Gupta, "Collusion attacks mitigation in Internet of Things: A fog based model," *Multimedia Tools and Applications*, vol. 77, no. 14, pp. 18249–18268, Jul. 2018, doi: 10.1007/s11042-017-5288-3.
- [40] A. B. C. Douss, R. Abassi, and S. G. El Fatmi, "A trust management based security mechanism against collusion attacks in a MANET environment," in *2014 Ninth International Conference on Availability, Reliability and Security*, pp. 325–332, Sep. 2014, doi: 10.1109/ARES.2014.50.
- [41] B. Li, R. Lu, W. Wang, and K.-K. R. Choo, "DDOA: A Dirichlet-based detection scheme for opportunistic attacks in smart grid cyber-physical system," *IEEE Transactions on Information Forensics and Security*, vol. 11, no. 11, pp. 2415–2425, Nov. 2016, doi: 10.1109/TIFS.2016.2576898.
- [42] C. V. L. Mendoza and J. H. Kleinschmidt, "Mitigating On-Off attacks in the internet of things using a distributed trust management scheme," *International Journal of Distributed Sensor Networks*, vol. 11, no. 11, pp. 1–11, Nov. 2015, doi: 10.1155/2015/859731.

- [43] M. A. Shaaban, A. S. Alsharkawy, M. T. AbouKreisha, and M. A. Razek, "Efficient ECC-based authentication scheme for fog-based IoT environment," *International Journal of Computer Networks & Communications (IJCNC)*, vol. 15, no. 4, pp. 31–50, Jul. 2023, doi: 10.5121/ijcnc.2023.15404.
- [44] B. Jafarian, N. Yazdani, and M. Sayad Haghighi, "Discrimination-aware trust management for social Internet of Things," *Computer Networks*, vol. 178, p. 107254, Sep. 2020, doi: 10.1016/j.comnet.2020.107254.
- [45] J. Ren and T. Qin, "A novel multidimensional trust evaluation and fusion mechanism in fog-based Internet of Things," *Computer Networks*, vol. 217, p. 109354, Nov. 2022, doi: 10.1016/j.comnet.2022.109354.
- [46] S. M. Zarei and R. Fotohi, "Defense against flooding attacks using probabilistic thresholds in the Internet of Things ecosystem," *Security and Privacy*, vol. 4, no. 3, e152, May 2021, doi: 10.1002/spy2.152.
- [47] A. Sawas and H. E. Z. Farag, "Real-time detection of stealthy IoT-based cyber-attacks on power distribution systems: A novel anomaly prediction approach," *Electric Power Systems Research*, vol. 223, p. 109496, Oct. 2023, doi: 10.1016/j.epsr.2023.109496.
- [48] S. Singh, M. Kandpal, and H. B. Bhadka, "Trust-aware and adaptive malicious node detection in fog network using independent DQN with centralized training," *Journal of Applied Science and Technology Trends*, vol. 7, no. 1, pp. 47–66, Jan. 2026, doi: 10.38094/jastt71547.
- [49] N. Mantas, M. Louta, E. Karapistoli, G. T. Karetsos, S. Kraounakis, and M. S. Obaidat, "Towards an incentive-compatible, reputation-based framework for stimulating cooperation in opportunistic networks: A survey," *IET Networks*, vol. 6, no. 6, pp. 169–178, Nov. 2017, doi: 10.1049/iet-net.2017.0079.
- [50] S. Madhavi and K. Duraiswamy, "Flooding attack aware secure AODV," *Journal of Computer Science*, vol. 9, no. 1, pp. 105–113, Jan. 2013, doi: 10.3844/jcssp.2013.105.113.

- [51] C. Rodrigues, W. S. S. Júnior, W. Oliveira, and I. Lima, “A data rate monitoring approach for cyberattack detection in digital twin communication,” *Sensors*, vol. 25, no. 24, p. 7476, Dec. 2025, doi: 10.3390/s25247476.
- [52] A. Yassine, S. Singh, M. S. Hossain, and G. Muhammad, “IoT big data analytics for smart homes with fog and cloud computing,” *Future Generation Computer Systems*, vol. 91, pp. 563–573, Feb. 2019, doi: 10.1016/j.future.2018.08.040.
- [53] A. M. Kowshalya and M. L. Valarmathi, “Trust management in the social Internet of Things,” *Wireless Personal Communications*, vol. 96, no. 2, pp. 2681–2691, 2017, doi: 10.1007/s11277-017-4319-8.
- [54] V. B. Reddy, A. Negi, S. Venkataraman, and V. R. Venkataraman, “A similarity based trust model to mitigate badmouthing attacks in Internet of Things (IoT),” in *2019 IEEE 5th World Forum on Internet of Things (WF-IoT)*, pp. 278–282, 2019, doi: 10.1109/WF-IoT.2019.8767170.
- [55] S. A. Soleymani *et al.*, “A secure trust model based on fuzzy logic in vehicular ad hoc networks with fog computing,” *IEEE Access*, vol. 5, pp. 15619–15629, 2017, doi: 10.1109/ACCESS.2017.2733225.
- [56] A. M. Rahmani, P. Liljeberg, J. S. Preden, and A. Jantsch, “Fog computing in the Internet of Things: Intelligence at the edge,” in *Fog Computing in the Internet of Things: Intelligence at the Edge*, Springer, pp. 1–172, 2018, doi: 10.1007/978-3-319-57639-8.
- [57] C. Puliafito, E. Mingozzi, F. Longo, A. Puliafito, and O. Rana, “Fog computing for the Internet of Things: A survey,” *ACM Transactions on Internet Technology*, vol. 19, no. 2, pp. 1–41, Apr. 2019, doi: 10.1145/3301443.
- [58] H. Gupta, A. Vahid Dastjerdi, S. K. Ghosh, and R. Buyya, “iFogSim: A toolkit for modeling and simulation of resource management techniques in the internet of things, edge and fog computing environments,” *Software: Practice and Experience*, vol. 47, no. 9, pp. 1275–1296, Sep. 2017, doi: 10.1002/spe.2509.
- [59] A. V. Dastjerdi, H. Gupta, R. N. Calheiros, S. K. Ghosh, and R. Buyya, “Fog computing: Principles, architectures, and applications,” in *Internet of Things*:

Principles and Paradigms, Elsevier Inc., pp. 61–75, 2016, doi: 10.1016/B978-0-12-805395-9.00004-6.

- [60] M. Aazam and E. N. Huh, “Fog computing micro datacenter based dynamic resource estimation and pricing model for IoT,” in *2015 IEEE 29th International Conference on Advanced Information Networking and Applications (AINA)*, pp. 687–694, 2015, doi: 10.1109/AINA.2015.254.
- [61] C. N. Höfer and G. Karagiannis, “Cloud computing services: Taxonomy and comparison,” *Journal of Internet Services and Applications*, vol. 2, no. 2, pp. 81–94, 2011, doi: 10.1007/s13174-011-0027-x.
- [62] J. Ni, K. Zhang, X. Lin, and X. S. Shen, “Securing fog computing for Internet of Things applications: Challenges and solutions,” *IEEE Communications Surveys & Tutorials*, vol. 20, no. 1, pp. 601–628, 2018, doi: 10.1109/COMST.2017.2762345.
- [63] A. Buchenscheit, F. Schaub, F. Kargl, and M. Weber, “A VANET-based emergency vehicle warning system,” in *2009 IEEE Vehicular Networking Conference (VNC)*, pp. 1–8, 2009, doi: 10.1109/VNC.2009.5416386.
- [64] J. Jin, J. Gubbi, S. Marusic, and M. Palaniswami, “An information framework for creating a smart city through Internet of Things,” *IEEE Internet of Things Journal*, vol. 1, no. 2, pp. 112–121, Apr. 2014, doi: 10.1109/JIOT.2013.2296516.
- [65] C. T. Fan, Z. Y. Wu, C. P. Chang, and S. M. Yuan, “Web resource cacheable edge device in fog computing,” in *2016 15th International Symposium on Parallel and Distributed Computing (ISPDC)*, pp. 432–439, 2016, doi: 10.1109/ISPDC.2016.74.
- [66] P. Hu, S. Dhelim, H. Ning, and T. Qiu, “Survey on fog computing: Architecture, key technologies, applications and open issues,” *Journal of Network and Computer Applications*, vol. 98, pp. 27–42, Nov. 2017, doi: 10.1016/j.jnca.2017.09.002.
- [67] A. A.-N. Patwary *et al.*, “Authentication, access control, privacy, threats and trust management towards securing fog computing environments: A review,” *arXiv preprint arXiv:2003.00395*, 2020. Available: <http://arxiv.org/abs/2003.00395>.

- [68] R. K. Naha *et al.*, “Fog computing: Survey of trends, architectures, requirements, and research directions,” *IEEE Access*, vol. 6, pp. 47980–48009, 2018, doi: 10.1109/ACCESS.2018.2866491.
- [69] D. Dragomir, L. Gheorghe, S. Costea, and A. Radovici, “A survey on secure communication protocols for IoT systems,” in *Proceedings - 2016 International Workshop on Secure Internet of Things (SIoT)*, pp. 47–62, 2016, doi: 10.1109/SIoT.2016.012.
- [70] V. Meena, M. Gorripatti, and T. Suriya Praba, “Trust enforced computational offloading for health care applications in fog computing,” *Wireless Personal Communications*, vol. 120, pp. 2969–2987, 2021, doi: 10.1007/s11277-021-08285-7.
- [71] A. Carrera-Rivera, W. Ochoa, F. Larrinaga, and G. Lasa, “How-to conduct a systematic literature review: A quick guide for computer science research,” *MethodsX*, vol. 9, p. 101895, 2022, doi: 10.1016/j.mex.2022.101895.
- [72] A. Botta, W. De Donato, V. Persico, and A. Pescapé, “Integration of cloud computing and Internet of Things: A survey,” *Future Generation Computer Systems*, vol. 56, pp. 684–700, 2014, doi: 10.1016/j.future.2015.09.021.
- [73] S. M. Habib, S. Ries, and M. Mühlhäuser, “Cloud computing landscape and research challenges regarding trust and reputation,” in *Proceedings - 7th International Conference on Ubiquitous Intelligence & Computing and 7th International Conference on Autonomic & Trusted Computing (UIC/ATC)*, pp. 410–415, 2010, doi: 10.1109/UIC-ATC.2010.48.
- [74] T. H. Noor, Q. Z. Sheng, L. Yao, S. Dustdar, and A. H. H. Ngu, “CloudArmor: Supporting reputation-based trust management for cloud services,” *IEEE Transactions on Parallel and Distributed Systems*, vol. 27, no. 2, pp. 367–380, Feb. 2016, doi: 10.1109/TPDS.2015.2408613.
- [75] M. Ghafoorian, D. Abbasinezhad-Mood, and H. Shakeri, “A thorough trust and reputation based RBAC model for secure data storage in the cloud,” *IEEE Transactions on Parallel and Distributed Systems*, vol. 30, no. 4, pp. 778–788, Apr. 2019, doi: 10.1109/TPDS.2018.2870652.

- [76] Z. Yan, X. Li, M. Wang, and A. V. Vasilakos, "Flexible data access control based on trust and reputation in cloud computing," *IEEE Transactions on Cloud Computing*, vol. 5, no. 3, pp. 485–498, Jul. 2017, doi: 10.1109/TCC.2015.2469662.
- [77] H. Hassan, A. I. El-Desouky, A. Ibrahim, E. S. M. El-Kenawy, and R. Arnous, "Enhanced QoS-based model for trust assessment in cloud computing environment," *IEEE Access*, vol. 8, pp. 43752–43763, 2020, doi: 10.1109/ACCESS.2020.2978452.
- [78] S. Ganeriwal, L. K. Balzano, and M. B. Srivastava, "Reputation-based framework for high integrity sensor networks," *ACM Transactions on Sensor Networks*, vol. 4, no. 3, pp. 1–37, May 2008, doi: 10.1145/1362542.1362546.
- [79] L. Wang, Y. Li, and L. Zuo, "Trust management for IoT devices based on federated learning and blockchain," *The Journal of Supercomputing*, vol. 81, no. 1, pp. 1–25, Jan. 2025, doi: 10.1007/s11227-024-06715-4.
- [80] R. Arul, Y. D. Al-Otaibi, W. S. Alnumay, U. Tariq, U. Shoaib, and M. D. J. Piran, "Multi-modal secure healthcare data dissemination framework using blockchain in IoMT," *Personal and Ubiquitous Computing*, 2021, doi: 10.1007/s00779-021-01527-2.
- [81] O.-K. Shahryari, H. Pedram, V. Khajehvand, and M. Dehghan TakhtFooladi, "Energy-efficient and delay-guaranteed computation offloading for fog-based IoT networks," *Computer Networks*, vol. 182, p. 107511, Dec. 2020, doi: 10.1016/j.comnet.2020.107511
- [82] P. Singh and R. Singh, "Energy-efficient delay-aware task offloading in fog-cloud computing system for IoT sensor applications," *Journal of Network and Systems Management*, vol. 30, no. 1, p. 14, Jan. 2022, doi: 10.1007/s10922-021-09622-8.
- [83] S. Aleteemat and M. Shurman, "Task offloading scheduling in fog computing using hybrid genetic algorithm," in *2023 14th International Conference on Information and Communication Systems (ICICS)*, pp. 1–6, Nov. 2023, doi: 10.1109/ICICS60529.2023.10330532.

- [84] J. Yadav and S. Sangwan, "Dynamic offloading framework in fog computing," *International Journal of Engineering Trends and Technology*, vol. 70, no. 7, pp. 32–42, Jul. 2022, doi: 10.14445/22315381/IJETT-V70I7P204.
- [85] F. Alenizi and O. Rana, "Dynamically controlling offloading thresholds in fog systems," *Sensors*, vol. 21, no. 7, p. 2512, Apr. 2021, doi: 10.3390/s21072512.
- [86] S. Vakilian, A. Fanian, H. Falsafain, and T. A. Gulliver, "Node cooperation for workload offloading in a fog computing network via multi-objective optimization," *Journal of Network and Computer Applications*, vol. 205, p. 103428, Sep. 2022, doi:10.1016/j.jnca.2022.103428.
- [87] M. Debe, K. Salah, M. H. U. Rehman, and D. Svetinovic, "IoT public fog nodes reputation system: A decentralized solution using Ethereum blockchain," *IEEE Access*, vol. 7, pp. 178082–178093, 2019, doi: 10.1109/ACCESS.2019.2958355.
- [88] F. H. Rahman, T. W. Au, S. H. S. Newaz, W. S. Suhaili, and G. M. Lee, "Find my trustworthy fogs: A fuzzy-based trust evaluation framework," *Future Generation Computer Systems*, vol. 109, pp. 562–572, Aug. 2020, doi: 10.1016/j.future.2018.05.061.
- [89] W. Ben Daoud, A. Meddeb-Makhlouf, F. Zarai, and M. S. Obaidat, "Distributed trust-based monitoring approach for fog/cloud networks," in *Advanced Multimedia and Ubiquitous Engineering* (Smart Innovation, Systems and Technologies, vol. 1054), Springer Singapore, pp. 43–49, 2020, doi: 10.1007/978-981-15-0135-7_6.
- [90] Y. Hussain and H. Zhiqiu, "TRFIoT: Trust and reputation model for fog-based IoT," in *Computer Security*, Springer International Publishing, pp. 319–331, 2018, doi: 10.1007/978-3-030-00021-9_23.
- [91] E. Wong *et al.*, "FogTrust: Fog-integrated multi-leveled trust management mechanism for Internet of Things," *Technologies*, vol. 11, no. 1, p. 11, Jan. 2023, doi: 10.3390/technologies11010011.
- [92] K. Fan, Y. Ren, and Z. Yan, "Trust management in fog/edge computing by means of blockchain technologies," in *2018 IEEE International Conference on Internet of Things (iThings) and IEEE Green Computing and Communications (GreenCom) and*

IEEE Cyber, Physical and Social Computing (CPSCoM) and IEEE Smart Data (SmartData), pp. 1349–1354, 2018, doi: 10.1109/Cybermatics_2018.232.

- [93] J. Liang, M. Zhang, and V. C. M. Leung, “A reliable trust computing mechanism based on multi-source feedback and fog computing in social sensor cloud,” *IEEE Internet of Things Journal*, vol. 7, no. 6, pp. 5481–5490, Jun. 2020, doi: 10.1109/JIOT.2020.2981005.
- [94] G. Zhang, T. Wang, M. Z. A. Bhuiyan, and G. Wang, “A fog-based hierarchical trust mechanism for sensor-cloud underlying structure,” in *2017 IEEE 15th International Symposium on Parallel and Distributed Processing with Applications and 16th IEEE International Conference on Ubiquitous Computing and Communications (ISPA/IUCC)*, pp. 481–485, 2017, doi: 10.1109/ISPA/IUCC.2017.00079.
- [95] T. Wang, G. Zhang, M. Z. A. Bhuiyan, A. Liu, W. Jia, and M. Xie, “A novel trust mechanism based on fog computing in sensor–cloud system,” *Future Generation Computer Systems*, vol. 109, pp. 573–582, Aug. 2020, doi: 10.1016/j.future.2018.05.049.
- [96] G. Rathee, R. Sandhu, H. Saini, M. Sivaram, and V. Dhasarathan, “A trust computed framework for IoT devices and fog computing environment,” *Wireless Networks*, vol. 26, pp. 2335–2351, 2020, doi: 10.1007/s11276-019-02106-3.
- [97] P. Ramamurthy and M. Nandagopal, “Bi-directional trust management system in fog computing using logistic regression,” *Indonesian Journal of Electrical Engineering and Computer Science*, vol. 29, no. 2, pp. 808–815, Feb. 2023, doi: 10.11591/ijeecs.v29.i2.pp808-815.
- [98] A. Almas, W. Iqbal, A. Altaf, K. Saleem, S. Mussiraliyeva, and M. W. Iqbal, “Context-based adaptive fog computing trust solution for time-critical smart healthcare systems,” *IEEE Internet of Things Journal*, vol. 10, no. 12, pp. 10575–10586, Jun. 2023, doi: 10.1109/JIOT.2023.3242126.
- [99] X. Zheng, S. Yang, and X. Wang, “A reliable and decentralized trust management model for fog computing in industrial IoT,” in *2023 IEEE/IFIP Network Operations and Management Symposium*, pp. 1–6, May 2023, doi: 10.1109/NOMS56928.2023.10154305.

- [100] J. Al Muhtadi, R. A. Alamri, F. A. Khan, and K. Saleem, "Subjective logic-based trust model for fog computing," *Computer Communications*, vol. 178, pp. 221–233, Oct. 2021, doi: 10.1016/j.comcom.2021.05.016.
- [101] A. M. Alwakeel and A. K. Alnaim, "Trust management and resource optimization in edge and fog computing using the CyberGuard framework," *Sensors*, vol. 24, no. 13, p. 4308, Jul. 2024, doi: 10.3390/s24134308.
- [102] S. O. Ogundoyin and I. A. Kamil, "A trust management system for fog computing services," *Internet of Things*, vol. 14, p. 100382, Jun. 2021, doi: 10.1016/j.iot.2021.100382.
- [103] Z. Sharif, L. T. Jung, M. Ayaz, M. Yahya, and D. Khan, "Smart home automation by Internet-of-Things edge computing platform," *International Journal of Advanced Computer Science and Applications*, vol. 13, no. 4, pp. 455–465, 2022, doi: 10.14569/IJACSA.2022.0130455.
- [104] H. Yar, A. S. Imran, Z. A. Khan, M. Sajjad, and Z. Kastrati, "Towards smart home automation using IoT-enabled edge-computing paradigm," *Sensors*, vol. 21, no. 14, p. 4932, Jul. 2021, doi: 10.3390/s21144932.
- [105] I. Froiz-Míguez, T. M. Fernández-Caramés, P. Fraga-Lamas, and L. Castedo, "Design, implementation and practical evaluation of an IoT home automation system for fog computing applications based on MQTT and ZigBee-WiFi sensor nodes," *Sensors*, vol. 18, no. 8, p. 2660, Aug. 2018, doi: 10.3390/s18082660.
- [106] M. Rahimi, M. Songhorabadi, and M. H. Kashani, "Fog-based smart homes: A systematic review," *Journal of Network and Computer Applications*, vol. 153, p. 102531, Mar. 2020, doi: 10.1016/j.jnca.2020.102531.
- [107] P. Pawar and K. P. Vittal, "Design and development of advanced smart energy management system integrated with IoT framework in smart grid environment," *Journal of Energy Storage*, vol. 25, p. 100846, Oct. 2019, doi: 10.1016/j.est.2019.100846.

- [108] T. M. Fernández-Caramés, “An intelligent power outlet system for the smart home of the Internet of Things,” *International Journal of Distributed Sensor Networks*, vol. 11, no. 11, p. 214805, Nov. 2015, doi: 10.1155/2015/214805.
- [109] C. Stolojescu-Crisan, C. Crisan, and B. P. Butunoi, “An IoT-based smart home automation system,” *Sensors*, vol. 21, no. 11, p. 3784, May 2021, doi: 10.3390/s21113784.
- [110] C. Xia, W. Li, X. Chang, F. Delicato, T. Yang, and A. Zomaya, “Edge-based energy management for smart homes,” in *2018 IEEE 16th International Conference on Dependable, Autonomic and Secure Computing, 16th International Conference on Pervasive Intelligence and Computing, 4th International Conference on Big Data Intelligence and Computing and Cyber Science and Technology Congress (DASC/PiCom/DataCom/CyberSciTec)*, pp. 849–856, 2018, doi: 10.1109/DASC/PiCom/DataCom/CyberSciTec.2018.00-19.
- [111] R. Jaiswal, R. Davidrajuh, and C. Rong, “Fog computing for realizing smart neighborhoods in smart grids,” *Computers*, vol. 9, no. 3, p. 76, 2020, doi: 10.3390/computers9030076.
- [112] M. Umair, M. A. Cheema, B. Afzal, and G. Shah, “Energy management of smart homes over fog-based IoT architecture,” *Sustainable Computing: Informatics and Systems*, vol. 39, p. 100898, Sep. 2023, doi: 10.1016/j.suscom.2023.100898.
- [113] M. Abdullah, A. Faruque, K. Vatanparvar, and G. S. Member, “Energy management-as-a-service over fog computing platform,” in *2016 IEEE First International Conference on Internet-of-Things Design and Implementation (IoTDI)*, pp. 161–169, 2016, doi: 10.1109/IoTDI.2015.34.
- [114] Q. Sun *et al.*, “A comprehensive review of smart energy meters in intelligent energy networks,” *IEEE Internet of Things Journal*, vol. 3, no. 4, pp. 464–479, Aug. 2016, doi: 10.1109/JIOT.2015.2512325.
- [115] M. Amadeo, A. Giordano, and C. Mastroianni, “On the integration of information centric networking and fog computing for smart home services,” in *Internet of Things: Concepts, Technologies and Applications* (Smart Innovation, Systems and

Technologies, vol. 130), Springer International Publishing, pp. 83–102, 2019, doi: 10.1007/978-3-319-96550-5_4.

- [116] T. Sultana and K. A. Wahid, “IoT-Guard: Event-driven fog-based video surveillance system for real-time security management,” *IEEE Access*, vol. 7, pp. 134881–134894, 2019, doi: 10.1109/ACCESS.2019.2941978.
- [117] G. Alsuhli and A. Khattab, “A fog-based IoT platform for smart buildings,” in *2019 International Conference on Innovative Trends in Computer Engineering (ITCE)*, pp. 316–321, 2019, doi: 10.1109/itce.2019.8646480.
- [118] C. Huang, R. Lu, and K.-K. R. Choo, “Vehicular fog computing: Architecture, use case, and security and forensic challenges,” *IEEE Communications Magazine*, vol. 55, no. 11, pp. 105–111, Nov. 2017, doi: 10.1109/MCOM.2017.1700322.
- [119] S. Al-Sultan, M. M. Al-Doori, A. H. Al-Bayatti, and H. Zedan, “A comprehensive survey on vehicular ad hoc network,” *Journal of Network and Computer Applications*, vol. 37, pp. 380–392, Jan. 2014, doi: 10.1016/j.jnca.2013.02.036.
- [120] M. Petracca, P. Pagano, R. Pelliccia, M. Ghibaudi, C. Salvadori, and C. Nastasi, “On-board unit hardware and software design for vehicular ad-hoc networks,” in *Roadside Networks for Vehicular Communications: Architectures, Applications, and Test Fields*, IGI Global, pp. 38–56, 2013, doi: 10.4018/978-1-4666-2223-4.ch002.
- [121] Q. Yang, L. Wang, W. Xia, Y. Wu, and L. Shen, “Development of on-board unit in vehicular ad-hoc network for highways,” in *2014 International Conference on Connected Vehicles and Expo (ICCVE)*, pp. 457–462, 2014, doi: 10.1109/ICCVE.2014.7297589.
- [122] X. Xu, K. Liu, K. Xiao, H. Ren, L. Feng, and C. Chen, “Design and implementation of a fog computing based collision warning system in VANETs,” in *2018 IEEE International Symposium on Product Compliance Engineering - Asia (ISPCE-CN)*, pp. 1–6, 2018, doi: 10.1109/ISPCE-CN.2018.8805783.
- [123] C. Zhu, G. Pastor, Y. Xiao, Y. Li, and A. Ylä-Jääski, “Fog following me: Latency and quality balanced task allocation in vehicular fog computing,” in *2018 15th Annual*

IEEE International Conference on Sensing, Communication, and Networking (SECON), pp. 1–9, 2018, doi: 10.1109/SAHCN.2018.8397129.

- [124] X. Hou, Y. Li, M. Chen, D. Wu, D. Jin, and S. Chen, “Vehicular fog computing: A viewpoint of vehicles as the infrastructures,” *IEEE Transactions on Vehicular Technology*, vol. 65, no. 6, pp. 3860–3873, Jun. 2016, doi: 10.1109/TVT.2016.2532863.
- [125] C. Zhu, G. Pastor, Y. Xiao, and A. Ylä-Jääski, “Vehicular fog computing for video crowdsourcing: Applications, feasibility, and challenges,” *IEEE Communications Magazine*, vol. 56, no. 10, pp. 58–63, Oct. 2018, doi: 10.1109/MCOM.2018.1800116.
- [126] J. Liu *et al.*, “Secure intelligent traffic light control using fog computing,” *Future Generation Computer Systems*, vol. 78, pp. 817–824, 2018, doi: 10.1016/j.future.2017.02.017.
- [127] C. Tang, X. Wei, C. Zhu, J. J. P. C. Rodrigues, and W. E. I. Chen, “Towards smart parking based on fog computing,” *IEEE Access*, vol. 6, pp. 70172–70185, 2018, doi: 10.1109/ACCESS.2018.2880972.
- [128] Q. Wu *et al.*, “Smart fog based workflow for traffic control networks,” *Future Generation Computer Systems*, vol. 97, pp. 825–835, 2019, doi: 10.1016/j.future.2019.02.058.
- [129] Z. Ning, J. Huang, and X. Wang, “Vehicular fog computing: Enabling real-time traffic management for smart cities,” *IEEE Wireless Communications*, vol. 26, no. 1, pp. 87–93, Feb. 2019, doi: 10.1109/MWC.2019.1700441.
- [130] Y. P. Fallah and M. K. Khandani, “Analysis of the coupling of communication network and safety application in cooperative collision warning systems,” in *2015 IEEE Vehicular Networking Conference (VNC)*, pp. 228–237, 2015, doi: 10.1109/VNC.2015.7385584.
- [131] E. Alemneh, S. M. Senouci, and P. Brunet, “PV-Alert: A fog-based architecture for safeguarding vulnerable road users,” in *2021 IEEE International Conference on Communications Workshops (ICC Workshops)*, pp. 1–6, 2021, doi: 10.1109/ICCWorkshops50388.2021.9473846.

- [132] M. Arshad *et al.*, “Beacon trust management system and fake data detection in vehicular ad-hoc networks,” *IET Intelligent Transport Systems*, vol. 13, no. 10, pp. 1459–1467, 2019, doi: 10.1049/iet-its.2018.5117.
- [133] A. Kumari, S. Tanwar, S. Tyagi, and N. Kumar, “Fog computing for Healthcare 4.0 environment: Opportunities and challenges,” *Computers and Electrical Engineering*, vol. 72, pp. 1–13, 2018, doi: 10.1016/j.compeleceng.2018.08.015.
- [134] T. N. Gia, M. Jiang, A. M. Rahmani, T. Westerlund, P. Liljeberg, and H. Tenhunen, “Fog computing in healthcare internet of things: A case study on ECG feature extraction,” in *2015 IEEE 15th International Conference on Computer and Information Technology; 14th IEEE International Conference on Ubiquitous Computing and Communications; 13th IEEE International Conference on Dependable, Autonomic and Secure Computing; 13th IEEE International Conference on Pervasive Intelligence and Computing (CIT/IUCC/DASC/PICOM)*, pp. 356–363, 2015, doi: 10.1109/CIT/IUCC/DASC/PICOM.2015.51.
- [135] A. M. Rahmani, T. Nguyen, B. Negash, and A. Anzanpour, “Exploiting smart e-Health gateways at the edge of healthcare Internet-of-Things: A fog computing approach,” *Future Generation Computer Systems*, vol. 78, pp. 641–658, 2018, doi: 10.1016/j.future.2017.02.014.
- [136] P. H. Vilela, J. J. P. C. Rodrigues, P. Solic, K. Saleem, and V. Furtado, “Performance evaluation of a fog-assisted IoT solution for e-Health applications,” *Future Generation Computer Systems*, vol. 97, pp. 379–386, 2019, doi: 10.1016/j.future.2019.02.055.
- [137] F. A. Kraemer, A. E. Braten, N. Tamkittikhun, and D. Palma, “Fog computing in healthcare—A review and discussion,” *IEEE Access*, vol. 5, pp. 9206–9222, 2017, doi: 10.1109/ACCESS.2017.2704100.
- [138] R. Craciunescu, A. Mihovska, M. Mihaylov, S. Kyriazakos, R. Prasad, and S. Halunga, “Implementation of fog computing for reliable e-health applications,” in *2015 49th Asilomar Conference on Signals, Systems and Computers*, pp. 459–463, 2015, doi: 10.1109/ACSSC.2015.7421170.

- [139] M. Soni, S. Chauhan, B. Bajpai, and T. Puri, "An approach to enhance fall detection using machine learning classifier," in *2020 12th International Conference on Computational Intelligence and Communication Networks (CICN)*, pp. 229–233, 2020, doi: 10.1109/CICN49253.2020.9242634.
- [140] A. Dev and S. K. Malik, "IoT and fog computing based prediction and monitoring system for stroke disease," *Turkish Journal of Computer and Mathematics Education (TURCOMAT)*, vol. 12, no. 6, pp. 3210–3218, 2021.
- [141] A. Elhadad, F. Alanazi, A. I. Taloba, and A. Abozeid, "Fog computing service in the healthcare monitoring system for managing the real-time notification," *Journal of Healthcare Engineering*, vol. 2022, p. 5337733, Mar. 2022, doi: 10.1155/2022/5337733.
- [142] H. Ben Hassen, W. Dghais, and B. Hamdi, "An e-health system for monitoring elderly health based on Internet of Things and fog computing," *Health Information Science and Systems*, vol. 7, no. 1, p. 23, Dec. 2019, doi: 10.1007/s13755-019-0087-z.
- [143] U. Hasan, C. Fernandes, and N. Ul Hasan, "Fog computing assisted anaesthesia monitoring to enhance realtime surgical efficiency," in *2024 IEEE International Conference on Pervasive Computing and Communications Workshops and other Affiliated Events (PerCom Workshops)*, pp. 1–6, 2024, doi: 10.1109/PerComWorkshops59983.2024.10503024.
- [144] J. A. Rincon, S. Guerra-Ojeda, C. Carrascosa, and V. Julian, "An IoT and fog computing-based monitoring system for cardiovascular patients with automatic ECG classification using deep neural networks," *Sensors*, vol. 20, no. 24, p. 7353, Dec. 2020, doi: 10.3390/s20247353.
- [145] M. Erfan, B. A. Hridita, M. Shoyaib, and M. S. Islam, "A comprehensive survey of code offloading mechanisms for mobile cloud computing," *International Journal of Computer Applications*, vol. 147, no. 8, pp. 20–29, Aug. 2016, doi: 10.5120/ijca2016911223.
- [146] C. Bormann, M. Ersue, and A. Keranen, "Terminology for constrained-node networks," IETF RFC 7228, May 2014, doi: 10.17487/RFC7228.

- [147] Z. Shelby, K. Hartke, and C. Bormann, “The constrained application protocol (CoAP),” IETF RFC 7252, Jun. 2014, doi: 10.17487/RFC7252.
- [148] S. Cheshire and M. Krochmal, “DNS-based service discovery,” IETF RFC 6763, Feb. 2013, doi: 10.17487/RFC6763. Available: <https://datatracker.ietf.org/doc/html/rfc6763>
- [149] E. Guttman, C. Perkins, J. Veizades, and M. Day, “Service location protocol, version 2,” IETF RFC 2608, Jun. 1999, doi: 10.17487/RFC2608. Available: <https://datatracker.ietf.org/doc/html/rfc2608>
- [150] UPnP Forum, “UPnP device architecture 2.0,” Open Connectivity Foundation, 2015. [Online]. Available: <https://openconnectivity.org/upnp-specs/UPnP-arch-DeviceArchitecture-v2.0-20200417.pdf>
- [151] B. Carballido Villaverde, R. D. P. Alberola, A. J. Jara, S. Fedor, S. K. Das, and D. Pesch, “Service discovery protocols for constrained machine-to-machine communications,” *IEEE Communications Surveys & Tutorials*, vol. 16, no. 1, pp. 41–60, First Quarter 2014, doi: 10.1109/SURV.2013.102213.00229.
- [152] C. Bormann, S. Lemay, H. Tschofenig, K. Hartke, and B. Silverajan, “CoAP (constrained application protocol) over TCP, TLS, and WebSockets,” IETF RFC 8323, Feb. 2018, doi: 10.17487/RFC8323.
- [153] K. P. Gaffney, M. Prammer, L. Brasfield, D. R. Hipp, D. Kennedy, and J. M. Patel, “SQLite: Past, present, and future,” *Proceedings of the VLDB Endowment*, vol. 15, no. 12, pp. 3535–3547, Aug. 2022, doi: 10.14778/3554821.3554842.
- [154] D. Iracleous, I. Aldrighetti, G. Flanagan, C. Kramer, A. Teryaki, and Dreadnaut, “phpLiteAdmin,” 2019. [Online]. Available: <https://www.phpliteadmin.org/>
- [155] P. Leach, M. Mealling, and R. Salz, “A universally unique identifier (UUID) URN namespace,” IETF RFC 4122, Jul. 2005, doi: 10.17487/RFC4122. Available: <https://tools.ietf.org/html/rfc4122>
- [156] R. Maes and V. van der Leest, “Countering the intrusion of scalable PUFs: The case of SRAM PUFs,” in *2014 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, pp. 1–6, 2014, doi: 10.7873/DATE.2014.288.

- [157] J. Wang, L. Xu, and W. Shen, “IoT-ID: A dynamic and lightweight device identification approach for Internet of Things,” *IEEE Internet of Things Journal*, vol. 5, no. 4, pp. 3336–3347, Aug. 2018, doi: 10.1109/JIOT.2018.2842247.
- [158] S. Baheti, S. Badiger, and Y. Simmhan, “VioLET: A large-scale virtual environment for Internet of Things,” *ACM Transactions on Cyber-Physical Systems*, vol. 5, no. 3, pp. 1–39, Jul. 2021, doi: 10.1145/3446346.
- [159] Texas Instruments, “INA219 zero-drift, bidirectional current/power monitor with I2C interface,” datasheet, Sep. 2008 [Revised 2015]. [Online]. Available: <https://www.ti.com/lit/ds/symlink/ina219.pdf>
- [160] L. Skorin-Kapov and M. Matijasevic, “Analysis of QoS requirements for e-Health services and mapping to evolved packet system QoS classes,” *International Journal of Telemedicine and Applications*, vol. 2010, p. 628086, 2010, doi: 10.1155/2010/628086.
- [161] J. D. Warner *et al.*, “scikit-fuzzy: Fuzzy logic toolkit for SciPy,” 2024. [Online]. Available: <https://github.com/scikit-fuzzy/scikit-fuzzy>

Appendix A: Implementation source code

The source code associated with this thesis is publicly available on a GitHub repository to support future research based on the proposed trust management delegation protocol. The repository provides the implementation and experimental materials related to the protocol design, proof-of-concept implementation, and evaluation presented in this thesis.

Repository:

<https://github.com/P-Fongta/Trust-management-delegation-Mechanism>

The implementation and experimental source code in the repository support the main protocol functions described in this work, including service discovery, delegation, trust computation, trust evidence collection, trust data storage, trust data access control, and delegation handover.