A Trust Management Delegation Protocol for Fog Computing Applications

Phongsathon Fongta
Department of Computer and Information Sciences
University of Strathclyde
Glasgow, United Kingdom
phongsathon.fongta@strath.ac.uk

Sotirios Terzis
Department of Computer and Information Sciences
University of Strathclyde
Glasgow, United Kingdom
sotirios.terzis@strath.ac.uk

***Abstract*—Fog computing applications involve collaboration among several devices, end and fog nodes. Trust management can enhance the security and reliability of fog applications by supporting the selection of trusted nodes to collaborate. Although several mechanisms for managing trust in fog computing have been proposed, they have not considered the difference in the ability of devices to perform trust management. Devices with limited resources are unable to manage trust on their own without support from resource-rich devices. We propose a trust management delegation protocol based on the Constrained Application Protocol (CoAP) to address this. Our protocol allows resource-constrained devices to delegate trust management to resource-rich devices that can manage trust on their behalf. Experimental evaluation results show that using our protocol for trust management delegation consumes less resources compared to managing trust locally. Our protocol enables resource-constrained devices to exploit the resources available in fog computing environments to manage their trust.**

Keywords—trust management, internet of things, fog computing, delegation protocol.

I. INTRODUCTION

Fog computing has been developed to overcome cloud computing issues like network congestion, high response latency and location awareness [1]. It consists of heterogeneous devices including routers, switches and other kinds of networking equipment with enhanced computation and storage capabilities acting as fog nodes located in close proximity to users at the edge of the network. End devices with limited resources, particularly Internet of Things (IoT) devices, can offload their heavy tasks and data to the fog nodes for further processing and storage rather than offloading them to cloud data centres. As a result, fog computing can provide lower network latency and decrease application response time and can support real-time interactions for time-sensitive applications that could not be supported by cloud computing [2].

However, fog computing raises concerns about the trustworthiness of different devices, end and fog nodes. To ensure the security and reliability of systems in fog computing, it is important for all devices to avoid interactions with misbehaving and malicious devices, and to isolate them from the system. Trust management can help devices select interaction partners that will be well behaved and provide good services. Trust management monitors the behaviour of devices in a system and determines their trustworthiness in particular contexts.

Several researchers have proposed trust and reputation models to manage trust in fog computing environments [3], [4], [5]. However, these models tend to assume a single trust management architecture with all devices in a fog system treated the same without consideration of the differences in their computational and storage capabilities. This is in contrast to the reality of fog computing [6], [7] where devices have a wide range of capabilities. So, although some devices have the resources to manage trust on their own, others do not.

In fog computing, devices like fog nodes with high computational and storage capabilities could make their resources available to resource limited devices allowing them to overcome resource constraints and manage their trust. To enable this, we propose a trust management delegation protocol for fog computing. The protocol allows resource-constrained devices to delegate trust management to resource-rich devices in a fog computing environment. Thus, managing trust while reducing their resource consumption and preserving their resources.

More specifically, our contributions are:

- The design of a trust management delegation protocol on top of the CoAP that uses self-describing messages to allow delegates (resource-rich devices) to offer delegation services to delegators (resource-constrained devices). The protocol uses a trust discovery server to allow delegators to find delegates that meet their trust management requirements. The protocol supports both pull and push discovery of trust discovery servers using CoAP multicast groups. Leasing is used when offering delegation services to deal with device volatility.
- An implementation and experimental evaluation of the protocol. The protocol is implemented in Python using a Raspberry Pi Zero to act as an end node, and a Raspberry Pi desktop running on Oracle VirtualBox to simulate fog nodes. Using this setup, we have compared the CPU time, RAM, HDD space and energy used by an end node when delegating trust management to a fog node against that used by an end node when performing trust management locally.

Our results show that delegation consumes less CPU time, HDD space and energy, with the difference becoming more prominent when managing trust for multiple devices, while RAM remains stable but higher when trust is managed for a small number of devices, a limitation of the CoAP implementation used. Overall, our protocol allows resource-constrained devices like end nodes

to overcome their resource limitation by delegating trust to resource-rich devices like fog nodes.

The rest of this paper is organised as follows: related work is reviewed in Section 2; our trust management delegation protocol is presented in Section 3; the experimental evaluation of the delegation protocol is presented in Section 4; finally, the conclusion and future work are discussed in Section 5.

II. RELATED WORK

This section introduces trust management terminology used in this paper and summarises recent work on trust management in IoT, fog computing, and cloud environments.

Trust refers to the relationship between two parties and the beliefs of one party towards the behaviour of the other. Trust is related not only to security, but also several other aspects of an entity, such as reliability, availability, etc. Trust management is about the mechanisms that allow entities to build trust with one another [8]. In computing systems, devices may cooperate to perform tasks or pool their resources to handle resource-intensive tasks. These devices must be trusted. However, the trustworthiness of a device may change over time, as a device may work in a satisfactory manner only from time to time. Therefore, trust management monitors devices and allows trust establishment between two devices, trustor and trustee. The trustor is the device who needs to assess the trustworthiness of the trustee. The trustee wants to have high trust values to be selected as a collaborator. So, trust management allows the trustor to select a trustee who will provide good service and be well-behaved relying on its future behaviour, even when it is uncertain.

More specifically, the trustor collects evidence about a trustee by monitoring its behaviour. The evidence is used to update the trust value for the trustee. There are two schemes for updating the trust values: event-driven and time-driven. In the former, the trust value is updated when new evidence becomes available. In the latter, trust values are updated periodically. In both cases, a trust calculation scheme is needed to combine the available evidence into the new trust value. Trust values are then used to make decisions on whether to collaborate with trustees. Decisions are made either by comparing the trust value against a set threshold, with trustees above the threshold considered trusted, or by selecting the trustee with the highest trust value from a set of candidates.

Trust management approaches that have been proposed in the fog computing area take into account the trust of both fog nodes, end nodes, and cloud data centres. Rahman et al. [9] proposed a fuzzy logic based trust management approach to estimate the trustworthiness of fog nodes. An end device with limited resources offloads its tasks to fog nodes based on their trustworthiness. The end node uses Quality of Service (QoS) parameters, including distance, latency, and reliability, as trust metrics to identify bad fog nodes that may provide poor quality of service or behave maliciously. In fog computing, once end nodes get connected to fog nodes for services, they may conceal malicious scripts or harmful code in offloaded data, leading to damaging effects on the fog nodes. An approach for dealing with the trust of fog nodes towards end nodes was proposed in [10]. The authors present a multi-layer architecture for trust management called FogTrust that assesses devices in the end layer before their data is transmitted to the fog layer. The fog nodes use both QoS and social trust aspects, namely availability, honesty, and cooperativeness, to identify malicious end nodes.

In contrast, the authors of [11] pointed out that trust should be considered as bi-directional, end nodes towards fog nodes and vice versa. So, they present trust management that allows end nodes and fog nodes to estimate the trustworthiness of each other in fog computing systems. Ramamurthy et al. [12] also considered bi-directional trust relationships and both QoS and social trust information for trust evaluation 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 the fog client. Wang et al. [13] considered trust between cloud nodes and end nodes to ensure that cloud nodes meet the requirements of end nodes and end nodes are well-behaved before they access cloud services. A particular issue with all these approaches is that they have not considered the differences in the capabilities of the devices and their ability to perform trust management.

This issue is further accentuated when one considers the different algorithms for trust aggregation. Although the weighted sum is a widely used technique that assigns a weight to each trust metric, many studies have proposed trust management systems based on complicated trust computation models. In [14], fuzzy logic is used in evaluating the trustworthiness of devices in fog computing systems. Random forest regression, a machine learning method for classification and regression, is used in [15] to predict the trust of a fog node based on QoS parameters, while in [16], trust evaluation is formulated as a multiple linear regression problem. In [12], trust is evaluated using logistic regression by the fog server towards fog clients, while subjective logic is used by the fog clients to estimate the trustworthiness of the fog servers. These trust management approaches have been shown to be effective at identifying and isolating malicious nodes. However, a significant drawback of them is that they are considered resource-intensive for devices with limited resources, especially end devices in fog computing systems. End devices are unable to use them due to their limited storage and computation capabilities.

Blockchain technology has also been suggested for trust management in fog computing systems [17]. The blockchain provides a decentralised repository for storing trust information of end nodes. Although using a blockchain effectively addresses their storage limitations, it does not address their limited computational capabilities.

For resource-constrained devices to take full advantage of trust management, they need to be able to utilise the resources available in fog computing environments, in fog nodes and cloud servers. So, a mechanism is needed for such devices to delegate trust management to resource rich devices.

III. PROPOSED TRUST MANAGEMENT DELEGATION PROTOCOL

In this section, we describe in detail a trust management delegation protocol allowing resource-constrained devices to

delegate trust management to resource rich devices in fog computing systems.

A. Trust Delegation Components

There are 4 components involved in trust management delegation:

- A delegate is a device providing trust management services to resource-constrained devices.
- A delegator is a device that needs to delegate trust management tasks to a delegate.
- A Trust Service Server (TSS) is a device that is responsible for trust management service registration, deregistration and delegate lookup.
- A delegation protocol is a set of functions that devices in a trust system can use to communicate with each other to delegate trust management.

B. Delegation Protocol Design and Implementation

We have designed the trust management delegation protocol over CoAP, a lightweight application protocol for message exchange and data transmission between devices defined in RFC 7252. Although CoAP runs over UDP which does not guarantee the reliable message delivery, it has smaller overheads than TCP making it suitable for fog computing environments. We use key value pairs to represent all message parameters making protocol messages self-describing. To reduce network overheads, we piggy-back response data in acknowledgement messages.

Our protocol is implemented in Python using the aiocoap library to implement CoAP. We use JavaScript Object Notation (JSON) to represent data in the payload for both request and response messages. SQLite is used to store trust value, trust evidence, delegation information, and relevant data.

C. Delegation Service Registration and Deregistration

A delegate that provides trust management delegation services must register a delegation description with a TSS. The TSS constantly listens to registration requests sent by delegates. A POST message with URI-Path */REG_Request* is used for registration. The message payload includes information about the delegation service, including the delegate's IPv6 address, delegation time, trust model for trust calculation, a set of trust evidence for evidence collection and storage, and decision-making method. On successful registration, the TSS sends an acknowledgement to the delegate with a piggyback response code *2.01 Created* and an 8-bit registration reference. On failure, it replies with response code *5.00 Internal Server Error*.

To deal with device volatility, we use leasing for the provision of delegation services. Delegates register their delegation services for a lease period. They can renew the lease before it expires to extend the availability of the services. The TSS will withdraw all delegation services provided by a delegate if the lease time expires.

The delegate may decide to stop offering some delegation services for a variety of reasons, e.g. running out of resources. The delegate submits a DELETE message to the TSS with URI-Path */DEREG_Request* with the registration reference number and the delegation services it needs to deregister in the payload. Upon receipt of the request, the TSS deletes the delegation services from the database and returns an acknowledgement message with a response code of *2.05 Deleted* to indicate successful deregistration. However, the delegate continues offering delegation services to delegators that have requested them before the withdrawal until their lease duration expires.

D. Delegate Lookup

Once the delegate has completed the registration process, the delegation services are available to be looked up by delegators. A delegator performs a TSS look-up to find a delegate that meets its delegation requirements. When a delegate is identified, the delegator can establish direct communication with it. The delegator sends a GET message to the TSS with URI-Path */Delegate_Lookup*. The message payload contains the delegate's specification, which includes delegation roles, a trust model for trust calculation, a set of trust evidence for evidence collection and storage, and a decision-making method. The TSS finds registered delegates that match the specifications in the database. The TSS gathers them and replies to the delegator with an acknowledgement message with a response code *2.05 Content* and the IPv6 addresses and trust management roles supported by the delegates in the payload. If no delegates that meet the required specification are found, the TSS sends a response code *4.04 Not Found* with a message *No delegates found*.

E. TSS Discovery

TSS discovery is used to find a TSS by a delegate and a delegator whenever there is no known discovery service, or the existing service has expired, or doesn't respond. There are two types of TSS discovery supported: pull and push. In the former, delegates and delegators broadcast a message to find the TSS, while in the latter the TSS broadcasts information about its availability.

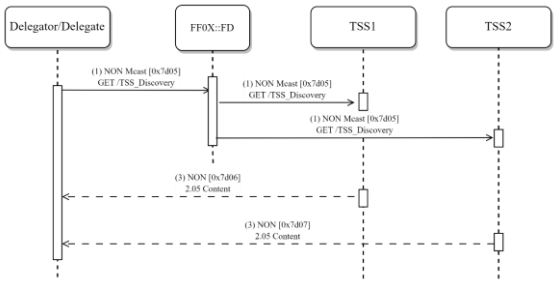

Fig. 1. Pull TSS Discovery

1) Pull: The delegator or delegate sends a non-confirmable GET request to the all CoAP nodes multicast address *FF0X::FD*. The request has the URI-path */TSS_Discovery*. All TSS must join a multicast group in order to receive packets sent to that group. Delegators and delegates do not need to be part of that group to send the request. Any TSS that receives the request will reply with a non-confirmable *2.05 Content* response. The response payload contains the TSS IPv6 address. The requester may receive multiple responses from different

TSS devices. However, it only accepts the first response message and ignores the others. The process is shown in Fig 1.

2) Push: As shown in Fig. 2, TSS advertises their availability with a PUT non-confirmation broadcast message with the URI-path */TSS_distribution*. For delegators and delegates to receive the broadcast message, they must join the multicast group. They can dynamically join and leave the multicast group at any time. The TSS broadcasts the message with a 3-second interval for 15 seconds and a 15 second backoff. The purpose of the backoff is to avoid network congestion and packet loss and ensure network stability. Delegators and delegates that join the multicast group will receive the request and can contact the TSS directly to use its services.

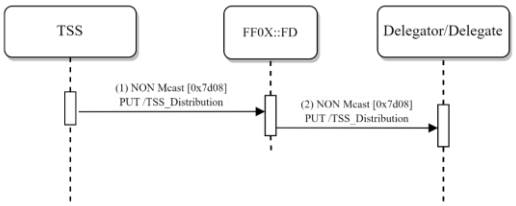

Fig. 2. Push TSS Discovery

F. Delegating Trust Management to a Delegate

As shown in Fig. 3, delegating trust management to a delegate involves the following message exchanges:

1) The delegator sends a delegation request

After looking up delegates from a TSS, the delegator sends the request for trust management delegation to the selected delegate using a POST request with URI-Path */Delegate_All_Roles.* The payload includes the IPv6 address of the delegator and the trustees to be monitored, the delegation lease period, and the details of the trust model to be used (i.e. trust calculation and update schemes, and a set of trust evidence used), and an indicator for sharing recommendations (i.e. trust values) with others. Our delegation approach supports delegating multiple trustees in a single request provided the same trust model is used for all of them.

The delegate saves the delegation information in its database, and sends to the delegator a response code *2.01 Created* piggybacked in an acknowledgement message if the delegation is successful, or *5.00 Internal Server Error* if not. When the delegation is successful, the delegate starts monitoring the delegated trustees, collecting evidence about them in its database, calculating and updating their trust values.

2) The delegator sends trust evidence to the delegate

Some kinds of trust evidence can only be collected by the delegator, e.g. response time. For this evidence, the delegator must collect it, and pass it on to the delegate. The delegator attaches the evidence in a POST message with the URI-Path */Evidence_Storage*. To simplify the processing of the evidence at the delegate, each message provides evidence for a single trustee. However, the delegate is implemented as a multi-threaded server with each thread dedicated to evidence collection for a single trustee. Upon receipt of the evidence, the delegate stores it in its database and replies with an acknowledgement with a response code *2.01 Created* if the evidence is successfully stored.

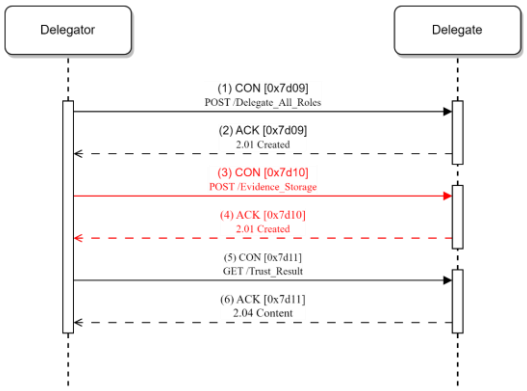

Fig. 3. The Process of Trust Management Delegation

3) The delegator queries for a trust result

To get trusted trustees that it can work with the delegator uses a GET request and with */Trust_Result* in the URI-Path. The payload contains the IPv6 address of the delegator and either one trustee and a threshold value for trust decision-making, or a list of trustees with a blank threshold. In both cases, the delegate replies with an acknowledgement message with a response code *2.05 Content* and the trust result. In the case of a single trustee, the trust result is either *trusted* or *untrusted*. In the case of multiple trustees, the delegate selects the trustee with the highest trust value, and returns to the delegator its IPv6 address.

G. Trust Management Delegation Protocol Security

Trust management systems can be vulnerable to attacks, such as bad-mouthing attack, on-off attacks, etc, that aim to manipulate the trustworthiness of entities and influence their chances of being selected as collaborators. Trust computational models can be designed to address such attacks, e.g. [19] protects against bad-mouthing attacks, while [18] against on-off attacks. The proposed delegation protocol can support such trust computational models and thus protect against such attacks.

However, the delegation protocol can be vulnerable to misbehaving and malicious devices that play its various protocol roles, like TSS, delegates, delegators and evidence collectors. Such devices can exploit protocol messages to disrupt the correct operation of delegation partners and manipulate trust management. Such devices can be detected and isolated by expanding trust management to cover the trustworthiness of all delegation roles, in a way that is like how trust in trustees is managed.

IV. Experimental Results

In this section, we describe the setup and results of an experimental evaluation of our trust delegation protocol. We compare the amount of resources (CPU time, RAM, HDD space, and energy), spent by an end node handling trust management locally using a weighted sum trust model, against those spent by an end node that delegates trust management with the same trust model to a fog node using our delegation protocol.

A. *Experimental setup*

We use a Raspberry Pi Zero to act as an end node, and a Raspberry Pi desktop running on Oracle VirtualBox to simulate fog nodes. The Raspberry Pi desktop runs a TSS and a delegate on separate virtual machines. CPU time, RAM and HDD space are measured by a Python script that operates independently from the trust management mechanism. To measure energy usage, we employ an INA 219 module to read DC voltage and current and calculate energy consumption.

TABLE I. EXPERIMENT PARAMETERS

Parameters	Value
Number of fog nodes	1-10 nodes
Trust calculation model	Weighted sum
Trust evidence	Distance, Availability Response time
Trust update	Event-based
Decision-making type	Threshold

The parameters of the experiment are shown in Table 1. In our experiment, we consider that devices will be relatively localised and only interact with a relatively small number of fog nodes within a particularly physical area. So, we range the number of fog nodes (trustees) between 1 and 10. When trust management is done locally, the end node (trustor) collects and locally stores trust evidence for the trustee(s); calculates and updates the trust value(s); and makes decision(s). When delegating trust management, the delegator (end node and trustor) does the following: discovers a TSS using the pull approach; looks-up for a delegate in the discovered TSS, requests from the found delegate to perform trust management on its behalf for the fog node(s) (trustees), collecting and storing evidence, calculating and updating the trust value(s), and decision making (trust result queries). In both cases (local and delegated), the process are run 10 times.

The trust model in this experiment uses three trust metrics: distance, availability and response time. Distance is defined as the number of network hops between the end node and a fog node. Availability indicates that the fog node is accessible and operational. It is the ratio of the number of ICMP echo reply messages to the total number of ICMP echo request messages. Response time is the time interval between when a message is sent and a response is received from the fog node. These trust metrics are aggregated using a weighted sum as follows:

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

Where $W_i = \{w1,w2,w3,\ldots\}$ are static weighting factors for trust metrics X_i and $\sum_{i=1}^{n} W_i = 1$.

We repeat the experiment 10 times and calculate a 95% confident interval for the measurements to show the uncertainty associated with them.

B. *Results and Discussion*

Fig. 4 shows a comparison of CPU time spent by the end node to perform trust management locally and using our trust management delegation protocol. The horizontal axis represents the number of fog nodes, while the vertical axis shows CPU time in seconds. Overall, as one would expect our trust management delegation protocol helps the end node to reduce CPU time for trust management. At the same time, there is a rise in CPU time when the number of fog nodes (trustees) increases in both approaches. However, CPU time for performing trust management locally grows more sharply than when delegation is used. With 1 fog node, the CPU time to delegate trust management is lower than performing trust management locally by 10.75%, with 1.12 and 1.26 seconds, respectively. While with 10 fog nodes, the corresponding numbers are 2.21 seconds and 6.89 seconds, which is 67.92% decrease.

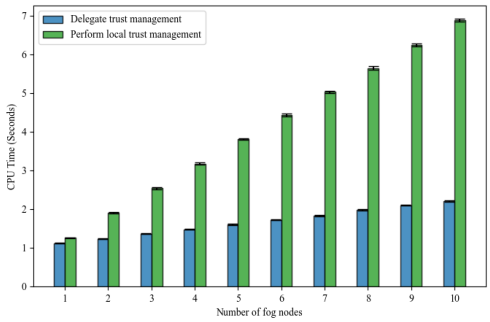

Fig. 4. CPU Time of Delegating Trust Management and Performing Trust Management Locally

The amount of RAM in MB used by the end node is shown in Fig. 5. The end node needs more RAM to perform trust management delegation. This is because the aiocoap library uses around 5 MB on the end node. However, RAM usage when delegating trust remains almost constant at around 18.5 MB as the number of fog nodes increases. In contrast, RAM usage when performing trust management locally increases as the number of fog nodes rises from 14.92 MB for 1 fog node to 15.94 MB for 10 fog nodes, a 6.84% increase. As a result, we expect that it will eventually overcome the amount of RAM used when delegating trust management.

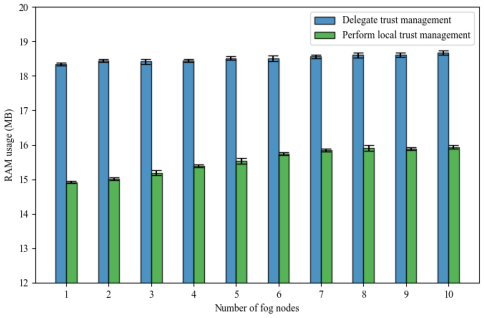

Fig. 5. RAM of Delegating Trust Management and Performing Trust Management Locally

As one would expect, delegating trust management reduces the amount of HDD space used for storage of trust evidence and trust values, as shown in Fig. 6. Delegating trust management allocates a fixed amount of HDD space, 20 KB. In contrast, when performing trust management locally, the HDD space rises as the number of fog nodes increases, from 32 KB for 1 fog node to 68 KB for 10 fog nodes.

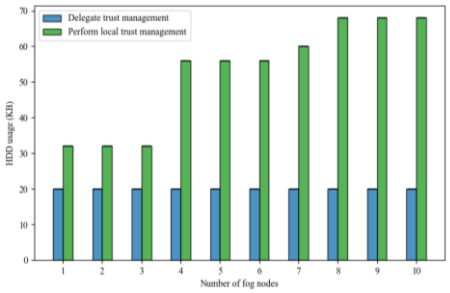

Fig. 6. HDD Space of Delegating Trust Management and Performing Trust Management Locally

The total energy consumption in milliwatt-hour (mWh) consumed by the end node is shown in Fig. 7. Note that in contrast to the earlier figures confidence intervals in Fig. 7 are much wider, reflecting the higher variance in the measurements. Both approaches have an upward trend in energy consumption as the number of fog nodes increases, but the energy consumption of delegating trust management goes up slower. Moreover, energy consumption when delegating trust management is lower by 9.20% for 1 fog node, 5.43 mWh versus 5.98 mWh, and 21.42% lower for 10 fog nodes, 5.76 mWh versus 7.33 mWh of energy. It is interesting to note that Fig. 7 shows similar trends to the CPU time in Fig. 4. This is because the more CPU time used, the more energy is consumed.

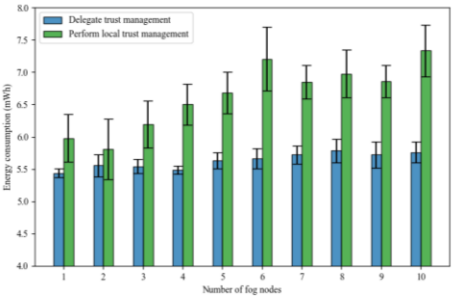

Fig. 7. Energy Consumption of Delegating Trust Management and Performing Trust Management Locally

In conclusion, we can see that trust management delegation offers clear benefits for resource-constrained end nodes, allowing them to take advantage of fog node resource to keep their trust management resource consumption low.

V. Conclusion

In this paper, we have proposed a trust management delegation protocol to accommodate the differences in capabilities between devices in fog computing environments. The protocol enables resource-constrained devices, like end nodes, to delegate management of trust to resource-rich devices, like fog node, in order to reduce resource consumption and preserve available resources for their main tasks. The evaluation of our protocol shows that trust management delegation clearly reduces resource consumption in terms of CPU time, HDD space, and energy, compared to performing trust management locally, while it keeps RAM usage stable as trustee increase.

In future work, we will focus on extending our delegation protocol to support delegating trust management to multiple delegates and transfer of delegations. Using multiple delegates will help to better balance delegation workload among delegates, while transfer of delegations will support mobile end nodes as they move through the fog computing infrastructure.

References

[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.

[2] S. Yi, C. Li, and Q. Li, “A Survey of Fog Computing,” Proceedings of the 2015 Workshop on Mobile Big Data - Mobidata ’15, pp. 37–42, 2015.

[3] 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.

[4] Y. Hussain et al., “Context-Aware Trust and Reputation Model for Fog-Based IoT,” IEEE Access, vol. 8, no. September 2019, pp. 31622–31632, 2020.

[5] M. Al-khafajiy et al., “COMITMENT: A Fog Computing Trust Management Approach,” J Parallel Distrib Comput, vol. 137, pp. 1–16, 2020.

[6] D. Shehada, A. Gawanmeh, C. Y. Yeun, and M. Jamal Zemerly, “Fog-based distributed trust and reputation management system for internet of things,” Journal of King Saud University - Computer and Information Sciences, vol. 34, no. 10, pp. 8637–8646, Nov. 2022.

[7] M. Aaqib, A. Ali, L. Chen, and O. Nibouche, “IoT trust and reputation: a survey and taxonomy,” Journal of Cloud Computing, vol. 12, no. 1. Springer Science and Business Media Deutschland GmbH, Dec. 01, 2023.

[8] 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 Things J, vol. 10, no. 12, pp. 10575–10586, Jun. 2023.

[9] F. H. Rahman, T. W. Au, S. H. Shah Newaz, and W. S. Suhaili, “Trustworthiness in fog: A fuzzy approach,” ACM International Conference Proceeding Series, pp. 207–211, 2017.

[10] E. Wong et al., “FogTrust: Fog-Integrated Multi-Leveled Trust Management Mechanism for Internet of Things,” 2023.

[11] 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.

[12] 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.

[13] 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, 2018.

[14] F. H. Rahman, T. W. Au, S. H. Shah Newaz, and W. S. Suhaili, “Trustworthiness in fog: A fuzzy approach,” ACM International Conference Proceeding Series, pp. 207–211, 2017.

[15] A. K. Junejo, N. Komninos, M. Sathiyanarayanan, and B. S. Chowdhry, “Trustee: A Trust Management System for Fog-enabled Cyber Physical Systems,” IEEE Trans Emerg Top Comput, pp. 1–12, 2019.

[16] 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.

[17] S. Hameed et al., “A Scalable Key and Trust Management Solution for IoT Sensors Using SDN and Blockchain Technology,” IEEE Sens J, vol. 21, no. 6, pp. 8716–8733, 2021.

[18] S. Kannan, R. Venkataraman, and G. S. Ramachandran, “On-off attack detection in trust model using intra-daily variability for the IoT,” Bulletin of Electrical Engineering and Informatics, vol. 12, no. 6, pp. 3880–3888, Dec. 2023.

[19] 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), 2019, pp. 278–282.