Evaluating Security and Performance of CoAP Protocol on Raspberry Pi 4 for IoT Applications

SHIVA BIERSTEKER, University of Twente, The Netherlands

The CoAP (Constrained Application Protocol) protocol is increasingly utilized in IoT applications due to its lightweight and RESTful communication model. This research project aims to assess the security and performance of the CoAP protocol on the Raspberry Pi 4, which is widely deployed as IoT endpoints. The study employs rigorous performance metrics including CPU utilization, memory usage, and latency, coupled with a thorough security assessment focusing on network-level vulnerabilities. Findings are attained through theoretical and practical means. The research outcomes will provide insights into the suitability of CoAP for IoT deployments on the Raspberry Pi 4 and offer recommendations for enhancing both security and performance.

CCS Concepts: • **Security and privacy → Security protocols**.

Additional Key Words and Phrases: CoAP, IoT, Internet of Things, Security Protocols, Evaluation, Raspberry Pi, Performance.

1 INTRODUCTION

The Internet of Things (IoT) is ever-growing. Many households and companies are adding IoT devices to their network [2]. It is important to hold these devices to a high-security standard, as they are a common target of attacks. There are several protocols for IoT applications for constrained devices, but not all of them are well-documented concerning their security features and performance. One of these protocols with lackluster documentation is the CoAP protocol.

The Constrained Application Protocol (CoAP) is a specialized web transfer protocol for use with constrained nodes and constrained networks [7], and the main focus of this research paper. This means it has to function under energy constraints, memory limitations, limited processing capability, high latency, unreliable networks with high packet loss, and being left unattended. CoAP is thus designed for devices typically found as IoT endpoints, whose main intercommunication is sending and receiving sensor data. CoAP also allows for seamless integration with HTTP, enabling easy connection with the internet and external, full-size devices like a server. Naturally, this HTTP communication takes a larger toll on resources, so an endpoint device is used, like a Raspberry Pi. This way, all the small sensor devices can communicate with the much more powerful Raspberry Pi, and the Raspberry Pi can then process this data or send it off using CoAP to HTTP communication. Although there have been papers in the past about the CoAP protocol and it's security features, none of them have specifically touched upon the CoAP protocol when implemented on the Raspberry Pi 4. Analyzing a protocol when configured on a specific piece of hardware will provide more accurate data for that hardware for real world applications, compared to what a virtual environment can provide. The same argument is even more applicable to performance benchmarks, as measuring things like resource utilization, message transmission efficiency, scalability on Raspberry Pi hardware and power consumption can only truly be analyzed by testing the protocol on the hardware itself. This brings this proposal to the following research questions:

(1) What are security strengths and vulnerabilities associated with the implementation of CoAP on the Raspberry Pi 4 in an IoT context?
(2) What are the performance metrics of the CoAP protocol on the Raspberry Pi 4 in an IoT context?
(3) How do existing studies evaluate the performance characteristics of CoAP protocol implementations on the Raspberry Pi 4?

This research project will specifically evaluate the security features of the CoAP protocol such as authentications, access control, and message integrity when implemented on the Raspberry Pi 4. Analyzing this on physical hardware comes with a higher accuracy than when it is done through a virtual environment, which contributes to any data that is already out there. Furthermore, the project will analyze the performance of the CoAP protocol on the Raspberry Pi 4 by measuring the message transmission efficiency, resource utilization, and scalability of this specific hardware. Since both the Raspberry Pi 4 and the CoAP protocol are commonly found in IoT applications, an analysis of the performance of this combination is important when making decisions about the design of an IoT network.

The rest of this paper is organized as follows: Section 2 will present what is done in the literature regarding the implementation of CoAP in Raspberry PI and how that impact the performance and security of the protocol. In section 3, the methodology to answer the research questions is discussed, along with the hardware and software setup to create the testbed for performance and security analysis. This section will also discuss the way everything is set up. Section 4 will perform a security analysis of CoAP. Section 5 will perform a performance analysis of CoAP on the Raspberry Pi 4. Finally, section 6 will discuss these findings and reach a conclusion.

2 RELATED WORK

Due to the common nature of CoAP, naturally there has been research done on it in the past. One of these is a performance analysis on CoAP done by Azeez and Abdullah [1]. This paper focused on communication between MQTT, CoAP and HTTP and looked at metrics such as error rate, message throughput and elapsed time. However, this study did not look at a CoAP implementation on the Raspberry Pi. Naik [5] did a comparative analysis between MQTT, CoAP, AMQP and HTTP. This paper provides an abstract comparison between these protocols and compares their strengths and weaknesses, but there are no concrete numbers regarding the performance of the CoAP protocol on the Raspberry Pi. Guaman et al. [3]

TScIT 41, July 5, 2024, Enschede, The Netherlands

conducted a comparative performance analysis between MQTT and CoAP protocols for IoT applications using Raspberry Pi 3 devices in IEEE 802.11 environments. While their study compared the two protocols, it did not comprehensively evaluate the security aspects of CoAP nor provide insights into its scalability on Raspberry Pi hardware. Rahman and Shah [6] focused on the security analysis of IoT protocols, with a particular emphasis on CoAP. While their study addressed security vulnerabilities and proposed solutions for CoAP, it did not extensively evaluate the protocol's performance or scalability on the Raspberry Pi 4. Kruger and Hancke [4] did a performance analysis of the CoAP protocol on some single-board computers, including the Raspberry Pi 3. The difference between the Raspberry Pi 3 and 4 is large enough that a performance analysis on the Pi 4 adds value. Furthermore, this research has not done a security analysis on Raspberry Pi hardware.

3 PROPOSED SOLUTION

In this section, the methodology to answer the research questions will be discussed. This involves the used hardware, software and the setup.

3.1 Hardware

For this study, the Raspberry Pi 4 Model B Rev 1.5 was the primary hardware platform, but other hardware components were used as well. These are described in table 1.

Table 1. Hardware Components

RPi Processor	Broadcom BCM2711, Quad core Cortex-A72 (ARM v8) 64-bit SoC @ 1.8GHz
RPi Memory	4GB LPDDR4-3200 SDRAM
RPi Storage	Kingston Canvas Select Plus microSDHC 32GB (Class 10)
RPi Ethernet	Gigabit Ethernet
RPi Wireless	2.4 GHz and 5.0 GHz IEEE 802.11ac wireless, Bluetooth 5.0, BLE
RPi Connectivity	2x USB 3.0 ports, 2x USB 2.0 ports, 40-pin GPIO header
RPi Power Supply	5.1V, 3.0A DC via USB-C connector
Power Measurement Tool	AVHzY C3

3.2 Software

Various software was also employed for this research paper to conduct the security and performance analysis for the CoAP protocol on the Raspberry Pi. The details of this software can be seen in table 2.

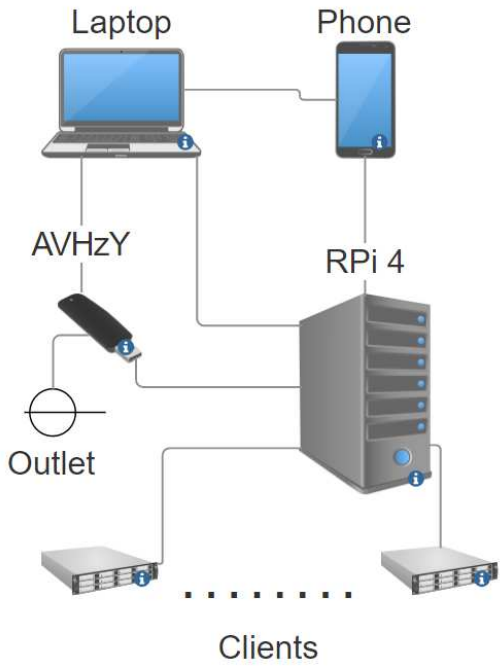

Fig. 1. Test setup

Table 2. Software Components

Operating System	Raspberry Pi OS (formerly Raspbian), Debian 11.5
Kernel Version	Linux 5.15.61-v8+
CoAP Library	aiocoap-0.4.7, installed via pip
Python Version	Python 3.7
Power measuring	Shizuku ToolBox V1.00.20
Wireshark	Version 3.4.10 (Git v3.4.10 packaged as 3.4.10-0+deb11u1)

3.3 Setup

The Raspberry Pi was controlled via an SSH connection through a laptop. A smartphone was used to host a hotspot for the laptop and RPi to connect to. An aiocoap server and client were made, along with a DoS client and a client which times its message delay and exports it to a CSV file to plot the data. A shell script was made to monitor CPU and memory usage over time, to be exported to a CSV file. All the CoAP programs are running on the RPi on localhost. The AVHzY C3 is connected between the RPi power supply and the RPi power port. It also has a connection to the laptop via a USB type A to USB type C cable. The setup can be seen in Figure 1.

4 SECURITY ANALYSIS

The first research question was: What are security strengths and vulnerabilities associated with the implementation of CoAP on the Raspberry Pi 4 in an IoT context? In order to answer this research question, threat modeling needs to be done on the CoAP protocol.

The most important threats then need to be tested in a real life scenario in order to see their impact on the CoAP protocol on the RPi 4.

4.1 Threat Modeling

The CIA Triad will be used to define the security requirements of the CoAP protocol on the Raspberry Pi 4 for IoT applications.

4.1.1 Confidentiality. Confidentiality refers to protecting information from unauthorized access. IoT systems are often located in places where sensitive data is prevalent, and users would typically like to keep this information private. The main threats to confidentiality include:

Sniffing: Unauthorized interception of data as it is transmitted over a network. This can expose sensitive information, such as credentials or personal data. A system is vulnerable to this attack if it lacks encryption for data in transit, uses unsecured communication channels or has inadequate network segmentation. Sniffing may be prevented by implementing strong encryption protocols, using secure network configurations or proper network segmentation and isolation of sensitive data. By default, CoAP transmits data in plaintext, making it vulnerable to interception by attackers. Without encryption mechanisms, any data sent over CoAP can be easily captured and read by eavesdroppers.

Spoofing: An attacker impersonates another device or user to gain unauthorized access to information. A lack of authentication mechanisms and weak or hardcoded credentials make a system susceptible to spoofing attacks. The implementation of certificate-based authentication and strong authentication mechanisms alongside unique credentials, can mitigate the threat of a spoofing attack. CoAP does not implement these mitigation tactics by default, and the use of default credentials is a common fault in setting up devices to be used in IoT systems.

Phishing: Deceptive attempts to acquire sensitive information by masquerading as a trustworthy entity. Phishing may compromise user credentials and access sensitive information. While phishing is a common attack on networks with a lot of human interaction, an IoT network is mainly machine-to-machine communication, making CoAP not very susceptible to phishing attacks.

4.2 Integrity

Integrity means data are trustworthy, complete, and have not been accidentally altered or modified by an unauthorized user. Common attack vectors include:

Data Tampering: is the deliberate or accidental alteration, deletion, or insertion of data without authorization or proper validation. This can lead to corrupted data, which leads to incorrect or misleading information. Without DTLS, CoAP offers no protection against data being altered in transit.

Replay Attacks: This type of attack involves reusing data that has already been transmitted to gain unauthorized access to a system. This may allow the attacker to alter the contents of the data or to re-transmit the data at a later time. Again, insecure communication protocols can leave a network vulnerable to this attack, which is not always a given with CoAP. Implementing session identifiers or timestamps, along with DTLS may provide CoAP with protection against replay attacks.

Message Forgery: Creating fake messages to deceive the recipient. An attacker could manipulate data to achieve unauthorized outcomes. Weak or no message authentication mechanisms and insecure key practices make a system vulnerable to this attack. Base CoAP does not employ cryptographic keys, which leaves it vulnerable to these attacks.

4.3 Availability

Availability means data are accessible when you need them.

Denial of Service (DoS) Attacks: Overloading a system with requests to make it unavailable to legitimate users. This disrupts the service and inability to access necessary resources. Not limiting a client's communication with the server in any way leaves the server vulnerable to DoS attacks. Implementing rate limiters goes a long way.

Relay Attacks: Forwarding of data to another system without authorization. This may cause disruption and misuse of resources. This vulnerability is again related to a lack of authentication. Implementing DTLS on CoAP should make the system far more resistant to these kinds of attacks. Implementing DTLS and OSCORE on CoAP is and very good idea if the network should be secure. Since DTLS and OSCORE are frequently paired with CoAP, there should be plenty of documentation available online to turn CoAP into a secure, lightweight protocol.

4.4 Practical Testing

The setup for the attack consists primarily of the Raspberry Pi. The RPi hosts the server to which the client connects via localhost. This requires the attacks to be run locally on the RPi as well, but the methodologies and results are the same as if the attacker would attack an online server. When the client is connected to the server, the attacks are initiated. The first attack is a sniffing attack. This type of attack is very powerful, as it gives the attacker access to every bit of data that is sent between the client and the server, completely eliminating all confidentiality. For this attack a client sends a secret message to the server via a PUT command, while the attacker uses Wireshark to intercept this packet. Because CoAP doesn't use any encryption, the attacker can freely read the payload data, as seen in figure 2. At the top of the figure, the inspected packet can be seen. This is the PUT packet sent from the client. The bottom of the figure shows the payload data. It is easy to see that the client sent "Super secret message" to the server, as the data is not encrypted. The data sent from the server to the client during a GET request is also easily readable this way. A good mitigation for this attack would be to implement Datagram Transport Layer Security (DTLS) with CoAP. DTLS employs cryptographic keys which encrypt the messages between the client and the server. This should make it computationally unfeasible for an attacker to decrypt an intercepted message. It is however important to note that since CoAP is most

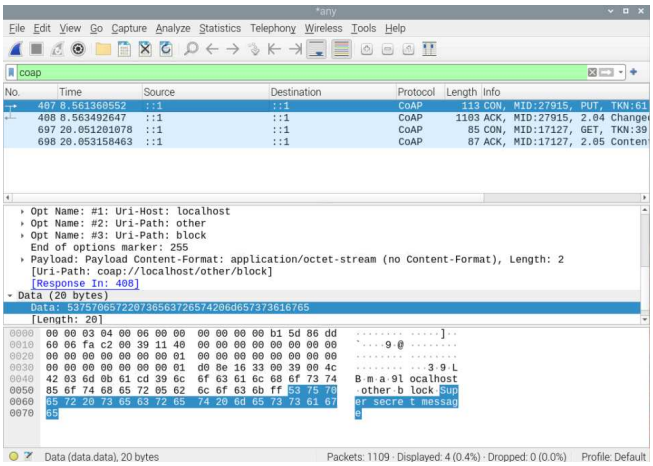

Fig. 2. Sniffing attack intercepted packet

Fig. 3. RPi Power Draw

frequently deployed on resource constrained systems, these cryptographic keys might be unsafe if not implemented properly. This is because the keys are generated using pseudo random functions. The outcome of these functions may be predicted if the generator does not have enough entropy, which is often the case for constrained devices. Using the right software for random number generation on constrained devices could be a cost-effective option, but this should be assessed on a case-by-case basis. The second attack is a Denial of Service (DoS) attack. Here 4 clients would spam GET and PUT commands to the server in an attempt to overload it. This caused the server to run at maximum capacity, and a legitimate client had a larger latency for their communication with the server. The exact performance ramifications will be discussed in the next section. It can thus be seen that CoAP is vulnerable to DoS attacks. Implementing rate limiting for clients would reduce the damage any one client can do. Furthermore, it would be a good idea to find a way to distinguish the device a client comes from, as otherwise it would be easy for an attacker to run multiple clients on one device, essentially bypassing the measures taken by using rate limiting.

5 PERFORMANCE ANALYSIS

In this section the performance of the CoAP protocol on the RPi4 will be discussed. The performance was analyzed using various metrics. These metrics are: power consumption of the RPi, message latency, CPU usage as a package total and per core, and memory usage. Since IoT devices are often restricted in resources, it is important to measure the resource usage of the RPi while running the CoAP protocol in order to get an idea of the weight of this protocol. The CPU measurements will show how much load the protocol will put on the RPi, along with how this load is spread over the multiple cores that are available on the RPi. The memory usage will show the memory efficiency of CoAP. Excessive memory usage significantly compromises the Raspberry Pi's performance, leading to consistent spikes in latency. Since the RPi has limited memory, this is more likely to happen compared to full-size servers. IoT devices may be battery powered, thus making it important that the protocol they are running is energy efficient. For this, power consumption is measured. Latency shows how responsive the system is during various loads. Timely communication is important in certain IoT applications, thus seeing how the latency changes with various loads is good to see. Aiocoap is a Python library and Python does not play nice with multiple cores by default. The tests have been done using base aiocoap for consistency. As per the aiocoap documentation, there are ways to scale the server to multiple cores.

Each of the metrics have been tested in five scenarios. Each scenario consists of one server and a varying amount of clients. The requests from the clients are GET requests, asking the server what the current date and time is. During a DoS attack (DoS), 1 client sending 10 requests per second (1-10), 10 client sending 10 requests per second (10-10), 1 client sending 100 requests per second (1-100), 10 client sending 100 requests per second (10-100). It is important to note that the Raspberry Pi runs on Raspberry Pi OS. As with any OS, there are things going on in the background which causes every metric to have a baseline above 0. All the graphs are measurements over a 30 second time frame. Figure 3 shows the power consumption during the various scenarios. It can be seen that there is only a very small increase in power consumption between 1-10 and 1-100 or 10-10, only about 0.1-0.2 watts. 10-10 is consistently a bit higher than 1-100, potentially indicating that CoAP runs more efficiently with fewer clients. Implementing DTLS will likely further this gap, as this adds per-client overhead. DoS and 10-100 are very similar to each other. This is likely due to these scenarios hitting a limit on the computational power of the RPi. CoAP thus cannot support a thousand requests per second, and should be limited to the order of a hundred requests per second.

Figure 4 shows the round-trip time (RTT) latency of the various scenarios. This is the computational RTT only, as both client and server were hosted on the same device. This means that there were no network delays, like propagation delay, (re)transmission delay or queueing delay. During all these scenarios, three large latency peaks occurred. DoS had a peak at 130ms, 1-100 had a peak at 54ms and 10-100 had a peak at 314ms. These peaks have been omitted from the graph for readability. These peaks indicate that under specific conditions, the system can experience sudden spikes in latency. Furthermore, it can be seen that the latency typically sits between

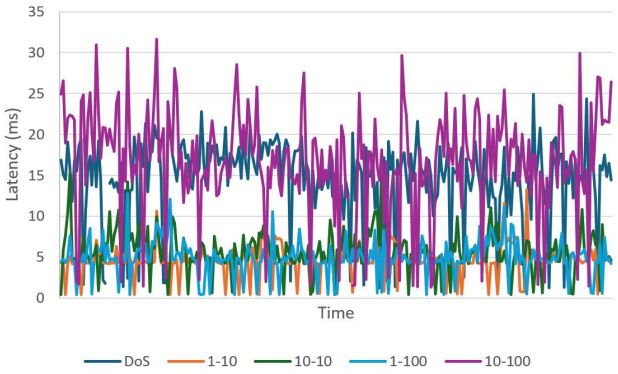

Fig. 4. RPi Latency

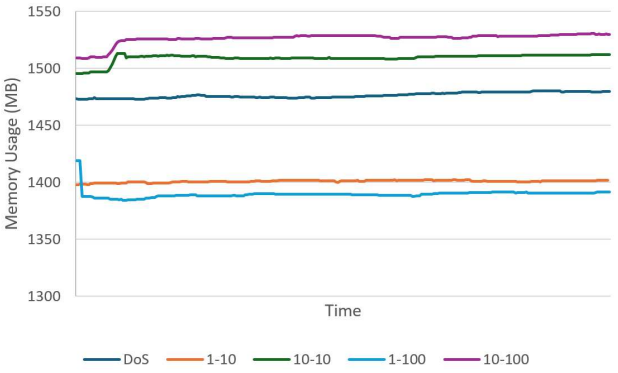

Fig. 5. RPi Memory Usage

0 and 10 ms, averaging out at 5 ms. When the server is throttled by too many messages, this latency increases to about 20ms, and becomes significantly less stable.

Figure 5 shows the memory usage of the RPi under the different scenarios. Here the memory usage seems to predominantly scale with client count. This is likely due to the fact that every client needs its own terminal instance to run. The memory usage is also very consistent, showing that the CoAP protocol doesn't really fluctuate in memory usage over sustained loads.

Figure 6 shows the package total CPU usage of the RPi under the different scenarios. This again shows that CoAP predominantly scales with the total request frequency, and not really with the number of clients. The small discrepancy can be explained by the RPi having to handle multiple terminals for the clients.

Figures 7, 8, 9, 10 and 11 show individual core load under each of the scenarios. Due to the implementation choice of aiocoap, the server may only run on one physical core at a time. Figure 7 clearly shows the server is running on CPU 3, and is capped out at 100% the whole time. Figure 11 shows the server swapping what physical core it is running on, with one core always standing above the rest at 100% usage. Since the clients are running in their own terminal, the operating system can distribute these loads evenly over the physical cores. This can be seen by all the cores running at essentially the

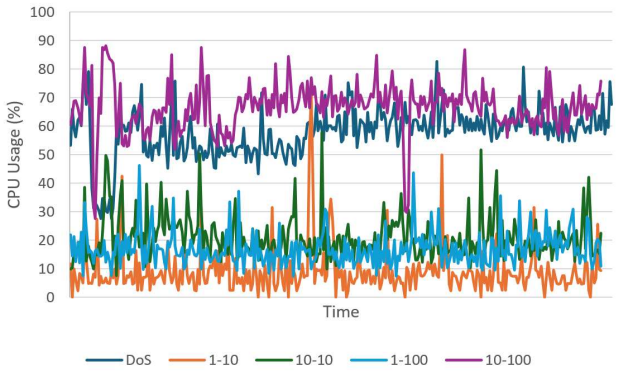

Fig. 6. RPi CPU Package Total

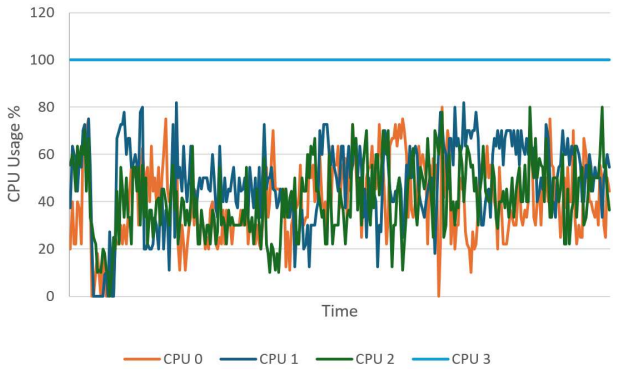

Fig. 7. RPi CPU During DoS Attack

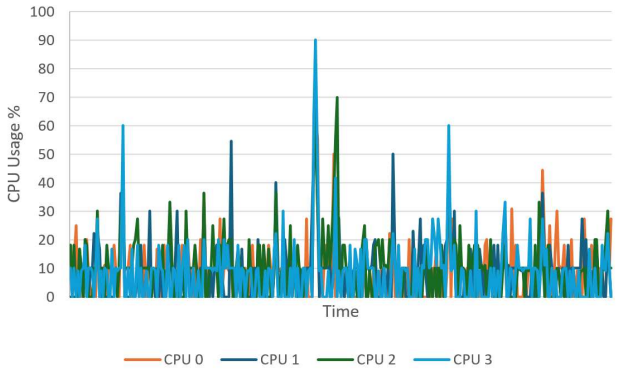

Fig. 8. RPi CPU with 1 client sending 10 requests per second

same speed, with the exception for the server's core. CoAP may be further optimized by using different implementations of the protocol, such as Californium.

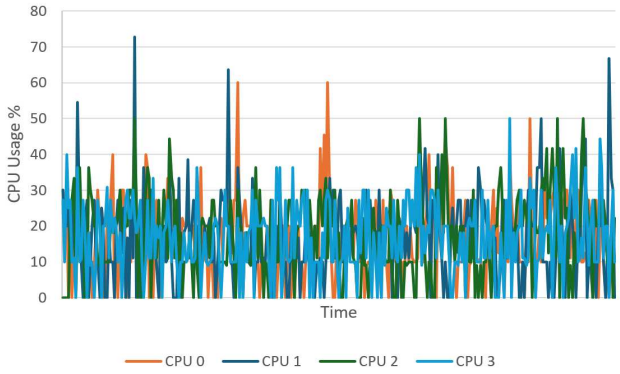

Fig. 9. RPi CPU with 1 client sending 100 requests per second

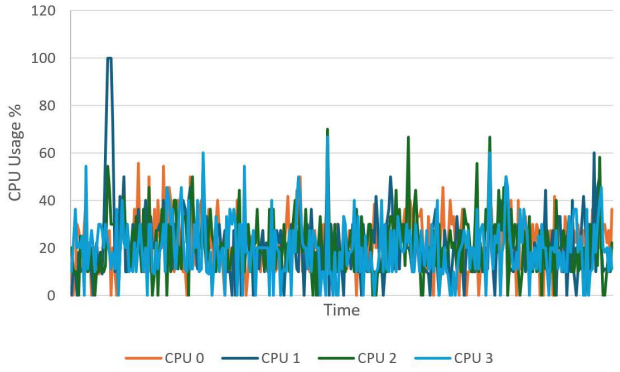

Fig. 10. RPi CPU with 10 clients sending 10 requests per second

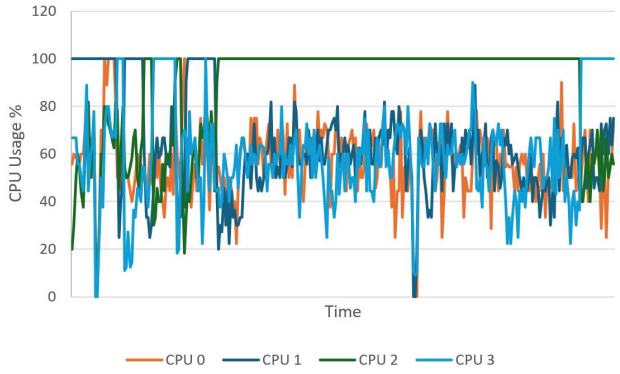

Fig. 11. RPi CPU with 10 clients sending 100 requests per second

6 CONCLUSION

The Internet of Things (IoT) is characterized by a multitude of connected devices that communicate data regularly. These devices are often constrained in terms of computational power and memory, operating within limited network environments. This makes providing these devices with sufficient security and performance a difficult task. In order to succeed at this, research into how CoAP performs at a baseline is necessary. To gain a deeper understanding of these challenges, this study conducted a performance and security analysis of CoAP on the RPi 4, an essential component in many IoT systems.

The security analysis highlighted several vulnerabilities in CoAP when implemented on the RPi 4 and suggested mitigations. By default, CoAP transmits data in plaintext, exposing it to eavesdropping attacks. The adoption of Datagram Transport Layer Security (DTLS) is essential to encrypt CoAP communications and ensure data confidentiality. CoAP does not employ authentication mechanisms, making it susceptible to spoofing attacks. Implementing robust authentication methods, such as certificate-based authentication, can significantly mitigate this risk. CoAP's lightweight nature makes it vulnerable to DoS attacks, where the system can be overwhelmed by a flood of requests. Strategies like rate limiting and traffic differentiation are crucial to protect against such attacks. Practical tests using Wireshark and custom scripts underscored these vulnerabilities, emphasizing the importance of integrating DTLS and implementing rate limiting to enhance CoAP security on the RPi 4. Integrating DTLS into CoAP deployments is essential for addressing confidentiality and integrity issues. DTLS provides encryption and message integrity, protecting against many attacks, e.g. sniffing and tampering. CoAP is often implemented with DTLS and OSCORE. This common combination should make it relatively easy to aid CoAP into being a secure, lightweight protocol. Utilizing stronger authentication mechanisms, such as certificate-based authentication, can significantly reduce the risk of spoofing attacks. Ensuring that only authenticated devices can communicate within the IoT network is crucial for maintaining security. To combat DoS attacks, implementing rate limiting can help manage the flow of requests and protect the system from being overwhelmed. CoAP has many ways to improve its security, while leaving the option to avoid the security overhead for applications which don't need it.

The performance analysis looked at power consumption, CPU utilization, memory usage and latency during various scenarios. CoAP maintained low latency and high efficiency under moderate load conditions, which is suitable for many IoT applications. However, latency increased with larger payload sizes and higher message rates, potentially impacting real-time applications. The RPi 4, equipped with a quad-core processor and 4GB of RAM, effectively handled a load in the order of hundreds or requests per second. By having the RPi run the server only, and enabling multi core support for aiocoap, the amount of traffic it can handle should increase drastically, likely quadrupling. This will be sufficient for most IoT applications where one would deploy a RPi as an endpoint. Deploying Raspberry Pi 4 devices as CoAP servers in sensor networks allows for efficient and responsive data collection. Low latency under moderate loads ensures timely transmission of sensor data, which is crucial for monitoring environmental conditions, industrial processes, or smart agriculture.

6.1 Limitations

This paper only tackles the bare-bones implementation of CoAP on the Raspberry Pi 4. This means there are no results for either security

or performance when CoAP is combined with a wide variety of security measures and performance optimizations. The bare-bones setup did not allow for communication over a network, which means no analysis was done regarding the various network environments the CoAP protocol may be found in.

6.2 Future Work

Future research should focus on analyzing the CoAP protocol with the most common security measures in place, in order to find any more potential weaknesses. These added security measures will also affect the performance of the CoAP protocol, thus giving new insight. Westphall et al. [8] provides a great start to this, as it already compares the impact of different implementations of DTLS with CoAP on performance on the Raspberry Pi 3. Additionally, further research could try to run two Raspberry Pi's simultaneously, connected via a short Ethernet cable. This would allow for an individual analysis of client and server performance, while not deviating too much from the environment achieved by running one Raspberry Pi device.

In conclusion, CoAP offers a robust framework for IoT communications on the RPi 4. However, enhancements in security and performance are necessary to fully leverage its potential. Implementing the recommendations outlined in this study will pave the way for more secure and efficient IoT networks, making CoAP a more viable solution for diverse IoT applications.

6.3 Use of AI

During the preparation of this work the author used ChatGPT for general aid, predominantly pointers to get started on research and text structure. After using this tool/service, the author(s) reviewed and edited the content as needed and take(s) full responsibility for the content of the work.

REFERENCES

[1] Hadeel Hussein Azeez and Mahmood Zaki Abdullah. 2023. Performance analysis of constrained application protocol (CoAP). *AIP Conference Proceedings* 2591, 1 (March 2023), 030074. https://doi.org/10.1063/5.0119584

[2] Mohammed El-hajj, Ahmad Fadlallah, Maroun Chamoun, and Ahmed Serhrouchni. 2019. A Survey of Internet of Things (IoT) Authentication Schemes. *Sensors* 19, 5 (Jan. 2019), 1141. https://doi.org/10.3390/s19051141 Number: 5 Publisher: Multidisciplinary Digital Publishing Institute.

[3] Y. Guaman, G. Ninahualpa, G. Salazar, and T. Guarda. 2020. Comparative Performance Analysis between MQTT and CoAP Protocols for IoT with Raspberry PI 3 in IEEE 802.11 Environments, Vol. 2020-June. https://doi.org/10.23919/CISTI49556.2020.9140905 ISSN: 2166-0727.

[4] C. P. Kruger and G. P. Hancke. 2014. Benchmarking Internet of things devices. In *2014 12th IEEE International Conference on Industrial Informatics (INDIN)*. 611–616. https://doi.org/10.1109/INDIN.2014.6945583 ISSN: 2378-363X.

[5] N. Naik. 2017. Choice of effective messaging protocols for IoT systems: MQTT, CoAP, AMQP and HTTP. https://doi.org/10.1109/SysEng.2017.8088251

[6] R.A. Rahman and B. Shah. 2016. Security analysis of IoT protocols: A focus in CoAP. 172–178. https://doi.org/10.1109/ICBDSC.2016.7460363

[7] Zach Shelby, Klaus Hartke, and Carsten Bormann. 2014. *The Constrained Application Protocol (CoAP)*. Request for Comments RFC 7252. Internet Engineering Task Force. https://doi.org/10.17487/RFC7252 Num Pages: 112.

[8] Johann Westphall, Leandro Loffi, Carla Merkle Westphall, and Jean Everson Martina. 2020. CoAP + DTLS: A Comprehensive Overview of Cryptographic Performance on an IOT Scenario. In *2020 IEEE Sensors Applications Symposium (SAS)*. 1–6. https://doi.org/10.1109/SAS48726.2020.9220033