Performance Impact Analysis of Securing MQTT Using TLS

Thomas Prantl, Lukas Iffländer, Stefan Herrnleben, Simon Engel, Samuel Kounev
University of Würzburg, Germany

ABSTRACT

The interconnectivity of devices on the Internet of Things (IoT) provides many new and smart applications. However, the integration of many devices—especially by inexperienced users—might introduce several security threats. Further, several often used communication protocols in the IoT domain are not out-of-the-box secured. On the other hand, security inherently introduces overhead, resulting in a decrease in performance. The Message Queuing Telemetry Transport (MQTT) protocol is a popular communication protocol for IoT applications—for example, in Industry 4.0, railways, automotive, or smart homes. This paper analyzes the influence on performance when using MQTT with TLS in terms of throughput, connection build-up times, and energy efficiency using a reproducible testbed based on a standard off-the-shelf microcontroller. The results indicate that the impact of TLS on performance across all QoS levels depends on (i) the network situation and (ii) the connection reestablishment frequency. Thus, a negative influence of TLS on the performance is noticeable only in deteriorated network situations or at a high reestablishment frequency.

KEYWORDS

Pub/Sub, MQTT, IoT, TLS, Performance

ACM Reference Format:

1 INTRODUCTION

A “smart” future based on the interaction of intelligent devices in the Internet of Things (IoT) is becoming a reality as promising applications in areas as smart cities, smart traffic, smart homes, or smart health show. IoT devices are the foundation of this "smart" future. Those typically relatively small devices equipped with sensors are often particularly resource-constrained and communicate with each other and cloud services. Various lightweight communication protocols emerged to facilitate communication between an excessive number of resource-constrained devices. One commonly applied protocol for communication in the IoT is the Message Queuing Telemetry Transport (MQTT) protocol [16]. MQTT is a lightweight publish-subscribe messaging protocol enabling efficient communication of IoT devices. A central instance—called the message broker—manages subscriptions and delivers the messages instead of passing directly from one client to another. This protocol is used, for example, in the context of smart home and industrial applications [9]. Despite all the convenience that smart IoT devices offer, one should keep in mind where the intelligence of such systems originates. It results from voluntarily surrounding ourselves with sensors that collect data about our environment, including personal data. That data is processed remotely and yields AI-driven reasoning. Accordingly, IoT device owners should have a genuine interest in the security of their data and devices—especially considering that IoT devices, on average, already suffer attacks five minutes after their connection to the Internet [13].

Due to these security risks, a correspondingly high scientific interest level in researching new security mechanisms for MQTT and examining them concerning the required performance exists (e.g., [17, 19, 22]). However, often works overlook that security mechanisms for making MQTT secure already exist for many applications and are just not used. For example, in applications in which the broker is trustworthy, TLS can prevent data such as usernames and passwords from being transmitted in plaintext, making the hijacking of IoT devices much more difficult. Several studies already investigated the use of TLS [1, 3, 4, 21, 25]. However, those publications do not contain all information regarding the used software (e.g., the MQTT or TLS implementations), workloads, metrics, measurement setup, and, respectively, accuracy. They often do not consider all three Quality of Service (QoS) levels of MQTT and completely ignore different network conditions. Those types of information are essential for ensuring their experiments’ reproducibility and their results’ validity [15].

In this paper, we present and document our reproducible performance measurements and analysis with well-described metrics. The used measurement scripts and the source code for the adapted MQTT client are available online1. Our analysis focuses on the performance loss when using MQTT with TLS in terms of throughput, broker connection establishment times, and energy efficiency. Using these analyses, we answer the relevant question for developers, whether securing MQTT using TLS has a significant negative impact on performance in typical IoT scenarios. Our contributions are threefold:

- The design of a reproducible testbed for measurements of MQTT, which supports the use of TLS, all QoS levels, and

1https://github.com/WueSePrantl/MQTT_TLS_Performance
different network scenarios, using a standard off-the-shelf microcontroller;
- the definition of suitable metrics including error measures considering the underlying measurement accuracy; and
- analyzing the impact of combining MQTT with TLS on the throughput, broker connection establishment times, and energy efficiency using our testbed.

The remainder of this paper is structured as follows. In Section 2, we describe the basics of MQTT and TLS, followed by an overview of related work in Section 3. Next, Section 4 describes our testbed design. Then, Section 5 presents the used workload patterns and metrics. Following in Section 6, we present the evaluation of the performance impact of combining MQTT with TLS. Lastly, Section 7 concludes this paper with a summary and future work.

2 BACKGROUND

For a better understanding of our setup, measurements, and their evaluation and design, this section explains the basic functionalities of MQTT and TLS. The explanations of MQTT and TLS originate from [6, 14, 20].

2.1 Message Queuing Telemetry Transport (MQTT)

MQTT is a lightweight Machine-to-Machine-protocol implementing a publish-subscribe architecture. In MQTT, several clients—which can be publisher and subscriber—and a central message broker interact. If a client wants to send a message, it publishes it under a specified topic at the message broker. The message broker forwards this message to all clients that previously subscribed to this topic. There is no direct communication between the clients eliminating coupling in time, space, and synchronization [7].

For each message published, the publisher can set a QoS level. This attribute defines the effort that is made to ensure that the data reaches its recipient. MQTT supports three different QoS levels [11], which we present in the following.

QoS 0 - At most once: The recipient does not confirm the reception of messages, and the publisher does not wait for such confirmations, nor do they store already sent messages to be able to retransmit them if necessary.

QoS 1 - At least once: Level 1 guarantees that the sender’s data will reach the recipient. For this purpose, the recipient confirms the reception of data to the publisher, who, in turn, caches the sent data to re-transmit it if necessary. However, it is possible that the recipient receives the same data multiple times or that a publisher sends data multiple times.

QoS 2 - Exactly once: Level 2 corresponds to Level 1. However it guarantees that each message is received only once by the recipient.

2.2 Transport Layer Security (TLS)

TLS uses a cipher suite to provide communication security on the TCP/IP stack at the transport level. A cipher suite consists of cryptography algorithms enabling the exchange of keys, encryption, and the securing of integrity and authenticity via message authentication codes (MACs). Thus, TLS implements a transport layer encryption between two directly communicating devices. The use of TLS combined with MQTT cannot guarantee end-to-end encryption between publishers and subscribers, but only encryption between publisher and broker or broker and subscriber. Therefore, the use of TLS with MQTT requires that the broker is trusted since it can read all messages. In practice, this is often the case since IoT devices’ owners often also provide and control the broker.

3 RELATED WORK

In this section, we discuss related literature in the area of performance analysis of MQTT with TLS. Thereby, we also highlight the novelty of our contribution.

The authors from [4] and the subsequent publication [3] propose a dynamic procedure to decide which TLS cipher suite fits best, depending on the remaining energy, desired encryption strength, and message length. The authors present a self-adaptive approach of TLS but do not consider different network situations or QoS levels and do not compare MQTT with and without TLS. Necessary information for reproducibility is missing (e.g., the used MQTT libraries), and there is also no information about the accuracy of the obtained results.

In [21], the authors compare, among other things, MQTT with all QoS levels with and without TLS. The authors also state the accuracy of their measurement results, but do not specify how the accuracy is determined. They also do not consider different network situations and information about the used workload and testbed (like the used access point or libraries) is incomplete.

The authors of [1, 25] determine the performance of all MQTT QoS levels without TLS in [25] and with TLS in [1]. However, it is impossible to compare using MQTT with and without TLS as they use different hardware. Furthermore, both papers lack a detailed description of (i) the measurement setup (e.g., the MQTT client implementation), (ii) the MQTT client’s behavior or workload (such as inter-arrival time of messages or the message size), (iii) the considered network scenario, and (iv) information about the measurement accuracy.

We present a testbed for reproducible measurements to close these gaps, including all information about used workloads, hardware, and software. This testbed allows adjusting different network situations and supporting all MQTT QoS levels, using TLS, and showing measurement accuracy propagation through the metric calculations. We then use the testbed to perform measurements for the comparison of MQTT with and without TLS. An initial comparison of MQTT with and without TLS has also been available in [21]. However, we (i) not only address energy efficiency as a metric but also connection setup times with the broker and throughput, (ii) clearly show our understanding of the accuracy of our metrics and how achieve it. We (iii) additionally consider different network situations, and (iv) use clearly defined and flexible workloads that allow specific parameters (e.g., message length) to be changed, which means that our analysis is not limited to a specific situation.

4 TESTBED CONCEPT & REALIZATION

We need an appropriate evaluation environment to perform measurements and analyze the influence of TLS on the performance of MQTT in an IoT context. In the following, we present a concept and realization of such a testbed.
4.1 Testbed Concept

We rely on a typical IoT device as an MQTT client whose performance is observable to examine the performance of MQTT with and without TLS on IoT devices. The MQTT client needs an MQTT broker that it can reach via an appropriate access point to communicate using MQTT. Since different network conditions such as packet loss should be easily configurable, an additional requirement is that either the broker or access point must offer appropriate functionality for manipulating the network traffic.

IoT networks typically consist of many devices that communicate with each other. However, considering that (i) we are only interested in the performance of the observed IoT device, which (ii) only communicates directly with the broker since (iii) the communication between clients is decoupled in time, space, and synchronization [7] through the broker (as described in Section 2), it is sufficient for the performance analysis of the MQTT client to model only the communication between broker, access point, and MQTT client. Hence, the broker must be potent enough so that its performance is not affected by the presence of other IoT devices. Usually, in IoT applications, most of the devices are resource-scarce devices. In such settings, the natural choice is—if not already present—to complement the system with a powerful device or Cloud resources that can act as an MQTT broker. Accordingly, it is readily achievable to scale the broker without affecting the whole system. This requirement also reflects the fundamental design principle of MQTT, that the broker handles the complexity regarding communication. Therefore, the broker should be provided with appropriate resources to relieve the low performance of—often only battery-powered—IoT devices as much as possible and achieve their best possible performance.

Consequently, we propose the concept of a testbed for performance measurements of MQTT with and without TLS with the following features: (i) a frequently used IoT device serving as an MQTT client, (ii) a separate device that serves as an MQTT broker, (iii) an access point, (iv) the access point or broker must be able to configure different network situations quickly, and (v) appropriate measuring equipment.

4.2 Testbed Realization

The testbed components comprise of an MQTT broker and MQTT client, a power meter, the MQTT client power supply, and a WiFi Access Point (see Figure 1). We chose the ESP8266 microcontroller as an MQTT client since it is a popular microcontroller supporting, for example, monitoring heart rate and inter-beat interval for several subjects [26] or home automation [10]. The ESP8266 is a 32-bit microcontroller from Espressif Systems and a so-called System-on-a-Chip. The Elegoo Power Supply Module 1PC powers the ESP8266 as it can directly provide the 3.3V required by the ESP8266. A Yokogawa WT310 power measurement device monitors the power consumption. As Access Point, we use a TELEKOM Speedport Smart router. The MQTT broker runs on a laptop with Windows 10 Enterprise Version 1803 (Build 17134.1365), having an Intel(R) Core(TM) i7-8550U CPU with 1.8 GHz and four cores, 16 GB RAM, and an Intel(R) Dual Band Wireless-AC 8265 network card.

We use the MQTT client implementation from [12] to realize the MQTT client on the ESP8266 since it supports TLS and all three MQTT QoS levels. We decided to use the Mosquitto broker in version 1.6.9 as an MQTT broker on the Windows 10 laptop. We generated the broker’s certificates required for TLS 1.2 using the Windows OpenSSL version. The ESP8266 stores the fingerprint of the created broker certificate to verify the broker’s identity before establishing an encrypted connection.

We used the network traffic control program NetBalancer [24] in version 9.16.1 on the Windows 10 laptop to create different network conditions. NetBalancer allows controlling the upload and download of individual applications by defining a packet loss rate for an application’s upload and download link. Using NetBalancer, we can manipulate the packet loss rate for the communication channel between client and broker.

5 METHODOLOGY FOR MEASUREMENT ANALYSIS

This section describes the methodology we used to analyze the performance impact of TLS. Specifically, we present the used workload patterns and metrics.

5.1 Workload Patterns

We consider three different workload patterns for which we want to evaluate the performance impact of TLS and present them in more detail below. Thereby, we use a state diagram to describe the program’s behavior on the ESP8266 microcontroller for each workload pattern. We assume that the following parameters have been defined initially for each state diagram: QoS, payload size B, message repetitions R, and time between the start of transmission of successive messages T. Also, we use the term Deep Sleep as a synonym for the ESP8266’s energy-saving mode. Next, we present the three workload patterns: Continuous Operation (CO), Operation with Deep Sleep (ODP), and Connection Establishment (CE).

Continuous Operation (CO): Figure 2 illustrates the CO workload pattern, in which the microcontroller never uses Deep Sleep. In this pattern, the microcontroller tries to publish a message with payload B and QoS q every T seconds. The publishing process of a message consists of two steps: (i) Establishing a connection to the broker if there is no connection to the broker yet, and (ii) sending the message to the broker. Both steps together may take (1) shorter or (2) longer than T seconds. In case (1), the microcontroller would delay the start of the next publishing process until T seconds have passed since the last publishing process started. In case (2), the broker will stop the current publishing process after T seconds and start publishing the next message. In total, the microcontroller tries to publish R messages using this pattern.
Figure 2: Continuous operation (CO) - Every $T$ seconds a publishing process for a message with payload of $B$ using QoS $q$ is started. In total, $R$ messages are tried to be published.

Operation with Deep Sleep (ODP): Workload pattern ODP, as shown in Figure 3, is very similar to the CO workload. The main difference is that the microcontroller in this pattern switches to Deep Sleep while delaying the next publishing process. Since at QoS Levels 1 and QoS Level 2, the microcontroller must wait for the successful transmission confirmation, the microcontroller must accordingly delay the start of Deep Sleep until receiving the confirmation.

Figure 3: Operation with Deep Sleep (ODP) - Like the CO workload pattern, except that Deep Sleep is used between the publishing processes.

Connection Establishment (CE): The CE workload pattern, see Figure 4, consists of the microcontroller establishing first a connection to the WiFi and then to the broker. After establishing this connection, the microcontroller immediately closes its connection to the broker and WiFi. The microcontroller repeats these three steps $R$ times. This workload pattern captures the connection setup time with the broker, which is, for example, important for motion detectors of security systems.

5.2 Metrics
Since IoT devices typically have limited hardware and limited power supply, it is especially critical to use the available energy as efficiently as possible. Therefore, we use energy efficiency as a comparative metric to evaluate the measurement results. As a metric, energy efficiency allows evaluating different approaches for the same application and determining the most efficient variant. In this way, the developer can select the most efficient variant for his application and use it to dimension the required battery accordingly. Following the SPEC specifications [23], we define in Equation 1, the energy efficiency $E$ as the ratio of the payload throughput to the power consumption:

$$ E = \frac{\text{Payload Throughput}}{\text{Power Consumption}} $$

We introduce the abbreviation $W$ and define it in Equation 2 as the average power consumption per second for power consumption. In this equation, $n$ stands for the measurement duration in seconds and $W_i$ for the power consumption during the $i$th second.

$$ W = \frac{1}{n} \sum_{i=1}^{n} W_i $$

In our measurements, we send $r$ messages with only fixed payload sizes of $B$ bytes at fixed intervals $T$. Therefore, we define the payload throughput in Equation 3 as the average successfully transmitted payload bytes normalized to the length of the sending interval $T$. In our case, either all $B$ bytes reach the broker or 0 bytes. Therefore, we model the amount of successfully sent bytes during the $i$th interval as the product of $B$ and $\delta_i$. $\delta_i$ indicates whether the bytes’ transfer was successful or not in Equation 4.

$$ \text{Payload Throughput} = \frac{1}{r} \sum_{i=1}^{r} \frac{B \cdot \delta_i}{T} = \frac{B}{r \cdot T} \sum_{i=1}^{r} \delta_i $$

$$ \delta_i = \begin{cases} 1, & \text{the broker gets the } i\text{-th message} \\ 0, & \text{the broker does not get the } i\text{-th message} \end{cases} $$

We use the Gaussian error propagation in Equation 5 to determine the accuracy of the measured payload throughput. Thereby, in Equation 5, $\Delta T$ describes how precisely the ESP8266 keeps the time intervals after which it starts publishing a message. We assume that $\Delta T$ is the largest occurring deviation from the fixed time interval $T$ due to the unknown accuracy of the internal clock of the ESP8266.
We performed measurements of the connection time and the energy efficiency $E$ following the SPEC specifications [23].

$$E = \frac{\text{Payload Throughput}}{\text{Power Consumption}} = \frac{B \cdot \sum_{i=1}^{r} \delta_i}{T \cdot r \cdot \sum_{i=1}^{n} W_i}$$

Using Equations 3 and 2, Equation 6 describes the energy efficiency $E$.

$$\Delta E = \frac{B \cdot n \cdot \sum_{i=1}^{r} \delta_i}{r} \sqrt{\frac{\Delta T^2}{W_{\text{total}}^2} + \frac{\Delta W_{\text{total}}^2}{T^2}}$$

According to the Gaussian error propagation, we compute $\Delta W_{\text{total}}$ using Equation 8, where $\Delta W_i$ is the measurement error of the $i$th second’s energy consumption.

$$\Delta W_{\text{total}} = \sqrt{\sum_{i=1}^{n} \Delta W_i^2}$$

According to the manufacturer, Yokogawa’s power measurement error is $\pm 0.1\%$ of reading $+ 0.2\%$ of range) [5]. The range error is 0.0006 Watt because we have set the measuring ranges to 3V and 100mA. These considerations result in the final calculation of $\Delta W_{\text{total}}$ according to Equation 9.

$$\Delta W_{\text{total}} = \sqrt{\sum_{i=1}^{n} \Delta (0.1\% \cdot W_i + 0.0006 \cdot W)^2}$$

We consider the errors $\Delta W_{\text{total}}$ of the power consumption and $\Delta T$ of the publishing intervals and how they propagate through the calculations to evaluate our accuracy of energy efficiency measurements. We use the Gaussian error propagation to model the error propagation, allowing us to calculate the energy efficiency error $\Delta E$ according to Equation 7.

$$\Delta E = \frac{B \cdot n \cdot \sum_{i=1}^{r} \delta_i}{r} \sqrt{\frac{\Delta T^2}{W_{\text{total}}^2} + \frac{\Delta W_{\text{total}}^2}{T^2}}$$

According to the Gaussian error propagation, we compute $\Delta W_{\text{total}}$ using Equation 8, where $\Delta W_i$ is the measurement error of the $i$th second’s energy consumption.

$$\Delta W_{\text{total}} = \sqrt{\sum_{i=1}^{n} \Delta W_i^2}$$

According to the manufacturer, Yokogawa’s power measurement error is $\pm 0.1\%$ of reading $+ 0.2\%$ of range) [5]. The range error is 0.0006 Watt because we have set the measuring ranges to 3V and 100mA. These considerations result in the final calculation of $\Delta W_{\text{total}}$ according to Equation 9.

$$\Delta W_{\text{total}} = \sqrt{\sum_{i=1}^{n} \Delta (0.1\% \cdot W_i + 0.0006 \cdot W)^2}$$

In addition to energy efficiency, we also consider the time $t_{\text{con}}$ it takes a client to connect to the broker and become ready to start sending messages as a metric. Since we measure $t_{\text{con}}$ through the internal clock of the ESP8266 and do not know its accuracy, we assume that the error $\Delta t_{\text{con}}$ is the standard deviation of $t_{\text{con}}$.

6 EVALUATION

We performed measurements of the connection time and the energy efficiency for the different workload patterns in two network scenarios: (i) assuming a stable connection and (ii) simulating an additional packet loss for the communication channel between the broker and the ESP8266 using NetBalancer. The workload patterns C0 and OD (i.e., $O$) rely on fixed time intervals $T$ in which ESP8266 can try to publish a message. Thus, we start our evaluation with the workload pattern CE to determine how long it takes the ESP8266 to establish a connection to the broker with and without TLS in different network situations. The information about the connection time is essential for the patterns C0 and OD. The period $T$ should be large enough to allow ESP8266 to (i) connect to the broker, and (ii) try to publish a specific message. Based on this, we analyze the energy efficiency by using the patterns C0 and OD.

The goal of our analysis here is to answer the following research questions (RQs). **RQ.1**, how does TLS affect the connection time between client and broker depending on the network situation. **RQ.2**, how does TLS affect a client’s energy efficiency under stable network conditions where (RQ.2.1) the connection with the broker is held continuously or (RQ.2.2) when the client holds the connection only to send a message and otherwise switches to energy-saving mode. **RQ.3**, how does TLS affect a client’s energy efficiency when combining the parameters from RQ.1 and RQ.2. **RQ.4**, how does TLS affect the energy efficiency in different network situations when using different QoS levels and the energy-saving mode.

6.1 Analysis of the Broker Connection Times

We performed measurements using the CE workload pattern to evaluate the time it takes the ESP8266 with and without TLS to connect to the broker. We varied the packet loss between 0% and 30% in steps of 10% for TLS and without TLS. In four rounds of repetition, for both TLS enabled and disabled, we established a new connection 400 times. Figure 6.1 shows the corresponding measurement results for the mean value and the standard deviation of the connection establishment time $t_{\text{con}}$. For TLS enabled and disabled, $t_{\text{con}}$ and its error increases with increasing packet loss. Thereby, $t_{\text{con}}$ and its error increases in the same network situation more when using TLS. Thus, within the scope of our measurements, we can answer RQ.1: The use of TLS negatively affects the connection time, and the worse the network situation, the stronger the adverse effect. Besides, the standard deviation of the connection time increases due to TLS, whereby the negative influence here is also more significant when the network situation is worsening.

6.2 Energy Consumption Impact of TLS

We investigate in this section our hypothesis that the use of TLS negatively influences the energy consumption for IoT communication relying on the MQTT protocol. We analyze our claim in two
different network scenarios, with and without packet loss. Using previous measurement results helps select a reasonable length for the sending interval $T$ for CO and ODP patterns. Thus, the ESP8266 has enough time on average to both (i) connect to the broker, and (ii) try to publish a message. As we consider in the following up to 15% packet loss, we select an appropriate value for $T$ using Figure 6.1. For the following measurements, we set the sending interval $T = 10$ seconds.

**Influence of TLS on Energy Consumption for Stable Connections.** We performed measurements for both workload patterns CO and ODP to identify the influence of TLS on the energy consumption in stable networks, with the following fixed parameters: the number of messages the ESP8266 should try to publish $R = 120$, the time between the start of two publishing processes $T = 10$ seconds, and QoS level $q = 0$. We varied the payload $B$ between 20 Bytes and 100 Bytes in 20-Byte steps and configured no extra packet loss with NetBalancer. Figures 6 and 7 illustrate the measurement results for both workload patterns. It is essential to note that each publishing process was successful for all measurements to interpret these figures. Within the scope of our measurement accuracy and range, we can answer $RQ.2.1$ and $RQ.2.2$ as follows: We observe no significant performance degradation with TLS both for continuously held connections and when using energy-saving mode.

However, our measurements allow us to conclude:

1. For both patterns, energy efficiency increases with the message size, regardless of using TLS or not.
2. The energy efficiency of ODP is—regardless of using TLS or not—significantly higher than that of CO (considering the Gaussian error values). This result indicates that using Deep Sleep is worthwhile as this is the sole difference between the two patterns.

3. Our scenario’s energy efficiency could, in the worst case, deteriorate by a maximum of 9 Byte/Joule for ODP and 3 Byte/Joule for CO when using TLS.

**Influence of TLS on Energy Consumption in Scenarios with Packet Loss.** Since we could not detect any provable difference in using TLS or not for energy efficiency in network situations without additional packet loss, we concentrate on situations with additional packet loss in the following. The previous measurements showed that using Deep Sleep can have a positive effect on energy efficiency. Accordingly, we next analyze the assumption that under worsened network conditions, there is a difference in energy efficiency between using TLS and not since the additional use of TLS alone increases the average connection time with the broker and reduces the possible time in which Deep Sleep is available. Therefore, we performed measurements for the workload pattern ODP, with the fixed parameters $R = 720$, $T = 10$ seconds, $q = 0$, $B = 40$, and varied the packet loss between 5% and 15% in 5% steps using NetBalancer. Figure 8 illustrates the energy efficiency. The measurements answer $RQ.3$ since they allow to conclude that, within the scope of the measuring accuracy and range, (i) there is a provable difference in the energy efficiency when using TLS and (ii) that regardless of the use of TLS, the energy efficiency decreases with increasing packet loss. The payload throughput can partially explain this difference in energy efficiency when using TLS for the different network situations (see Figure 9): The same statements that apply to energy efficiency also apply to the payload throughput. TLS negatively impacts a vital component of the energy efficiency since the payload throughput is essential for energy efficiency, and the use of TLS degrades it. However, our measurements do not allow us to conclude that the energy efficiency difference is solely related to...
Influence of the QoS Level in Scenarios with Packet Loss. Finally, we suspected that under network conditions with additional packet loss, higher QoS levels would result in higher throughput, which could also have a positively affect energy efficiency. We performed measurements for the workload pattern ODP as the previous experiments showed that this increases energy efficiency to evaluate this assumption. We defined the parameters $R = 720$, $T = 10$ seconds, $B = 40$, and varied the packet loss between 5% and 15% in 5% steps using NetBalancer and varied $q$ between 0 and 2 in steps of 1. Figures 10 and 11 illustrate the measurement results. Figure 11 shows that using QoS levels higher than 0 increases the throughput in our measurements, regardless of whether using TLS or not. Upon packet loss, the message is resent at QoS 1 or 2, explaining the higher throughput. However, there is no significant difference to derive that when using TLS, QoS Level 1 allows higher throughput than QoS Level 2; the same applies to the measurements without TLS. Nevertheless, our measurements show that (i) the throughput decreases with increasing packet loss across all QoS levels and (ii) the use of TLS has a provable negative effect for QoS Level 1 and QoS Level 2 for a packet loss rate of 15%. Concerning energy efficiency (see Figure 10), we answer RQ 4 as follows: The use of TLS has a provable negative effect on energy efficiency within our measuring range and accuracy when network conditions deteriorate, even when using higher QoS levels. This effect results from the fact that when using TLS with increasing packet loss, the Deep Sleep mode can only activate for a shorter time than when not using TLS. Unlike throughput, the use of QoS levels higher than 0 does not result in a demonstrable improvement in energy efficiency. However, the use of QoS Level 2 does even result in a provable deterioration of energy efficiency than QoS Level 0. These results suggest that the throughput is vital for energy efficiency and the effort that ESP8266 takes to successfully send a message, which has a direct influence on the time in which the Deep Sleep is available.

6.3 Threats to Validity

This paper focuses on specifying a testbed, measurement workflow, and metrics for reproducible measurements of the impact on performance and energy efficiency when integrating TLS with MQTT-based IoT communication. Using our concept, we performed several measurements to (i) assess the energy efficiency and the performance impact when combining MQTT with TLS and (ii) show our concept’s applicability. However, we have identified the following threats to the validity of the evaluation results.

First, we focused on the performance of the MQTT client’s performance in all measurements, using only a single MQTT client and assessing the impact of TLS on its performance. Thereby, we did not take into account a higher number of MQTT clients also using TLS. Handling multiple clients at the same time can influence the performance of the broker. Multiple clients could harm message delivery times from publishers to subscribers because as the load increases, messages are queued and delayed in processing by the broker. However, we focus on the properties of the direct connection between a client and the broker. We plan to use our IoT network emulator for analyzing effects resulting from the interactions of multiple clients as part of our future work [8]. Further, as MQTT brokers today often run in Cloud environments—such as specialized services from AWS, Azure, or Google’s Cloud Platform—the scalability of such platforms helps to avoid negative impacts for larger systems.

Second, the respective metrics contain their corresponding errors, which is because—compared to servers or desktop PCs—microcontrollers have significantly lower current and voltage levels. Therefore, we can only make statements within the scope of the existing measurement accuracy and only detect influences that have a more significant impact than this accuracy.

Third, we focused on MQTT as a communication protocol. Similar studies (such as [2]) have shown significant performance differences between the standard IoT protocols such as MQTT, CoAP, and DDS. In contrast, we focus on the reproducibility of the results and the reusability of the measurement environment. For future work, we plan to work on generalizing our results to other standard IoT communication protocols.

Fourth, although TLS can protect the communication channel, there is still a risk of tapping messages at the application level after decryption. However, this problem would also occur with end-to-end encryption on the IoT device. From the IoT device’s point of view, TLS and end-to-end encryption have corresponding keys and messages that must be encrypted or decrypted. Thus, both methods have the same problems on the application level.
7 CONCLUSION & FUTURE WORK

One of the most prominent protocols for communication with IoT devices is the MQTT protocol. However, out-of-the-box MQTT does not integrate security mechanisms. In this work, we performed an analysis of the impact on performance and energy efficiency when complementing MQTT with TLS. To overcome the issues of reproducibility of related studies (e.g., [21] and [1]), we define a hardware testbed, metrics, and workload patterns. The results support our hypotheses. First, the use of TLS negatively impacts the connect time to the broker, especially in settings with higher packet loss (RQ.1). Second, assuming a stable network connection without packet loss, there are no significant energy consumption effects when adding TLS (RQ.2.1 and RQ.2.2). However, energy efficiency increases with a higher message payload and benefits from the deep sleep mode. Third, in the scenario with deep sleep use, packet loss, and QoS Level 0, TLS negatively influences the throughput and the efficiency (RQ.3).

Further, energy efficiency decreases with higher packet loss rates. Lastly, QoS Level 1 and QoS Level 2 can increase the throughput independently from using TLS. For the settings with small packet loss rates, TLS does not influence the throughput. Increased packet loss rates decrease energy efficiency and throughput, even with higher QoS levels (RQ.4). The application of QoS Level 2 can even reduce energy efficiency in contrast to QoS Level 0. In all measurements with packet loss, TLS decreases energy efficiency. Additionally, the results show no identical relation between energy efficiency and throughput that is valid for all QoS levels and packet loss rates.

With this paper, we contribute to the increasing body of research in IoT communication by conceptualizing a measurement environment for reproducible analysis of the impacts on energy efficiency and performance when securing MQTT with TLS. In this work, we focus on the direct connection between one client and its broker. Studying the effects in environments with multiple MQTT clients is part of future work. We also plan to generalize our results to other IoT communication protocols and show how, using our results, battery sizing can be done in practice.

ACKNOWLEDGMENTS

This research has been funded by the Federal Ministry of Education and Research of Germany in the framework KMU-innovativ - Verbundprojekt: Secure Internet of Things Management Platform - SIMPL (project number 16KIS0852) [18].

REFERENCES

[1] Edgaras Baranauskas et al. 2019. Evaluation of the impact on energy consumption of MQTT protocol over TLS. In IFVS.
[22] M. Singh et al. 2015. Secure MQTT for Internet of Things (IoT). In CSNT. 746–751. https://doi.org/10.1109/CSNT.2015.16