Formal Specification and Verification of MQTT Protocol Using CoQ Proof Assistant

1st Ibtissem Talamali
LIMOSE Laboratory
Department of computer science
University of M'Hamed Bougara
Faculty of science
Boumerdes, Algeria
i.talamali@univ-boumerdes.dz

2nd Razika Lounas
LIMOSE Laboratory
Department of computer science
University of M'Hamed Bougara
Faculty of science
Boumerdes, Algeria
razika.lounas@univ-boumerdes.dz

3rd Mohamed Mezghiche
LIMOSE Laboratory
Department of computer science
University of M'Hamed Bougara
Faculty of science
Boumerdes, Algeria
mohamed.mezghiche@univ-boumerdes.dz

2024 International Conference on Advanced Aspects of Software Engineering (ICAASE) | 979-8-3315-2281-0/24/$31.00 ©2024 IEEE | DOI: 10.1109/ICAASE64542.2024.10850926

***Abstract*—Because of its efficiency and lightweight design, the Message Queue Telemetry Transport (MQTT) protocol is widely used in Internet of Things (IoT) and messaging systems. However, despite its widespread use, guaranteeing the correctness and reliability of MQTT implementations remains a challenge. While model checking has been used to specify and verify properties of MQTT systems, there is a notable absence of formal verification using theorem proving methods such as the Coq proof assistant. This research attempts to fill this gap by formalizing and verifying the MQTT protocol in Coq. This article opens up a new field of study, paving the way for more rigorous verification techniques and enhancing the robustness and reliability of MQTT-based systems through a formalisation of the protocol main concepts and the demonstration of safety properties.**

***Index Terms*—Internet of Things, Formal Verification, MQTT Protocol, Theorem Proving, Coq.**

I. Introduction

The Internet of Things (IoT) encompasses a vast array of devices, necessitating the use of standardized communication protocols that are both energy-efficient and highly reliable. The Message Queue Telemetry Transport (MQTT) protocol, which is popular for its reliability and lightweight architecture, is one of the most extensively used IoT protocols. It makes efficient and dependable communication in IoT networks possible. The MQTT protocol is a lightweight messaging protocol designed for constrained devices and low-bandwidth, high-latency, or unreliable networks. Despite its widespread adoption, ensuring the reliability and correctness of MQTT implementations is a significant challenge due to the protocol's complexity and the critical nature of its applications. It may be found in applications across several fields such as eHealth applications and smart homes [1].

The use of MQTT in applications requiring a high level of rigor fostered several researches using formal methods to establish its correctness properties. These methods offer mathematical and logical means to specify systems and reason about their properties [2]. Among formal methods, model checking is the most widely used to verify safety properties on MQTT [3]. Model checking approaches have been used in previous studies to design and verify characteristics of MQTT systems; however, these approaches lack the expression of high order properties and faces some issues such as combinatory explosion of the number of the states. In response to these concerns, theorem proving approaches have been considered to verify IoT applications, and protocols ([4], [5]).

Formal verification using theorem proving offers a higher degree of rigor. Theorem proving is a technique where both the system and its properties are expressed as formulas in some mathematical logic. This logic is given by a formal system, which defines a set of axioms and inference rules. Theorem proving is the process of finding a proof of a property from the axioms of the system. Theorem provers are increasingly used today in the verification of safety-critical properties of hardware and software designs. Tools like the Coq proof assistant enable the development of formal proofs that certain properties hold for all possible executions of the system increasing the level of trust in its accuracy. Unlike model checking, theorem proving does not suffer from state space explosion, making it suitable for verifying complex protocols.

Despite the advantages of theorem proving, there is a notable absence of research applying this method to the formal verification of the MQTT protocol. While some studies have explored the formal verification of other protocols using Coq, MQTT has not yet been the focus of such efforts. This gap presents an opportunity to leverage theorem proving for more rigorous verification of MQTT, enhancing the reliability of systems that depend on this protocol.The objective of this research is to formalize the MQTT protocol in Coq and verify its key properties. By doing so, we aim to provide a more rigorous verification of MQTT, contributing to the reliability and robustness of systems that rely on this protocol.

The remainder of the paper is organized as follows. In Section 2, we outline some related work. In Section 3, we briefly discuss the basics of the MQTT protocol. Section 4 gives basic concepts about CoQ proof assistant. In Section 5, we describe our formalization of MQTT procol constructs using CoQ. Finally, we conclude the paper and give some perspectives in Section 6.

II. Related work

Several researchers endeavored to analyse the application of formal methods for the MQTT protocol with an extensive use of model checking. In [6] the authors focused on the formal modeling and performance analysis of an Internet of Things (IoT) protocol. The authors employed probabilistic timed automata for the formal modeling of the Message Queue Telemetry Transport (MQTT) protocol and uses the UPPAAL SMC tool for performance evaluation through statistical model checking. The article [7] provides a detailed method for specifying and verifying the MQTT protocol using PlusCal-2, an extension of the TLA+ language. It outlines the process of formal specification, verification, and the benefits of using PlusCal-2 for safety and reliability. In paper [8] evaluates security-enhanced IoT protocols with a focus on MQTT. It introduces a Secured Message Queue Telemetry Transport (SMQTT) protocol, which incorporates cryptographic techniques to improve security. The protocol undergoes formal verification using ProVerif to confirm it meets the desired security attributes. The verification process aims to address potential security gaps in the original MQTT standard, assessing metrics such as confidentiality and integrity.

Although there has been significant success with the application of model checking for IoT protocols, the critical nature of certain IoT applications and protocols necessitates considering the use of theorem proving approaches, which offer a higher level of rigor and expressiveness. In [4], the authors investigate the use of the Coq proof assistant to verify and certify IoT systems in Smart Cities. They modeled Smart Cities as Cyber-Physical Systems and formalized it into Finite State machines, which improves verification in this area through logic-based systems and proof assistants such as Coq. The paper [9] introduces a novel approach for ensuring the correctness and reliability of IoT communication protocols through a framework in Event-B. The framework is used to model several protocol and to verify communication properties such as connection-establishment and caching. The authors used both simulation and proof obligation to establish the properties. The most closely related work to our proposal is presented in the paper [5]. In this work, the authors use theorem proving to establish the correctness of QoS properties in IoT protocols, employing the Unifying Theories of Programming (UTP). Our approach differs significantly in both methodology and tools. We employ the Coq proof assistant for the specification and verification processes, which offers a higher level of rigor and expressiveness due to its basis in constructive logic. Another novelty of our work is the detailed formalization of protocol states and the use of Coq functions to align the specification with the actual implementation.

III. MQTT Protocol

Message Queue Telemetry Transport (MQTT) is an open standard messaging protocol that has existed for over 20 years. It is a publish/subscribe protocol designed to reliably transfer messages under low-bandwidth network conditions and during long network delays. The connection types can be classified as machine-to-server (M2S), server-to-server (S2S), and machine-to-machine (M2M) communication designs, as well as routing mechanisms (one-to-many, one-to-one, and many-to-many) [10]. MQTT is a widely used application layer protocol for transmitting data among devices in IoT due to its lightweight nature and scalability. It is also used in various applications where data exchange is necessary, such as asset management, automotive telematics, traffic monitoring, home automation and supervisory control and data acquisition [11].

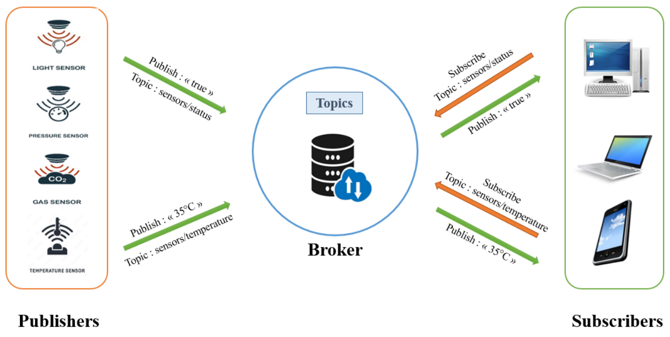

Fig. 1. MQTT architecture

A. MQTT components

The MQTT protocol consists of four major components to facilitate communication between IoT devices: the broker, subscriber, publisher, and topics, as shown in Figure 1.

1) Broker: A broker is a central server that mediates communication between clients. It manages the distribution of information and is primarily responsible for receiving all messages from publishers, filtering them, determining the interested recipients, and then delivering the messages to all subscribed clients [12]. It performs the following tasks: accepting client requests; receiving messages published by users; processing various requests, such as subscribe and unsubscribe actions from users; and delivering received messages from publishers to the appropriate users depending on the quality of service QoS required by the message.

MQTT includes a quality of service (QoS) feature with three levels of message delivery. Each level represents an increasing degree of effort by the server to ensure message delivery. Higher QoS levels provide greater reliability but may introduce latency or increase network bandwidth usage [1]:

1) **QoS 0 (At most once)**: The message is sent once, with no confirmation of delivery.
2) **QoS 1 (At least once)**: The message is resent until delivery is acknowledged.
3) **QoS 2 (Exactly once)**: The message is delivered only once using a two-step handshake between sender and receiver.

2) Publisher: The publisher's role is to provide messages to the subscribers, labeled with the relevant themes. There is no direct communication between the publisher and the subscribers. It only forwards the communication to the broker, who is in charge of sending it elsewhere. Before sending any messages, the publisher connects with the broker. Once connected, it can share the messages and close the connection by disconnecting from the broker [7].

3) Subscriber: Its role is to ask the broker for the topics subscription in order to subscribe to them. Subscribers stay in contact with the broker and provide it with information on the topics they have subscribed to. Through the broker, the publishers send all of the updated content to the associated subscribers [7].

4) Topics: In MQTT, messages are published and subscribed to under specific topics (categories), which act like subject fields. A key feature of MQTT topics is the use of wildcard designators (tag), allowing for the restriction of received messages to particular topics [1]. Similarly, the subscribers are responsible for subscribing to the topics. Then, they will receive all the messages for their subscribed topics. The broker maintains the list of all active topics along with registered subscribers for each topic [7].

B. MQTT Messages

Messages are the holders of information that has been sent to the broker by the publisher or received by the subscriber from the broker [8]. There are different types of message exchanged during the execution of the MQTT protocol. The MQTT messages are explained on Table I.

TABLE I
MQTT MESSAGES AND THEIR DESCRIPTIONS

Message Name	Description
`CONNECT`	Sent by the publisher or subscriber to the broker to initiate the connection.
`DISCONNECT`	Sent by the publisher or subscriber to the broker to terminate the connection.
`CONNACK`	Acknowledgement sent by the broker to the publisher or subscriber, confirming receipt of the `CONNECT` message.
`PUBLISH`	Initiated by the publisher to share information with the subscribers.
`PUBACK`	Response message sent by the broker or subscriber to acknowledge the receipt of the `PUBLISH` message.
`PUBREC`	Response to the `PUBLISH` message with quality of service level 2.
`PUBREL`	Response to the `PUBREC` message during publishing with quality of service level 2.
`PUBCOMP`	Completion message in response to the `PUBREL` message, finalizing the publish process.
`SUBSCRIBE`	Used by the subscriber to subscribe to a topic with the broker.
`SUBACK`	Acknowledgement sent by the broker in response to the `SUBSCRIBE` message.
`UNSUBSCRIBE`	Used by the subscriber to notify the broker that they no longer wish to receive messages for a previously subscribed topic.
`UNSUBACK`	Acknowledgement sent by the broker to the subscriber in response to the `UNSUBSCRIBE` message.
`PINGREQ`	Sent by the publisher or subscriber to let the broker know they are still alive when no other communication is occurring.
`PINGRESP`	Response message sent by the broker in reply to the `PINGREQ` message.

IV. THE COQ PROOF ASSISTANT

Coq is an interactive proof assistant and a programming language widely used in the field of formal verification. It is developed since 1986 at INRIA and has since benefited from numerous contributions [13]. It is based on a higher-order logic known as the Calculus of Inductive Constructions. The essential components of the Coq system are:

- A type-checking kernel and a mechanism for constructing well-typed environments. It allows checking correctness of definitions, functions, and proofs.
- A specification language for developing theories: Gallina. It allows users to define required objects: types, functions, predicates, and programs.
- Tooling to assist in the interactive construction of proofs using proof tactics: commands used to perform demonstrations step by step.
- A program extractor: It allows the extraction of a correct-by-construction program from the proof of its specification. It provides extraction to Ocaml and Haskell.

A. Types and functions definition

The Coq language allows to define the types of data that are usually encountered in other programming languages. It also provides an important notion of inductive type. Inductive types allow to define new data types by giving a set of constructors. These constructors can then be used to build values of the inductive type. Inductive types are powerful because they can be used to define both simple data types and complex structures, like trees or lists. The syntax for declaring an inductively defined type is:

Inductive $name : sort := c_1 : C_1 \mid ... \mid c_n : C_n.$

where name is the name of the type to be defined; sort is one of Set or Type (or even Prop); c_i are the names of the constructors and C_i is the type of the constructor c_i. For example, the booleans are defined inductively as follows:

INDUCTIVE BOOL : SET :=
| TRUE : BOOL
| FALSE : BOOL.

The introduction of new definitions to name object or define functions uses the keyword *Definition* as presented by the following syntax: **Definition** *name args* : *type* := *term.*

When the function is recursive, the keyword **Fixpoint** is used instead of **Definition**. For example, the square function can be defined as follows:

DEFINITION SQUARE (X:NAT) : NAT := X * X.

The definition of functions takes advantage of inductive types, by providing a primitive operator allowing filtering, This mechanisms is called pattern matching. The Coq syntax is the following:

match $term$ **with** $\mid c_1\ args_1 \Rightarrow term_1 ... \mid c_n\ args_n \Rightarrow term_n$ **end**

For example, if **n** has type **nat**, the function checking whether **n** is **0** can be defined as follows:

DEFINITION ISZERO N := MATCH N WITH $\mid O \Rightarrow true \mid S\ x \Rightarrow$ FALSE END.

B. Manipulation of proofs

In order to establish that a proposition is true, we need to produce a proof. Following the approach introduced by R. Milner for the LCF system, we use backward reasoning with tactics. A tactic transforms a goal into a set of subgoals such that solving these subgoals is sufficient to solve the original goal. The proof succeeds when no subgoals are left. In practice, we introduce a new goal in Coq using one of the following commands with prop representing a logical proposition (a well-formed term of type Prop).

Lemma id : prop. **Theorem** id : prop.

V. Formalization and verification of MQTT protocol

This section presents the formalization of the MQTT protocol using the Coq proof assistant. The formalization is divided into three principal parts: the formalization of the concepts and components of the protocol, primarily using Coq inductive types; the formalization of the protocol's behavior using Coq functions; and the statement of properties and their proofs in the form of Coq theorems.

A. Specification of the basic concepts

The formalization begins with the definition of the four main concepts of MQTT. In the following Coq definitions, a topic is defined as an atomic string, and a message is defined as a record containing the message identifier, the data, and a natural number that specifies the QoS level. The subscriber, publisher, and broker are also represented as records. The subscriber is defined by its identifier, its connection state, and a list of topics of interest. The publisher is defined by its identifier and connection state, represented as a boolean. The broker is defined by a list of subscribers, a list of publishers, and a list of pairs *(topic, message)* that stores messages and their corresponding topics.

```
Definition Topic := string.

Record Message := {
  topic : Topic;   (* The subject/topic of the message *)
  payload : string;  (* The actual data being sent *)
  qos : nat;  (* Quality of Service level: 0, 1, or 2 *) }.

Record Subscriber := {
  subscriber_id : string;
  subscriber_connected : bool;
  subscriptions : list Topic;}.

Record Publisher := {
  publisher_id : string;
  publisher_connected : bool;}.

Record Broker := {
  subscribers : list Subscriber;
  publishers : list Publisher;
  retained_messages : list (Topic * Message);}.
```

An inductive type called *Packet* is defined to represent the MQTT messages in more details. The definition is based on the constructors of the type, which enumerate the different cases for messages. Some constructors require arguments to specify additional information. For instance, the SUBSCRIBE message requires specifying the topic to which the client wants to subscribe, as does the UNSUBSCRIBE message.

```
Inductive Packet :=
| CONNECT
| CONNACK
| PUBLISH (topic : string) (payload : string) (qos : nat)
| PUBACK
| SUBSCRIBE (topic : string)
| SUBACK
| UNSUBSCRIBE (topic : string)
| UNSUBACK
| DISCONNECT
| PUBACK
| PUBREC
| PUBREL
| PUBCOMP.
```

B. Specification of the behavior

The system's behavior is specified based on the concept of component states and the transitions between these states. The table II shows the different states that a publisher, subscriber, and broker can go through in MQTT protocol. For example, a publisher starts in the "DisconnectedPub" state and transitions to "ConnectingPub" when it tries to connect to the broker. If the connection is successful, the publisher moves to the "ConnectedPub" state and can then publish messages. The broker manages the communication between publishers and subscribers, ensuring that messages are delivered to all interested subscribers.

TABLE II
MQTT Protocol Component States

Component	From State	Message	To State
Publisher	DisconnectedPub	CONNECT	ConnectingPub
	ConnectingPub	CONNACK	ConnectedPub
	ConnectedPub	PUBLISH	Publishing
	Publishing	PUBACK	ConnectedPub
	ConnectedPub	DISCONNECT	DisconnectedPub
Subscriber	DisconnectedSub	CONNECT	ConnectingSub
	ConnectingSub	CONNACK	ConnectedSub
	ConnectedSub	SUBSCRIBE	Subscribing
	Subscribing	SUBACK	ConnectedSub
	ConnectedSub	PUBLISH	Receiving
	Receiving	PUBACK	ConnectedSub
	ConnectedSub	UNSUBSCRIBE	Unsubscribing
	Unsubscribing	UNSUBACK	ConnectedSub
	ConnectedSub	DISCONNECT	DisconnectedSub
Broker	Idle	CONNECT	ProcessingConnect
	ProcessingConnect	CONNACK	Idle
	Idle	PUBLISH	ReceivingPublish
	Idle	SUBSCRIBE	ManagingSubscriber
	ReceivingPublish	PUBACK	Idle
	ManagingSubscriber	SUBACK	Idle
	Idle	UNSUBSCRIBE	Processing-Unsubscriber
	Processing-Unsubscriber	UNSUBACK	Idle
	Idle	DISCONNECT	DisconnectingClient

The figure 2 shows the state diagram for a publisher. A publisher starts in the "DisconnectedPub" state and transitions to "ConnectingPub" when it tries to connect to the broker. If the connection is successful, the publisher moves to the "ConnectedPub" state and can then publish messages. Once a

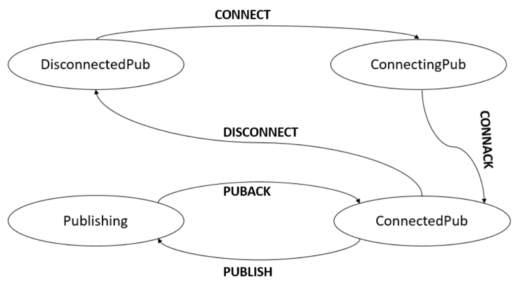

Fig. 2. State Machine for a Publisher

message is published, the publisher transitions to the "Publishing" state and waits for a PUBACK message from the broker. If the PUBACK message is received successfully, the publisher returns to the "ConnectedPub" state and can publish another message. If the connection to the broker is lost, the publisher transitions back to the "DisconnectedPub" state.

The figures for the Subscriber and Broker components follow a similar structure and can be deduced analogously. The following code represents three inductive types for the possible states of the publisher, the subscriber, and the broker respectively. In these definitions, each constructor represent a state of the component.

```
Inductive PublisherState :=
| ConnectedPub
| ConnectingPub
| Publishing
| DisconnectingPub.

Inductive SubscriberState :=
| ConnectedSub
| ConnectingSub
| Subscribing
| Unsubscribing
| Receiving
| DisconnectingSub.

Inductive BrokerState:=
| Idle
| ProcessingConnect
| ReceivingPublish
| ManagingSuscriber
| ProcessingUnsuscriber
| DisconnectingClient.
```

After specifying the states, Coq functions are used to describe how the state of a client (publisher or subscriber) or broker changes in response to incoming messages. The specification, given in the following code, exploits pattern matching to determine the corresponding output state based on the received message and the current state. For instance, in the function *publisher_transition*, the function takes the current state *ps* and a packet *m* as arguments, and produces a publisher state as output. The pattern matching mechanism evaluates the possible cases for the pair *(ps, m)* to determine the appropriate state transition. For example, the first case specifies that if the current state is DisconnectingPub and the message is CONNECT, the output state will be ConnectingPub. Similarly, we define the *subscriber_transition* and the *Broker_transition*.

```
Definition publisher_transition (ps: PublisherState)
(m: Packet) : option PublisherState :=
 match ps, m with
 | DisconnectingPub, CONNECT => Some ConnectingPub
 | ConnectingPub, CONNACK => Some ConnectedPub
 | ConnectedPub, PUBLISH _ _ _ => Some Publishing
 | Publishing, PUBACK => Some ConnectedPub
 | ConnectedPub, DISCONNECT => Some DisconnectingPub
 | _, _ => None
 end.
```

C. *Formal verification*

The proposed formalization enables the verification of both general safety properties and QoS1_related properties. This subsection presents the formal proof of two theorems: a safety property and a QoS1_related property.

1) Safety Property: Safety properties are used in system design and verification to ensure that "something bad will never happen". They guarantee that the system avoids certain undesirable states or behaviors. In the context of the MQTT protocol, one crucial safety property is:

"A client cannot publish without being connected"

This property ensures that a client must establish a connection with the broker before it can publish any messages.The mathematical representation of this safety property is given by:

Theorem: *Safety_Property*

$$\forall \text{ps} \in \text{PublisherState}, \forall m \in \text{Packet},\\ (\text{publisher\_transition}(\text{ps}, m) = \text{Some Publishing})\\ \implies (\text{ps} = \text{ConnectedPub}) \quad (1)$$

This theorem indicates that for all publisher states *ps* and packets *m*, if a Publisher is in the Publishing state after a transition, it must have been in the ConnectedPub state before that transition. The following proof establishes this property:

```
Theorem Safety_Property:
  forall (ps: PublisherState) (m: Packet),
  publisher_transition ps m = Some Publishing
                           ->  ps = ConnectedPub.
Proof. intros ps m H.
  destruct ps; simpl in H.
  - reflexivity.
  - destruct m; simpl in H; try discriminate.
  - destruct m; simpl in H; try discriminate.
  - destruct m; simpl in H; try discriminate. Qed.
```

In this proof, we will systematically apply various tactics to address each component of the goal. For instance, the tactic *intros* is used to introduce the hypothesis for the proof. The proof is based on case analysis of the publisher states with the use of the tactic *destruct*. The tactic *simpl* within the proof is used to simplify calculus where necessary and *discriminate* to handle contradictions.

2) QoS1-related property: QoS 1 is a level of service assurance that guarantees a message is delivered at least once. In the MQTT protocol, a key property related to Quality of Service level 1 (QoS1) is highlighted as follows:

" A published message will be delivered to at least one subscriber "

This property guarantees that every message published by a client will not only receive an acknowledgment from the broker but also be delivered to at least one subscriber.The mathematical representation of this property is given by:

Theorem: *QoS1_related_property*

$$\begin{aligned}
&\forall \mathrm{ps} \in \mathrm{PublisherState}, \forall \mathrm{bs} \in \mathrm{BrokerState},\\
&\forall \mathrm{ss} \in \mathrm{SubscriberState}, \forall \mathrm{m} \in \mathrm{Packet},\\
&(\mathrm{publisher\_transition}(\mathrm{ps}, \mathrm{m}) = \mathrm{Some\ Publishing}) \wedge\\
&(\mathrm{broker\_transition}(\mathrm{bs}, \mathrm{m}) = \mathrm{Some\ ReceivingPublish}) \implies\\
&\exists \mathrm{m\_ack}, (\mathrm{m\_ack} = \mathrm{PUBACK}) \wedge\\
&(\mathrm{publisher\_transition}(\mathrm{Publishing}, \mathrm{m\_ack}) = \mathrm{Some\ ConnectedPub})\\
&\wedge (\mathrm{broker\_transition}(\mathrm{ReceivingPublish}, \mathrm{m\_ack}) = \mathrm{Some\ Idle})\\
&\wedge \exists \mathrm{ss'}, \mathrm{subscriber\_transition}(\mathrm{ConnectedSub}, \mathrm{PUBLISH}\, t\, p\, 1)\\
&= \mathrm{Some\ Receiving} \wedge \mathrm{subscriber\_transition}(\mathrm{Receiving}, \mathrm{PUBACK})\\
&= \mathrm{Some\ ConnectedSub}. \quad (2)
\end{aligned}$$

In essence, the theorem *QoS1_related_property* states that if a publisher transitions to the *Publishing* state, it will eventually receive a *PUBACK*. Meanwhile, the broker will transition to *ReceivingPublish* upon receiving the published packet and will eventually return to the *Idle* state. Additionally, a subscriber will transition to *Receiving* upon receiving the published packet and will then transition back to *ConnectedSub* upon receiving the *PUBACK*. The following formal proof illustrates this property:

```
Theorem QoS1-related property :
  forall (ps: PublisherState) (bs: BrokerState)
  (ss: SubscriberState) (m: Packet),
  publisher_transition ps m = Some Publishing ->
broker_transition bs m = Some ReceivingPublish ->
exists (m_ack: Packet), m_ack = PUBACK /\
 publisher_transition Publishing m_ack = Some ConnectedPub /\
  broker_transition ReceivingPublish m_ack = Some Idle /\
  exists (ss': SubscriberState),
subscriber_transition ConnectedSub (PUBLISH "t" "p" 1)
= Some Receiving
 /\ subscriber_transition Receiving PUBACK = Some ConnectedSub.
Proof. intros ps bs ss m Hpub Hbroker.
  exists PUBACK.
  split.
- reflexivity.
- split.
      + reflexivity.
      + split.
       * reflexivity.
       * exists Receiving.
split; [reflexivity |].
  reflexivity. Qed.
```

The *split* tactic is used to break a conjunction in the goal into two separate subgoals and the *exists* tactic is used to provide a witness for an existential statement. This formal proof confirms that the QoS1-related property holds true, guarantees the **at least once** delivery of a published message, which is a fundamental QoS level in communication protocols.

VI. Conclusions and future work

This paper has presented a formal verification of the IoT communication protocol, MQTT, using the Coq proof assistant. The use of theorem proving in the formal verification of IoT protocols has gained significant interest due to the use of these protocols in critical applications. MQTT is a publish/subscribe-based messaging protocol that is widely utilized for device communication in IoT. Through this study,the protocol's properties were verified using theorem proving, enhancing the trust in its reliability. The formalization of the MQTT protocol in this work elaborated on the key concepts using Coq's robust type system, which offers inductive types, function definitions, and essential built-in data structures, such as lists. This formalization enables the expression of important features of the protocol through Coq theorems. This paper represents a pioneering and successful case study in utilizing the Coq proof assistant to establish properties of the MQTT protocol. This work is ongoing. Future work will focus on extending the formalization to cover all Quality of Service (QoS) levels of the protocol and to obtain a certified version of the protocol through Coq's extraction mechanism.

References

[1] Ahmed J Hintaw, Selvakumar Manickam, Mohammed Faiz Aboalmaaly, and Shankar Karuppayah. Mqtt vulnerabilities, attack vectors and solutions in the internet of things (iot). *IETE Journal of Research*, 69(6):3368–3397, 2023.

[2] Katharina Hofer-Schmitz and Branka Stojanović. Towards formal methods of iot application layer protocols. In *2019 12th CMI conference on cybersecurity and privacy (CMI)*, pages 1–6. IEEE, 2019.

[3] Katharina Hofer-Schmitz and Branka Stojanović. Towards formal verification of iot protocols: A review. *Computer Networks*, 174, 2020.

[4] Erick Simas Grilo and Bruno Lopes. Formalization and certification of software for smart cities. In *2018 International Joint Conference on Neural Networks (IJCNN)*, pages 1–8. IEEE, 2018.

[5] Muhammad Saqib Nawaz, Meng Sun, Basit Shahzad, M Ikram Ullah Lali, Tariq Umer, and Shaohua Wan. Quality of service in iot protocol as designs and its verification in pvs. *Transactions on Emerging Telecommunications Technologies*, 33(8):e3742, 2022.

[6] Manel Houimli, Laid Kahloul, and Sihem Benaoun. Formal specification, verification and evaluation of the mqtt protocol in the internet of things. In *2017 International conference on mathematics and information technology (ICMIT)*, pages 214–221. IEEE, 2017.

[7] Sabina Akhtar and Ehtesham Zahoor. Formal specification and verification of mqtt protocol in pluscal-2. *Wireless Personal Communications*, 119:1589–1606, 2021.

[8] MOHAMMAD NURUS SALAM. *Modelling of a Communication Protocol for IoT based Applications*. PhD thesis, DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING, 2021.

[9] Maithily Diwan and Meenakshi D'Souza. A framework for modeling and verifying iot communication protocols. In *Dependable Software Engineering. Theories, Tools, and Applications: Third International Symposium*, pages 266–280. Springer, 2017.

[10] Abdullah Ahmed Omar Bahashwan and Selvakumar Manickam. A brief review of messaging protocol standards for internet of things (iot). *Journal of Cyber Security and Mobility*, pages 1–14, 2019.

[11] Eyhab Al-Masri, Karan R. Kalyanam, John Batts, Jonathan Kim, Sharanjit Singh, Tammy Vo, and Charlotte Yan. Investigating messaging protocols for the internet of things (iot). *IEEE Access*, 8, 2020.

[12] Dipa Soni and Ashwin Makwana. A survey on mqtt: a protocol of internet of things (iot). In *International conference on telecommunication, power analysis and computing techniques*, volume 20, 2017.

[13] The Coq Development Team. The coq proof assistant. https://coq.inria.fr/, 2024. Accessed: 2024-08-09.