Received 24 April 2024, accepted 24 June 2024, date of publication 2 July 2024, date of current version 10 July 2024.

Digital Object Identifier 10.1109/ACCESS.2024.3421962

RESEARCH ARTICLE

Finite State Machine of the MQTT-SN Protocol for Its Operation Over IEEE 802.15.4 in Linear Topologies

LUIS CRIOLLO CAJAMARCA [1], CARLOS EGAS ACOSTA [1], CHRISTIAN TIPANTUÑA [1], JORGE CARVAJAL-RODRIGUEZ [1], AND CARLA PARRA [2]

[1]Departamento de Electrónica, Telecomunicaciones y Redes de Información, Escuela Politécnica Nacional, Quito 170525, Ecuador
[2]Departamento de Estudios Organizacionales y Desarrollo Humano, Escuela Politécnica Nacional, Quito 170525, Ecuador

Corresponding author: Christian Tipantuña (christian.tipantuna@epn.edu.ec)

This work was supported by the Escuela Politécnica Nacional.

ABSTRACT Currently, the Message Queuing Telemetry Transport for Sensor Networks (MQTT - SN) protocol has been implemented in operating systems such as TinyOS and Contiki to operate above the network layer, in sensor nodes with low processing capacity, powered by batteries and operating in various topologies. However, minimizing the processes at the node side is necessary to make the MQTT - SN protocol applicable in wireless sensor networks (WSNs) with large-scale linear structures. This paper presents the development of the MQTT-SN protocol's finite state machine (FSM) for its operation over IEEE 802.15.4 in linear topologies. For this purpose, an FSM for each of the twelve procedures indicated in the MQTT-SN specification is obtained. Furthermore, each FSM is represented using the specification and description language, and the characteristics of the IEEE 802.15.4 protocol and the sensor node are considered. Through simulation and the exchange of messages between nodes, the operation of the FSMs is verified. Subsequently, implementing the FSMs in WSNs allows for validating the deployment of the MQTT-SN protocol in linear topologies.

INDEX TERMS MQTT-SN, message queuing telemetry transport for sensor networks, IEEE802.15.4, finite state machine, linear topology, wireless sensor networks.

I. INTRODUCTION

Currently, the use of wireless sensor networks (WSNs) has experienced an increase [1]. With this, the need has arisen to develop technologies that require using the Message Queuing Telemetry Transport for Sensor Networks (MQTT-SN) protocol [2]. However, to date, no implementation of the MQTT-SN protocol directly over IEEE 802.15.4 is available. The MQTT-SN protocol has been implemented in operating systems on the network layer. To ensure that the MQTT-SN protocol is applied in WSNs, minimizing the processes carried out in the sensor nodes is necessary [3]. Therefore, this paper develops the finite state machines of the MQTT-SN protocol for operation over IEEE 802.15.4 in linear topologies using the specification and description language (SDL) [4]. With this development, a sender node can be encoded to respond to sequences of 802.15.4 frames containing MQTT-SN messages. Implementing the MQTT-SN FSMs operating over IEEE 802.15.4 allows verifying that it is possible to use MQTT-SN over the link level in wireless networks and in a linear topology where routing functions are minimal [5].

The associate editor coordinating the review of this manuscript and approving it for publication was Hongwei Du.

II. BACKGROUND

A. LINEAR WSNS

Linear WSNs are a specific type of WSN in which the nodes are organized in a straight line, as seen in Fig. 1. This topology aims to reduce installation and maintenance costs, improve network stability and fault tolerance, extend sensor battery life, and reduce end-to-end communication latency to improve the quality of service (QoS) of sensitive data [6], [7]. The linear alignment of sensor nodes can be applied in

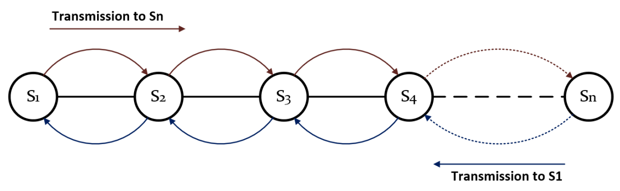


FIGURE 1. Linear wireless sensor network [7].

various situations, such as monitoring and surveillance of international borders to detect illegal crossings or smuggling operations, monitoring highways, or monitoring pipelines that transport oil, as has been analyzed in our previous work in [8], [9], and [10].

The transport of the MQTT-SN protocol over IEEE 802.15.4 frames in large-scale linear infrastructures with hundreds of nodes considers typical transactional traffic for monitoring linear structures. Potential packet loss due to congestion issues stemming from the low computational capacity of nodes when multiple IEEE 802.15.4 frames circulate through the linear topology is addressed in [8]. This paper presents an algorithm for reliable data transport using IEEE 802.15.4 between a sensor node in the linear topology and a border node, eliminating the need for a routing protocol.

Implementing the MQTT-SN protocol over the IEEE 802.15.4 protocol considers a routing protocol unnecessary when nodes are stationary in a linear topology. Therefore, the same MAC identifier can be used to identify the node within the multi-hop network, as studied in [11]. In dynamic linear topologies where nodes are in motion, a node may transition from one coverage area to another, necessitating network-level identifiers. Therefore, MQTT must be encapsulated in a protocol such as 6LoWPAN. Furthermore, in the scenario in which linear WSNs are used for monitoring large-scale linear infrastructures, nodes have equal priority in transmitting sensing messages because the parameters to be monitored are the same across all nodes and are equally important. This study does not consider nodes that have prioritized information to transmit.

The implemented scenarios consider multi-hop linear topologies, where each node's coverage zone spans nodes to the right and left. Monitoring large-scale linear infrastructures involves transactional traffic and monitoring data that can be transported within the payload of a single IEEE 802.15.4 frame.

B. MQTT-SN PROTOCOL

The MQTT-SN protocol is considered an adaptation of the MQTT protocol designed for resource-constrained devices with low processing power, low memory, and requiring batteries, making it ideal for WSNs. Any network that provides bidirectional data transfer service supports MQTT-SN. For this reason, TCP/IP is not essential for the operation of MQTT-SN [12]. In general terms, MQTT-SN can be defined as an optimized publish/subscribe protocol for WSNs [13].

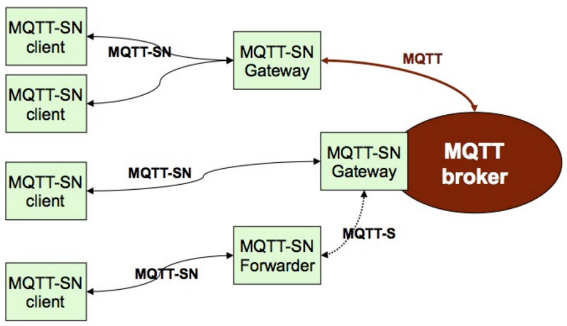


FIGURE 2. MQTT-SN architecture [12].

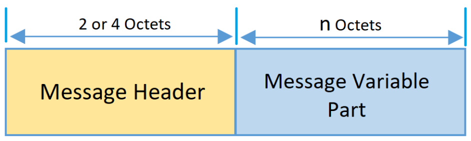


FIGURE 3. MQTT-SN message format [12].

1) MQTT-SN ARCHITECTURE

Figure 2 shows the architecture used by MQTT-SN [12]. MQTT-SN must work in conjunction with MQTT, so three components are defined:

- **MQTT-SN clients:** They are responsible for publishing and subscribing to the MQTT broker, although they cannot connect to it directly [12].
- **MQTT-SN forwarder:** MQTT-SN clients can access a gateway through a forwarder. This component encapsulates the MQTT-SN messages in a forwarding frame to send them to the gateway. When the forwarder receives an encapsulated frame, it decapsulates it and sends it to the client [12].
- **MQTT-SN gateway:** It operates as an intermediary between the client and the broker, converting MQTT-SN communication messages to MQTT. A gateway can be integrated into the broker or be independent [12]. A stand-alone gateway must convert MQTT-SN messages to MQTT and vice versa because it uses MQTT-SN messages to communicate with clients and MQTT messages to communicate with the broker [12].

2) MQTT-SN MESSAGE FORMAT

An MQTT-SN message comprises a fixed header and a variable part, as shown in Fig. 3. The fixed header is mandatory, and its fields are identical for all messages, while the variable part and its fields depend on the type of MQTT-SN message [12].

- **MQTT-SN message header:** The message header consists of the Length field and the MsgType field, as seen in Fig. 4. The Length field indicates the total octets of the message, including the field size. Its length

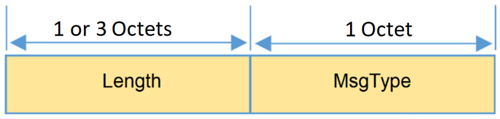
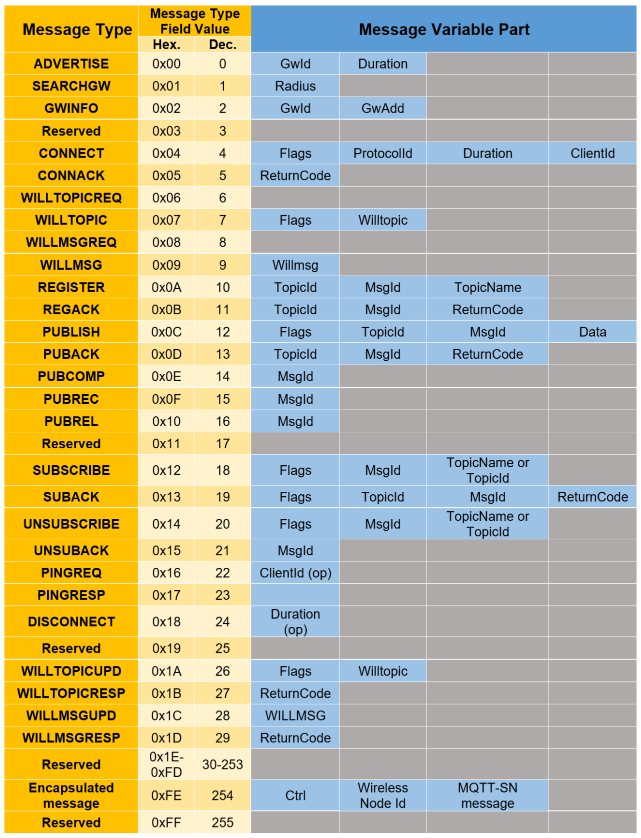


FIGURE 4. MQTT-SN message header [12].



Message Type	Message Type Field Value		Message Variable Part			
	Hex.	Dec.				
ADVERTISE	0x00	0	GwId	Duration		
SEARCHGW	0x01	1	Radius			
GWINFO	0x02	2	GwId	GwAdd		
Reserved	0x03	3				
CONNECT	0x04	4	Flags	ProtocolId	Duration	ClientId
CONNACK	0x05	5	ReturnCode			
WILLTOPICREQ	0x06	6				
WILLTOPIC	0x07	7	Flags	Willtopic		
WILLMSGREQ	0x08	8				
WILLMSG	0x09	9	Willmsg			
REGISTER	0x0A	10	TopicId	MsgId	TopicName	
REGACK	0x0B	11	TopicId	MsgId	ReturnCode	
PUBLISH	0x0C	12	Flags	TopicId	MsgId	Data
PUBACK	0x0D	13	TopicId	MsgId	ReturnCode	
PUBCOMP	0x0E	14	MsgId			
PUBREC	0x0F	15	MsgId			
PUBREL	0x10	16	MsgId			
Reserved	0x11	17				
SUBSCRIBE	0x12	18	Flags	MsgId	TopicName or TopicId	
SUBACK	0x13	19	Flags	TopicId	MsgId	ReturnCode
UNSUBSCRIBE	0x14	20	Flags	MsgId	TopicName or TopicId	
UNSUBACK	0x15	21	MsgId			
PINGREQ	0x16	22	ClientId (op)			
PINGRESP	0x17	23				
DISCONNECT	0x18	24	Duration (op)			
Reserved	0x19	25				
WILLTOPICUPD	0x1A	26	Flags	Willtopic		
WILLTOPICRESP	0x1B	27	ReturnCode			
WILLMSGUPD	0x1C	28	WILLMSG			
WILLMSGRESP	0x1D	29	ReturnCode			
Reserved	0x1E-0xFD	30-253				
Encapsulated message	0xFE	254	Ctrl	Wireless Node Id	MQTT-SN message	
Reserved	0xFF	255				

FIGURE 5. Values that the MsgType field can acquire, together with its respective variable parts, in an MQTT-SN message [12].

can be 1 or 3 octets. The 1-octet format is used for messages with a length less than or equal to 255 octets. For messages longer than 256 octets, the 3-octet format is used. In this case, the first octet of the length field is encoded with the value 0×01, while the remaining two octets indicate the total number of octets of the message. The MsgType field is used to identify any MQTT-SN message uniquely [12]. It has a length of 1 octet, and its values can be seen in Fig. 5.

- **Variable part of the message:** It can consist of 14 fields whose use and distribution depend on the message type, as shown in Fig 5 [14]. The Flags and ReturnCode fields are described below.
  - **Flags (1 byte):** It contains six flags, which are indicated in Fig. 6) and are described below:
    1) **DUP:** Used in PUBLISH messages to indicate whether the message is transmitted for the first time (0) or retransmitted (1).

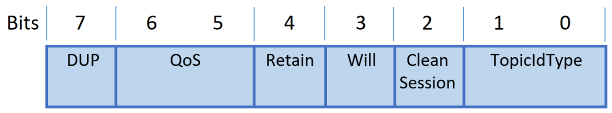


FIGURE 6. MQTT-SN flags field [12].

TABLE 1. Values and meaning of the Return Code field.

Value	Meaning
0x00	Accepted
0x01	Rejected: congestion
0x02	Rejected: invalid topic ID
0x03	Rejected: not supported
0x04 - 0xFF	Reserved

    2) **QoS:** Indicates the quality of service level: QoS 0 (0b00), QoS 1 (0b01), QoS 2 (0b10), and QoS -1 (0b11); the latter is specific to MQTT-SN and is used in publications that do not require prior procedures.
    3) **Retain:** Used in PUBLISH messages for the client to tell the broker to replace any existing held messages and store the new received message.
    4) **Will:** Used in CONNECT messages sent by the client, indicates that a Will topic and a Will message will be sent.
    5) **CleanSession:** If set to 0, it tells the broker to resume communication with the client as long as it has an associated session. If set to 1, the client and server must discard previous sessions and start a new one, which will be deleted when the network connection ends. CleanSession is used only in CONNECT messages.
    6) **TopicIdType:** In an MQTT-SN message, the value of the TopicId or TopicName field can be a regular topic identifier, a predefined topic identifier, or a short topic name.
  - **ReturnCode (1 byte):** Used to accept or reject a message awaiting an acknowledgment, the values and meanings of which are shown in Table 1.

C. FINITE STATE MACHINE

A finite state machine (FSM) or finite automaton is an abstract model for the manipulation of symbols that allows us to know if a chain of symbols belongs to a language or, in turn, generates another set of symbols [15], [16]. An FSM comprises a set of states that includes an initial and final state. Furthermore, it depends on the string entered and the state changes on the FSM. An FSM is composed of five parts and can be defined as a quintuple: $A = \{Q, q_0, F, \Sigma, \delta\}$, where:

Q : Finite set of states.

q_0 : Initial state with $q_0 \in Q$.

F : Set of final states.

Σ : Finite input alphabet.

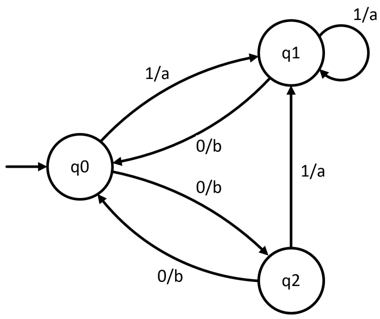


FIGURE 7. Mealy machine diagram [19].

$$\delta : \text{Transition function } Q \times \Sigma \rightarrow Q.$$

If the FSM is in state q_i (where $q_i \in Q$) and enters symbol a (where $a \in \Sigma$), it causes the FSM to change from state q_i to state q_k. The function δ, called the transition function, describes this change in the form $\delta(q_i, a) \rightarrow q_k$. In this way, a new state is obtained. Another way to represent an FSM is through a state diagram [15].

1) MEALY MACHINE

A derivation of the FSMs is the transducer state machines (finite-state transducers), which deliver a set of symbols that belong to a language. The set of final states is changed by an output function, which takes the current state or a transition of the FSM as a parameter and returns an element of the set of output symbols [17], [18]. An example of this type of FSM is the Mealy machine. The Mealy machine requires an initial state q_0; the output function λ, which returns a symbol s as soon as a state transition occurs, and the transition function δ, which reads an element of the input string Σ and indicates the new state [19], [20]. A tuple or a state diagram can represent a Mealy machine, as shown in Fig. 7. A Mealy machine is defined as a 6-tuple: $C = \{Q, \Sigma, S, \delta, \lambda, q_0\}$, where:

$$\begin{aligned}
&Q : \text{Finite set of states.}\\
&\Sigma : \text{Input alphabet.}\\
&S : \text{Output alphabet.}\\
&\delta : \text{Transition function } Q \times \Sigma \rightarrow Q.\\
&\lambda : \text{Output function } Q \times \Sigma \rightarrow S,\\
&\quad \text{with } \lambda(q_i, a) \rightarrow s, \ \text{con } s \in S, q \in Q \text{ y } a \in \Sigma.\\
&q_0 : \text{Initial state.}
\end{aligned}$$

D. SPECIFICATION AND DESCRIPTION LANGUAGE

The SDL is based on finite state machines and was originally developed to specify and describe the functional behavior of telecommunications systems [4], [21], [22]. This language describes a system's structure, communication, behavior, and data. It is also used to specify communication protocols [23]. Two types of SDL notations can be selected: i) graphical representation (SDL/GR) and ii) phrase representation (SDL/PR).

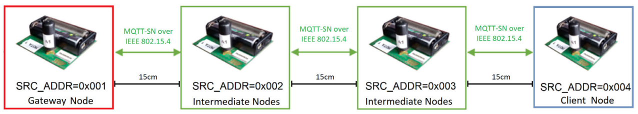


FIGURE 8. IEEE 802.15.4 linear topology wireless network.

III. CONSIDERATIONS FOR THE DEVELOPMENT OF THE FSM

Because wireless nodes, in this case, the ATMEL RCB256RFR2 [24], have their operating characteristics, determining whether any of them affect the development of the FSM is essential. This section analyses Features such as the data reception and transfer service, the operation of the nodes in linear topologies, the programming language used by the node, the operation of the MQTT-SN protocol over IEEE 802.15.4, and the interaction between the broker and the gateway.

A. NODE DATA TRANSFER AND RECEPTION SERVICE

As specified in the characteristics of the RCB256RFR2 node [24], it cannot receive and send information at the same time, so it is considered that the reception of a message should be the trigger for a state change. In response to receiving a message, the node can send another depending on its procedure and state. When the node is in a certain state, it can receive different types of messages, but only one message at a time. After reception and transmission, if applicable, the node must go to another state or return to the same state.

B. NODE PROGRAMMING LANGUAGE

The RCB256RFR2 node uses the C programming language, so the device cannot run multithreaded processes [24]. Because of this, the node can only run one process at a time, being another reason why the node cannot receive and send multiple messages simultaneously. Since the C language does not allow object-oriented programming, generating MQTT-SN messages uses vector pooling. As they are simple messages, they do not take up a large amount of device resources. For this reason, the node remains in a certain state until it receives a message or signal allowing a state change. The C language allows this functionality to be coded without any problem within the node; therefore, there should be no problem in the development of the FSM either.

C. NODE OPERATION ON LINEAR TOPOLOGIES

For the development of the FSM of the MQTT-SN protocol, only two nodes are considered capable of handling the states of each MQTT-SN procedure. The mentioned nodes must be at the network's edge; thus, a network with linear topology is formed, as shown in Fig. 8. The intermediate nodes only verify if the MQTT-SN message type is correct and then send it to the next node until it reaches its destination.

The MQTT-SN protocol, designed for WSNs, operates directly on the link layer, not requiring a network layer because the information is sent only through a single route [3], [5]. The RCB256RFR2 node allows direct access to the payload field of an IEEE 802.15.4 frame [24]. Therefore, the gateway node and the publisher/subscriber (client) node can generate MQTT-SN messages and send them through the intermediate nodes. Because the nodes work on a linear topology, functions are omitted in the announcement and discovery procedure. The ability to respond to SEARCHGW messages by nodes other than the gateway is not considered because the publisher/subscriber node receives the ADVERTISE messages through the intermediate nodes in the proposed network topology. The timers with which the response priority is given to the gateway are also not taken into account; the whole procedure is not omitted because the constant sending of ADVERTISE messages allows the client to know the gateway's status. It should also be mentioned that when working on the proposed linear topology, it is assumed that there are no problems due to the transmission or reception of simultaneous messages.

D. OPERATION OF THE MQTT-SN PROTOCOL OVER IEEE 802.15.4

MQTT-SN was originally developed to run over ZigBee, which has selected the IEEE 802.15.4 standard as the protocol for the PHY and MAC layers, thus providing interoperability between products from different vendors [14], [25]. MQTT-SN is designed to be independent of the lower layers it operates on. A network providing a bidirectional data transfer service between any node and a particular node (gateway) must support MQTT-SN [12]. Thus, encapsulating MQTT-SN messages in IEEE 802.15.4 frames should not affect the protocol's operation.

E. INTERACTION BETWEEN THE MQTT BROKER AND THE MQTT-SN GATEWAY

No FSM corresponding to the MQTT broker is developed in this paper. However, this device's operation must be considered when creating FSMs, especially gateway-related functionalities. In an MQTT-SN network, the gateway must periodically announce its presence. The gateway must establish a connection with the broker if it is independent. In contrast, if the gateway is integrated with the broker, it announces its presence when it starts its operation. In both cases, a signal is established that the broker is ready to receive messages for FSM development. On the other hand, by not using a broker in the proposed MQTT-SN network, it is considered that the gateway responds to messages sent by the client in a similar way as it would if it were connected to a broker, and the time it should take for the broker to respond is not taken into account.

F. MQTT-SN MESSAGES TO BE EXCHANGED BETWEEN CLIENT AND GATEWAY

The messages are considered input and output signals for the developed FSMs. Both the client and gateway nodes can generate their respective MQTT-SN messages. In addition, in several cases, a device must generate a message in response to receiving another message sent by another device. To develop the FSMs of the MQTT-SN protocol, all the messages of the MQTT-SN specification except the MQTT-SN encapsulation frame must be used. The messages must be sent and received according to the procedure being executed in the device using them.

G. OTHER EVENTS AFFECTING THE DEVELOPMENT OF THE FSMS

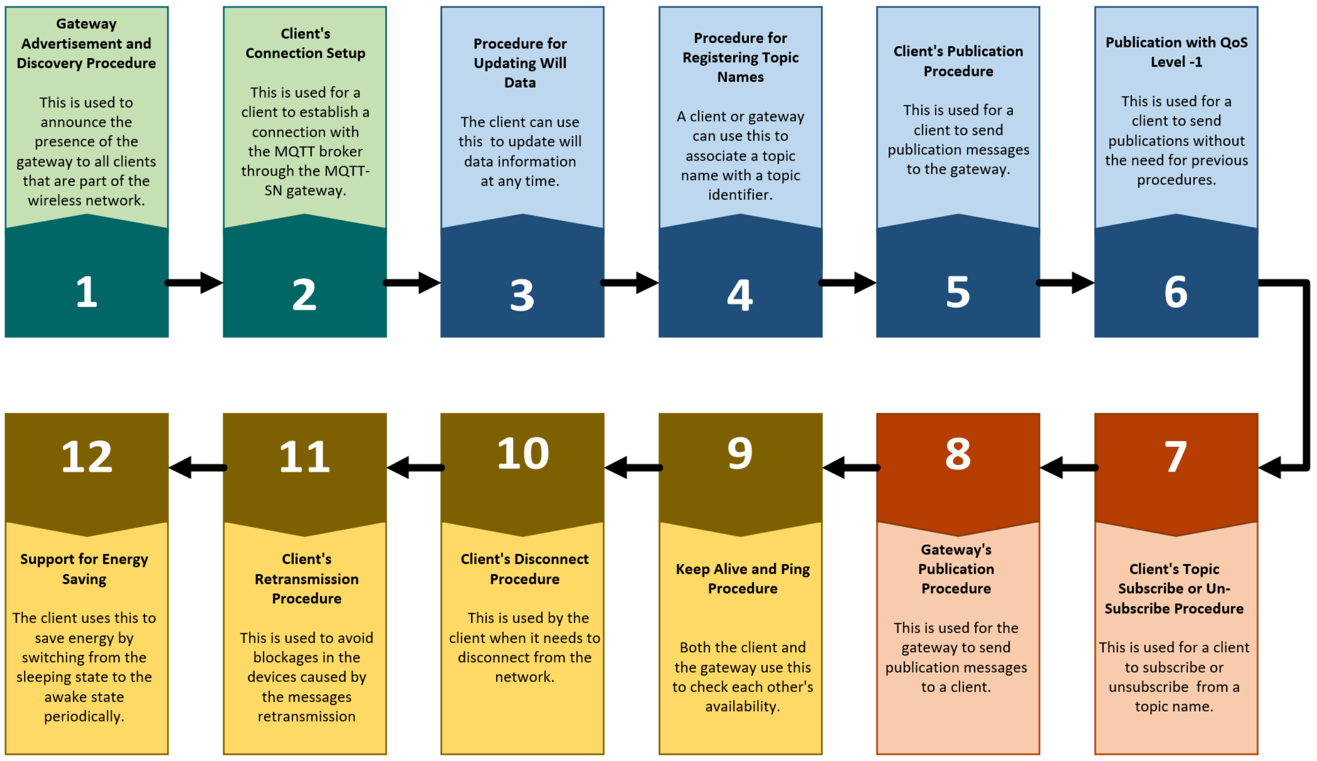
Apart from the reception of MQTT-SN messages, a state change is also possible when the time of a previously set timer has expired. In addition, a status change can occur by an internal signal generated within the node itself. For example, when the client has a new publication-ready, the message PUBLISH should signal that it is ready to be published. The pushbutton will be used on the RCB256RFR2 node board to simulate these internal signals. On the other hand, messages that have a returnCode field or the flags field (QoS and will) also cause a state change.

IV. DEVELOPMENT OF THE FSMS OF THE MQTT-SN PROTOCOL

Given that in most of the procedures, the devices involved respond to the reception of a message with another message, the Mealy machine represents the states through which the client and gateway nodes pass. The Mealy machine allows accepting the input of different MQTT-SN message sequences and, in turn, sending a sequence of output MQTT-SN messages. Considering the 12 procedures for operating the MQTT-SN protocol, it is chosen to develop a Mealy machine (both for the client and the gateway) for each procedure independently because it will be easier to understand each FSM obtained. In addition, the message sequences allowed for use are shorter and depend on each procedure. Similarly, it is easier to test the operation of one procedure at a time rather than all procedures together. Fig. 9 shows an overview of all procedures specified in MQTT-SN specification.

A. GATEWAY ANNOUNCEMENT AND DISCOVERY PROCEDURE

A gateway must establish a connection with a broker and then send ADVERTISE messages to all devices in the MQTT-SN network. If the gateway is part of the broker, it will announce its presence instantly. ADVERTISE messages are sent periodically. Each period will have a duration TADV, indicated in the "Duration" field of the message. If new clients join the network, they can wait for ADVERTISE messages or send SEARCHGW messages. Each client will wait a random time, between 0 and TSEARCHGW, before transmitting a SEARCHGW message. In response, the gateway will send a GWINFO message indicating the device is active. SEARCHGW messages can be retransmitted when there is no response to their sending. Each time interval between two consecutive SEARCHGW messages



1
Gateway Advertisement and Discovery Procedure
This is used to announce the presence of the gateway to all clients that are part of the wireless network.
2
Client's Connection Setup
This is used for a client to establish a connection with the MQTT broker through the MQTT-SN gateway.
3
Procedure for Updating Will Data
The client can use this to update will data information at any time.
4
Procedure for Registering Topic Names
A client or gateway can use this to associate a topic name with a topic identifier.
5
Client's Publication Procedure
This is used for a client to send publication messages to the gateway.
6
Publication with QoS Level -1
This is used for a client to send publications without the need for previous procedures.
7
Client's Topic Subscribe or Un-Subscribe Procedure
This is used for a client to subscribe or unsubscribe from a topic name.
8
Gateway's Publication Procedure
This is used for the gateway to send publication messages to a client.
9
Keep Alive and Ping Procedure
Both the client and the gateway use this to check each other's availability.
10
Client's Disconnect Procedure
This is used by the client when it needs to disconnect from the network.
11
Client's Retransmission Procedure
This is used to avoid blockages in the devices caused by the messages retransmission
12
Support for Energy Saving
The client uses this to save energy by switching from the sleeping state to the awake state periodically.

FIGURE 9. Overview of the description of the 12 procedures indicated in the MQTT-SN specification [12].

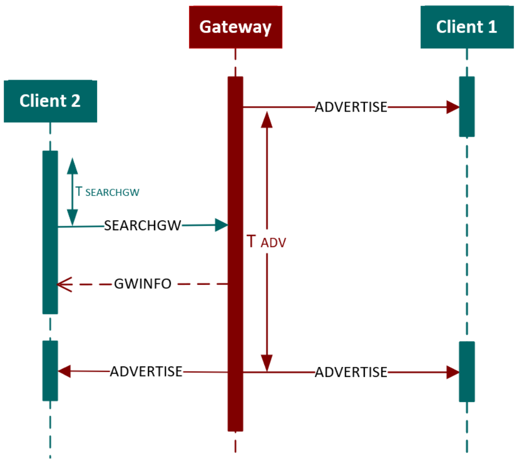


FIGURE 10. Gateway announcing its presence to two clients.

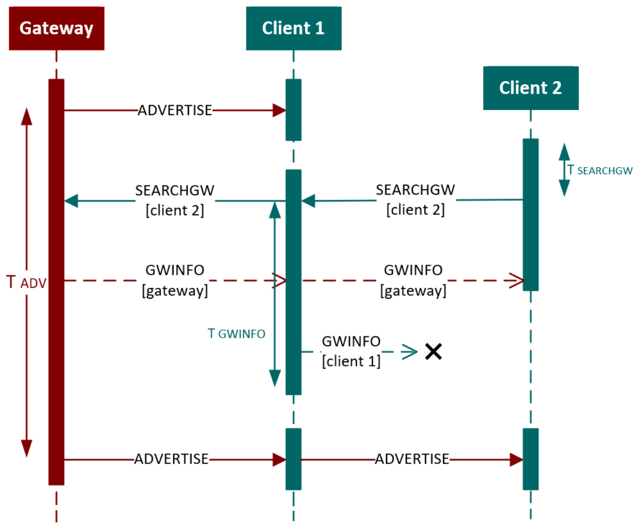


FIGURE 11. Client requesting information about an active gateway from another client.

must increase exponentially. Fig. 10 shows two clients discovering an active gateway. Client 1 only waits for the warning message from the gateway, while client 2 requests a GWINFO. After these interactions, both clients wait for the following ADVERTISE messages.

All clients contain a list that stores active gateway identifiers. The list is kept updated with information from ADVERTISE and GWINFO messages sent by the gateway. During an interaction between clients, a client can reply to a SEARCHGW message with its respective GWINFO if its gateway list contains information for at least one active gateway. The client selects an active device from the list and sends its address and ID within a GWINFO. The transmission of GWINFO messages by the client must be delayed by TGWINFO time because the gateway has priority in responding to a SEARCHGW. Fig. 11 shows the sequence diagram, where client 2 wants to know the existence of the active gateway. Both client 1 and the gateway receive the SEARCHGW message. However, this client cancels sending its response because the gateway has priority in responding.

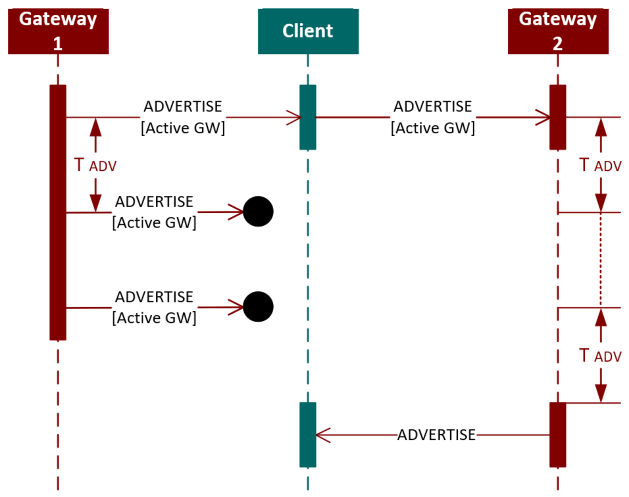


FIGURE 12. **Operation of an active and inactive gateway.**

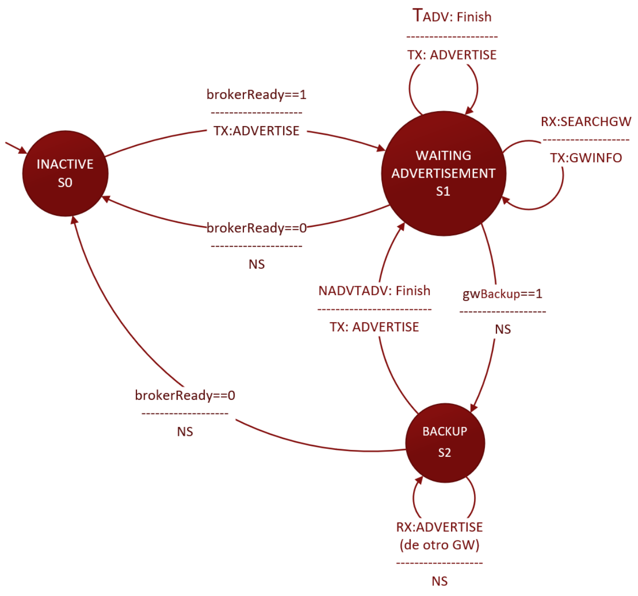


FIGURE 13. **State diagram for the gateway, representing the announcement and discovery procedure.**

Although the network may operate with multiple gateways, a client can connect to only one. If the connection fails, the client will look for another gateway. An MQTT-SN network can support several inactive gateways that do not send messages. An inactive gateway only allows the reception of ADVERTISE messages from other active gateways. As seen in the sequence diagram in Fig.12, the client and gateway 2 (inactive) receive the announcement message from Gateway 1 (active). However, the latter's messages are lost NADV times, causing gateway 2 to wake up and send its ADVERTISE messages. Because multiple active gateways can coexist on the network, clients must receive their ADVERTISE messages repeatedly every so often TADV. If clients stop receiving messages a certain number of times (NADV times) from a given gateway, their information will be removed from all lists.

1) DESIGN OF THE MEALY MACHINE FOR GATEWAY OPERATION

The state machine diagram for this procedure indicates that the device can remain in three states: i) INACTIVE, ii) WAITING ADVERTISEMENT, and iii) BACKUP, as shown in Fig. 13).

When the MQTT broker is ready to receive and transmit messages, the gateway node will also be prepared to start its operation if the latter is part of the broker. Otherwise, both devices must establish a connection for the gateway to operate. Therefore, the node remains in the INACTIVE state until the broker, through the input signal $brokerReady = 1$, tells the gateway that it can start operating. Once the input signal is received, the device responds with the message *ADVERTISE*. When the device reaches the WAITING ADVERTISEMENT state, the gateway periodically sends *ADVERTISE* messages when the timer *TADV* expires. Additionally, during this state, the gateway can receive *SEARCHGW* messages to respond with a *GWINFO* immediately. This way, the node can announce its presence in an MQTT-SN network. Also, there is a state called BACKUP, which the node will reach if it receives the signal $gwBackup = 1$ when in the WAITING ADVERTISEMENT state. During this state, the node waits for *ADVERTISE* messages from another nearby gateway, which must announce its presence periodically. The gateway in the BACKUP state will return to the WAITING ADVERTISEMENT state if the *NADVTADV* timer expires after it stops receiving *ADVERTISE* messages from another gateway. Furthermore, from the INACTIVE state it is not possible to go to the backup state directly, you must wait for the broker to indicate that it is ready to work. Finally, a Gateway will go to the INACTIVE state, from any state, if the broker, for any reason, stops working, for which a $brokerReady = 0$ signal is used.

The diagram in Fig. 14 is similar to the Mealy machine of the corresponding procedure, with the difference that here the timers *TADV* and *NADVTADV* are added that start when a message is sent or received *ADVERTISE*. The variables `durationRX`, `durationTX`, and `NADV` are added to control the duration of the timers. The diagram better describes the state changes described in Mealy's machine.

2) DESIGN OF THE MEALY MACHINE FOR THE CLIENT'S OPERATION

Although an MQTT-SN client must be able to respond to messages from other clients, this feature was not considered for developing this Mealy machine because it is unnecessary if the node operates on a linear topology. Due to the above, only two states will be required: i) DISCOVERY and ii) RX_GWINFO, as shown in Fig. 15.

The initial state is the Discovery state, during which the client node waits to receive *ADVERTISE* messages from the gateway node, also waiting for the *NADVTADV* or

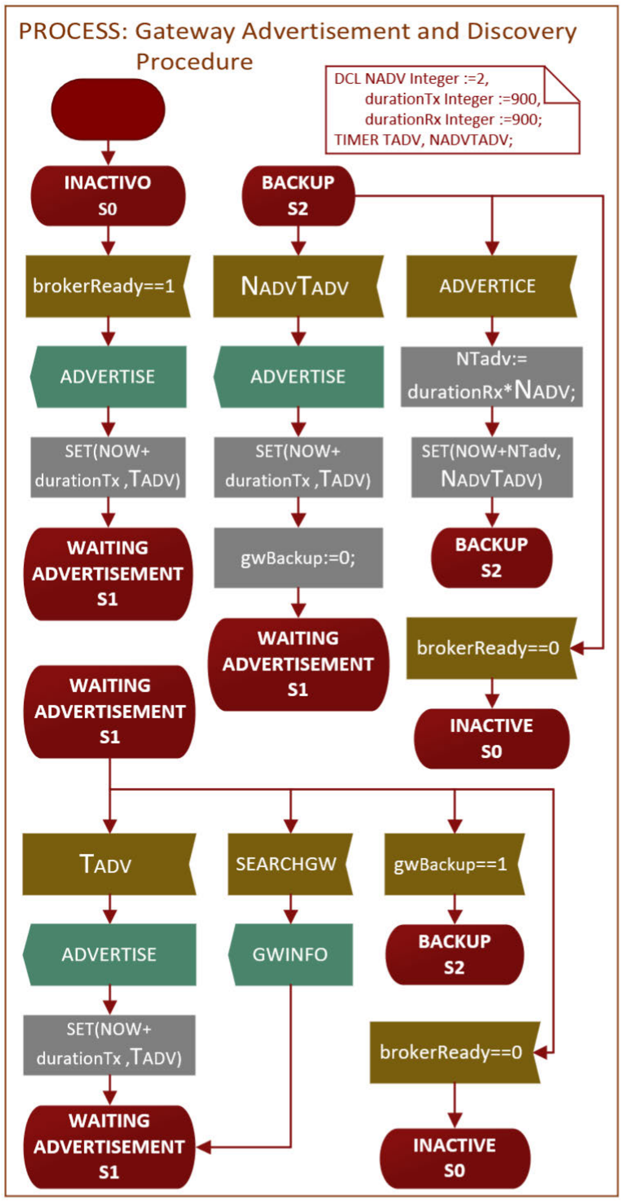


FIGURE 14. SDL process for the gateway, representing the gateway advertisement and discovery procedure.

TSEARCHGW timers to expire. When the *TSEARCHGW* timer expires, a *SEARCHGW* message is sent, and the node enters the RX_GWINFO state. In the mentioned state, the node waits for the *GWINFO* message to return to the initial state and continues waiting for warning messages from the gateway. Conversely, if the node does not receive the expected message, it must retransmit the previously sent *SEARCHGW* message.

Figure 16 shows how the timers *NADTADV*, *TSEARCHGW*, and *TRTX* are added to the SDL process. The *NADTADV* and *TSEARCHGW* timers are started when an *ADVERTISE* is received or a *SEARCHGW* is sent. Fig. 16 shows that the timer *TSEARCHGW* is configured before starting the state transition and how the timer *TRTX*

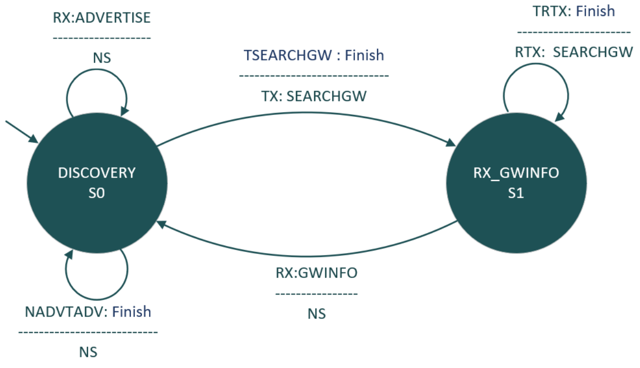


FIGURE 15. State diagram for the client, representing the gateway announcement and discovery procedure.

increases the duration when the client does not receive a response from the gateway. The SDL process also adds the variables `tSEARCHGW, NADV, durationRx, tini, tRTX` and `NTadv` to control the duration of the timers. On the other hand, the variable `tablaGwUpdated` indicates that when the correct sequence of messages has been exchanged, a client's gateway table should be updated.

B. CLIENT'S CONNECTION SETUP

Before a client can exchange information with a gateway, it must establish a connection. During this procedure, a client must send a CONNECT message to the gateway to which it wishes to connect. The message tells the gateway whether or not the client wants to send *will* data via the corresponding flag. If the client requires sending a topic and a will message, it will set the *will* flag to 1. After the gateway receives the CONNECT message, it will request the topic and the *will* message using the WILLTOPICREQ and WILLMSGREQ messages, respectively. The client will send the data *will* by responding with the messages WILLTOPIC and WILLMSG, respectively. Finally, the gateway will send a CONNACK message accepting or rejecting the connection. If the client sets the *will* flag of the CONNECT message to 0, the gateway will immediately respond with a CONNACK message. Fig. 17 shows two clients establishing a connection with a gateway with a different value of the *will* flag.

1) DESIGN OF THE MEALY MACHINE FOR GATEWAY OPERATION

This procedure starts when the previous procedure has sent at least one warning message. Consequently, the state of WAIT ADVERTISE is renamed WAIT CONECTION and is considered the initial state of this procedure. Additionally, the states RX_TOPIC and RX_MSG are considered, which are used according to the setting of the will flag, as shown in Fig. 18. In the initial state, the node waits for *CONNECTmessages*. After responding with a *CONNACK* message, it remains in this state if the received message contains the will flag with a zero value. However, a state change occurs, and a request message *WILLTOPICREQ* is sent if a message with the will

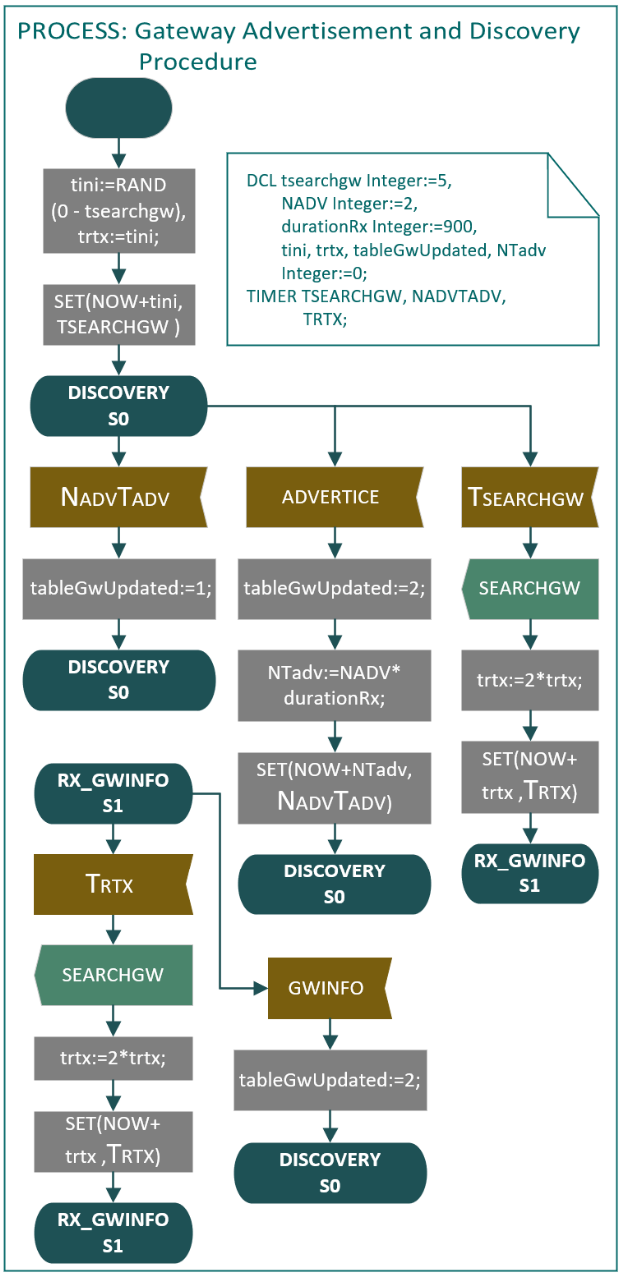


FIGURE 16. SDL process for the client, representing the gateway advertisement and discovery procedure.

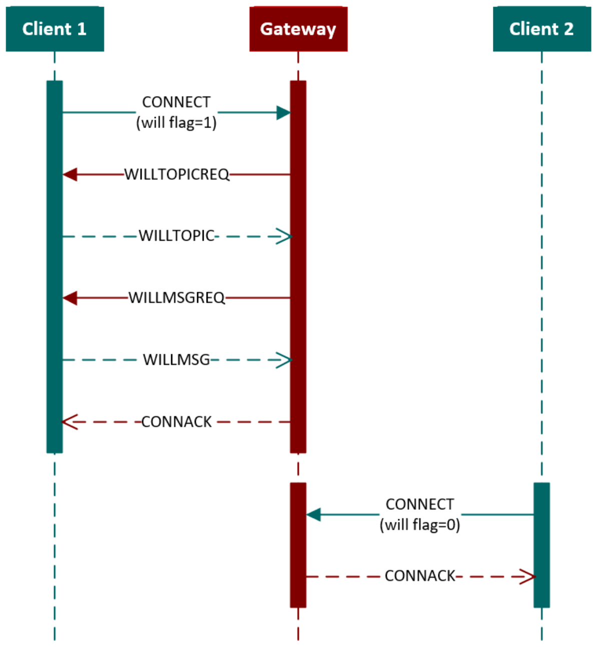


FIGURE 17. Connection between client and gateway, with the will flag set to 1 and 0.

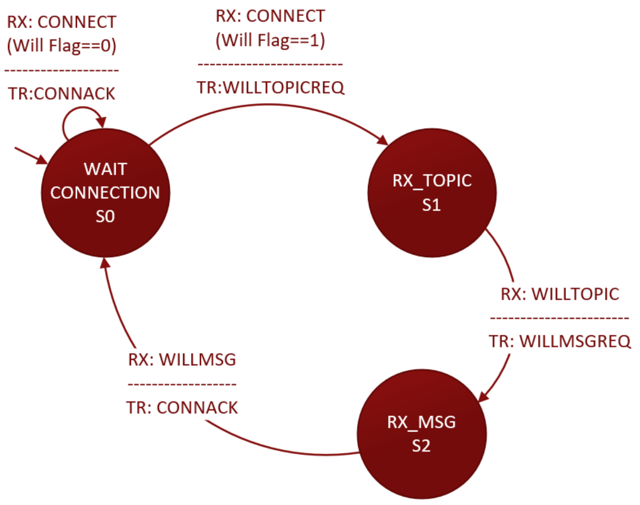


FIGURE 18. State diagram for the gateway, representing the configuration for the client connection.

flag equal to one is received. In state RX_TOPIC, the node waits for a *WILLTOPIC* message to send a *WILLMSGREQ* and move to state RX_MSG. Finally, the moment the gateway receives a *WILLMSG* message, a *CONNACK* message is transmitted, and the node returns to the initial state of the procedure to wait for another connection.

Figure 19 shows the SDL process for the gateway in which the state change of the gateway is observed in response to the reception of a *CONNECT* message, along with the different settings of the Will flag. Additionally, the variable `proc2Count` indicates when the correct sequence of messages has been exchanged. Once a value has been assigned to this variable, the procedure can be considered to have fulfilled its objective.

2) DESIGN OF THE MEALY MACHINE FOR THE CLIENT'S OPERATION

When a client node has found a gateway to connect, it will begin establishing a connection. For this reason, an initial state called ESTABLISH CONNECTION is used, in which the value of the *will* flag is verified before being sent within a *CONNECT* message. According to the state diagram in Fig. 20, if the flag *will* is equal to one, the node advances

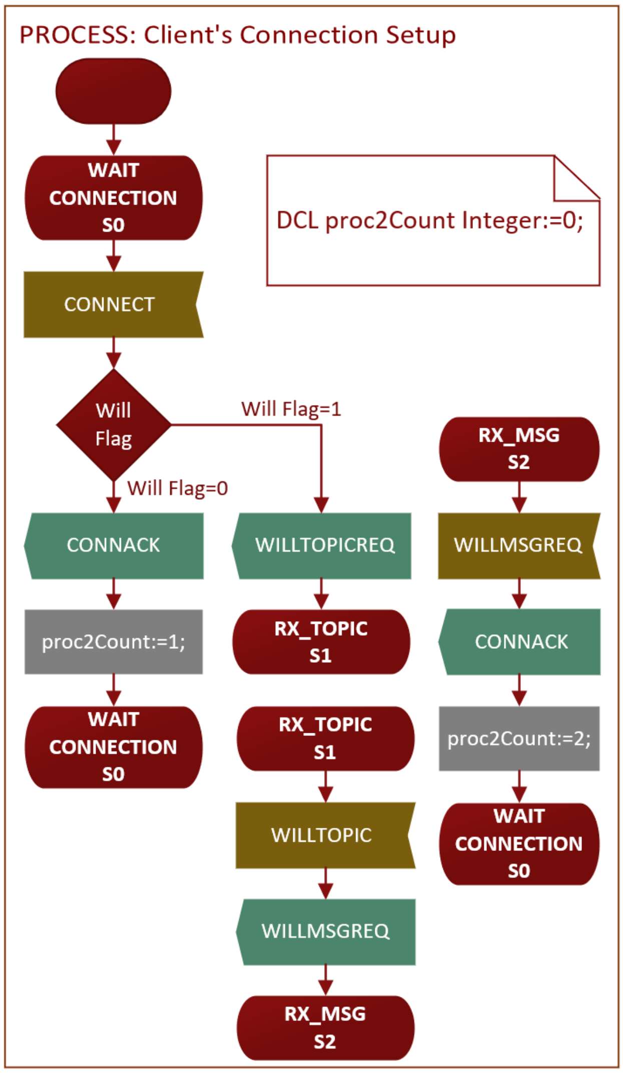


FIGURE 19. SDL process for the gateway, representing the configuration for the client connection.

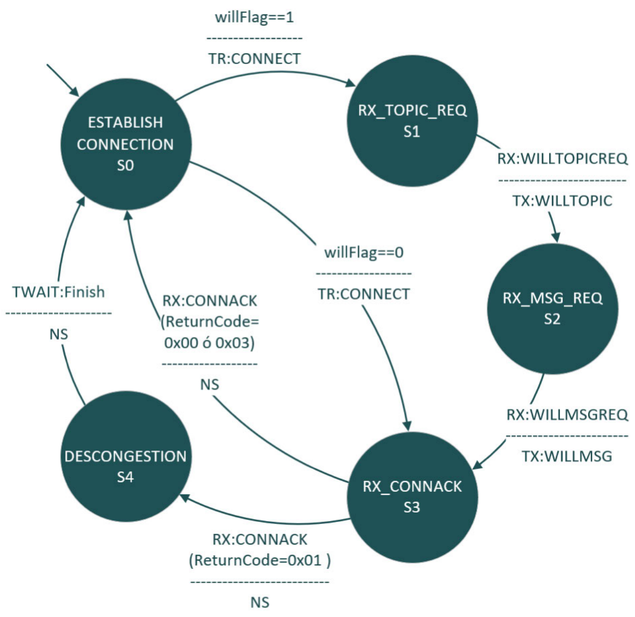


FIGURE 20. State diagram for the client, representing the configuration for the client connection.

to state RX_TOPIC_REQ, where it awaits receipt of the message *WILLTOPICREQ*. Subsequently, it goes to the state RX_TOPIC_REQ, in which it waits to receive the message *WILLMSGREQ* and then to the state RX_CONNACK. After receiving the respective request in its corresponding state, the node transmits the messages *WILLTOPIC* and *WILLMSG*. If the *will* flag equals zero, the node goes directly to the RX_CONNACK state after sending its *CONNECT* message. In the state RX_CONNACK, the node waits for the arrival of a *CONNACK* message, which contains the ReturnCode field that indicates whether the connection is accepted or rejected. Once the connection is accepted or rejected (ReturnCode: not supported), the node returns to the initial state to execute another procedure or restart the connection. If the node is rejected due to congestion, it enters the *DESCONGESTION* state and remains until the *TWAIT* timer expires. Afterward, it returns to the initial state to reconnect to the gateway.

Through the SDL process in Fig. 21, the change of client states is evident, attributed to the previous configuration of the *Will* flag of the *CONNECT* message that will be sent. Regardless of the flag forwarded, reaching the RX_CONNACK state is crucial, where the client's behavior due to the receipt of the ReturnCode field can be observed. When this field has the value of 0×01, it is necessary to initialize a timer before entering the DECONGESTION state. The variable `proc2Count` is also introduced, indicating whether the correct message sequence has been exchanged. After assigning a value to this variable, it can be considered that the procedure has fulfilled its function.

C. PROCEDURE FOR UPDATING WILL DATA

After establishing an MQTT-SN connection, the client will update the last messages stored at any time. The will topic can be updated by sending a WILTOPICUPD message, while the will message can be updated by a WILLMSGUPD message. It is important to note that neither of these two messages depends on the other. In response to update messages, the gateway will send WILLTOPICRESP or WILLMSGRESP, as detailed in the diagram in Fig. 22. Additionally, if it is necessary to remove the topic and the message from a client's will, the latter can send an empty WILLTOPICUPD message.

1) DESIGN OF THE MEALY MACHINE FOR GATEWAY OPERATION

As mentioned above, if a gateway has received a will message or topic, it will be subject to receiving requests to update the will data at any time. The procedure to update the last will data will have a single state (WAIT WILLTOPIC OR WILLMSG), which can be activated by

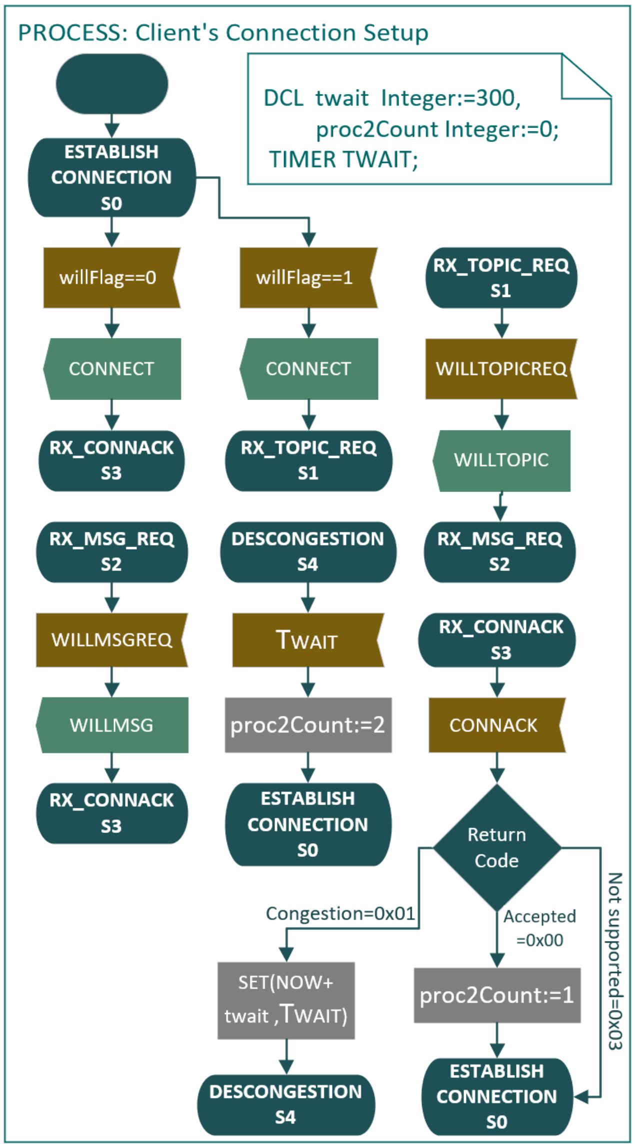


FIGURE 21. SDL process for the client, representing the configuration for the client connection.

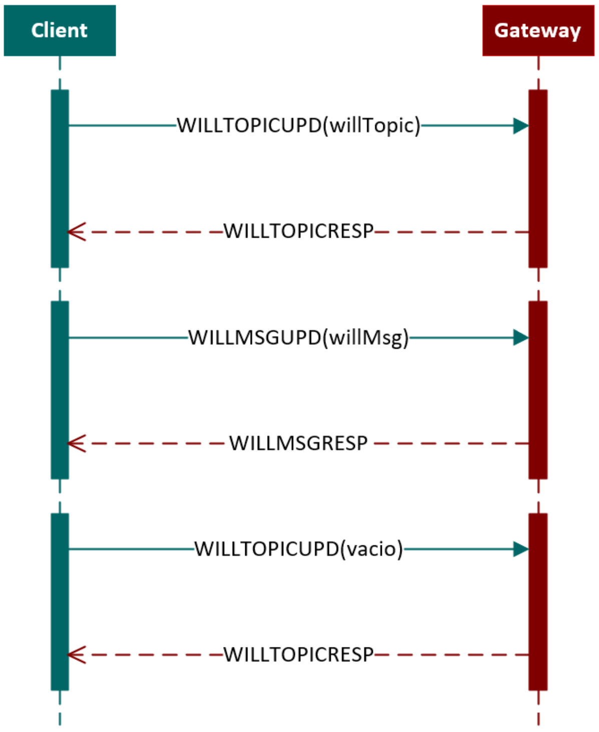


FIGURE 22. Update of last will data.

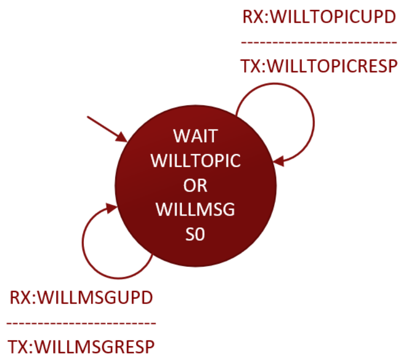


FIGURE 23. State diagram for the gateway, corresponding to the procedure for updating will data.

receiving a message *WILLTOPICUPD* or a *WILLMSGUPD*, and then responding with a message *WILLTOPICRESP* or a *WILLMSGRESP* respectively, and finally return to the original state, as illustrated in Fig. 23.

This SDL process, as shown in Fig. 24, is similar to the one indicated in the Mealy machine, with the difference that the variable `proc3Count` is added. This variable signals that the successful exchange of messages between the gateway and the client has been completed. Once a value is assigned to this variable, the procedure can be considered to have fulfilled its function.

2) DESIGN OF THE MEALY MACHINE FOR THE CLIENT'S OPERATION

Once a client has connected to a gateway, it is considered an active client and can update its will data at any time by running the will data update procedure. If the client node wants to update its will topic, it will need to receive a signal $updateWILLTOPIC = 1$, when it is in state ACTIVE UPDATE WILL DATA, to transmit a message *WILLTOPICUPD* and go to state RX TOPIC RESP. Similarly, if the client node wants to update its will message, it will need to receive a signal $updateWILLMSG = 1$ to transmit a message *WILLMSGUPD* and move to state RX_MSG_RESP. The client will return to the initial state once it has received its respective response, *WILLTOPICRESP* or *WILLMSGRESP*. Fig. 25 shows the state diagram of this procedure.

The SDL process in Fig. 26 is similar to that indicated on the Mealy machine, with the difference that the variable `proc3Count` is added. This variable indicates that the successful exchange of message sequences between the client

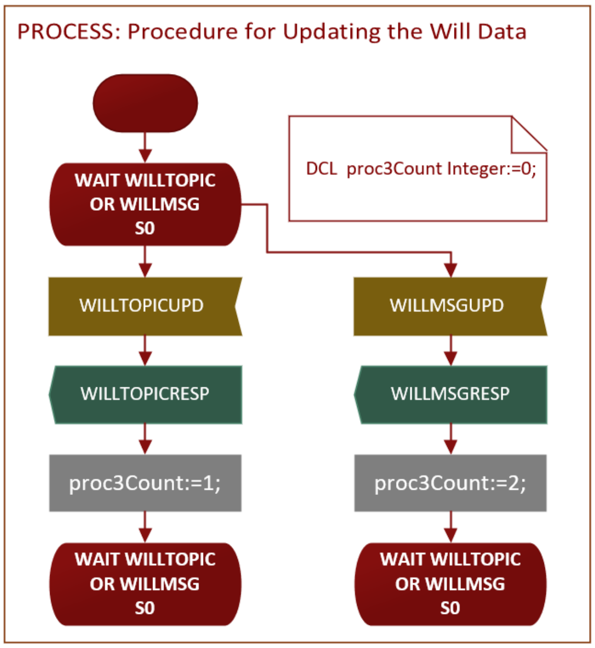

PROCESS: Procedure for Updating the Will Data
DCL proc3Count Integer:=0;
WAIT WILLTOPIC OR WILLMSG S0
WILLTOPICUPD
WILLMSGUPD
WILLTOPICRESP
WILLMSGRESP
proc3Count:=1;
proc3Count:=2;
WAIT WILLTOPIC OR WILLMSG S0
WAIT WILLTOPIC OR WILLMSG S0

FIGURE 24. SDL process for the gateway, corresponding to the procedure for updating will data.

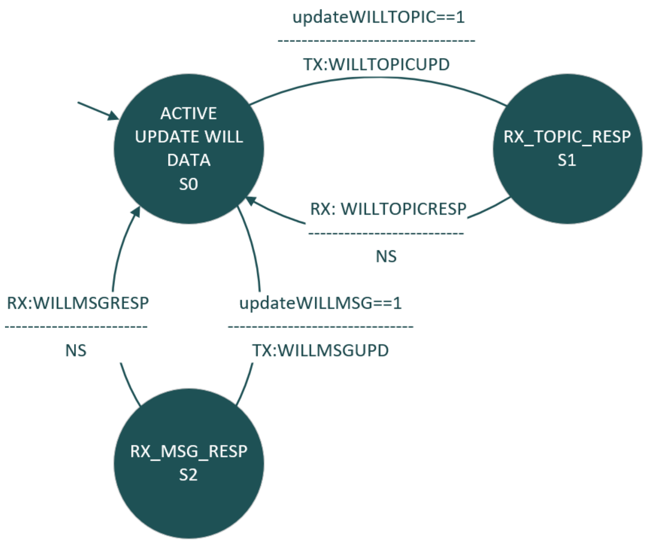


FIGURE 25. State diagram for the client, corresponding to the procedure for updating data will.

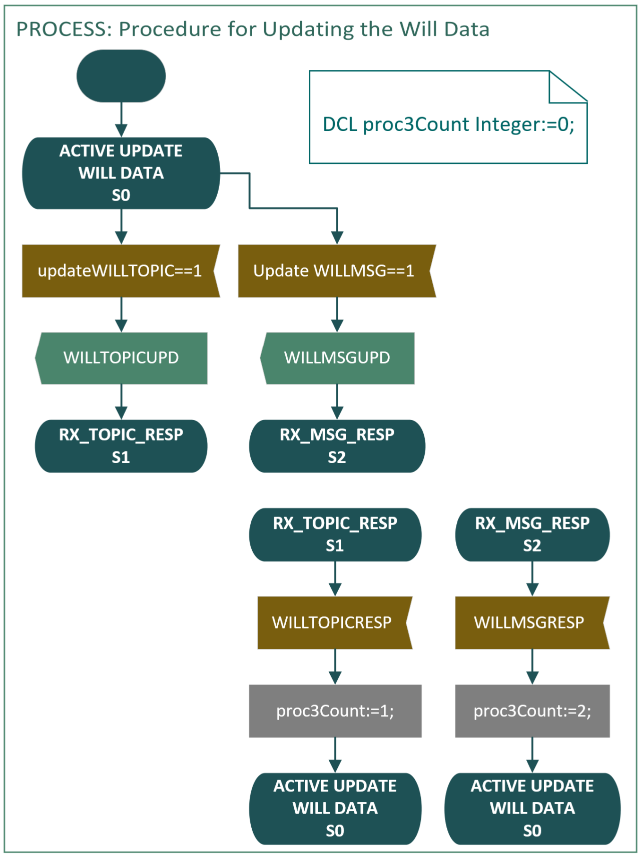


FIGURE 26. SDL process for the client, corresponding to the procedure for updating will data.

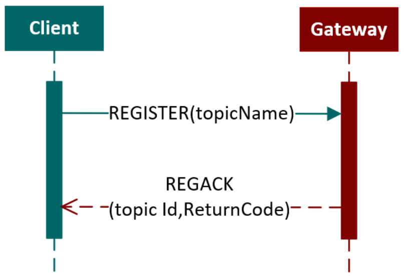


FIGURE 27. Procedure to register a topic name, executed by a client.

and the gateway has been completed. Once a value is assigned to this variable, it can be considered that the procedure has fulfilled its function.

D. PROCEDURE FOR REGISTERING TOPIC NAMES

MQTT-SN introduces a procedure for registering topic names to reduce the size of publishing messages. This procedure allows the client and the gateway to inform their counterpart about a topic identifier in the topic field. This topic identifier should be shorter than a conventional topic name. When the client wants to register a topic name, it starts by sending a REGISTER message. Upon receiving this message, the gateway parses it, assigns an identifier to the corresponding topic name, and transmits it to the client via a REGACK message. If the registration is rejected, the gateway also sends a REGACK message, including in the returnCode field an indication of the reason for the rejection, as detailed in Fig. 27.

If the client loses connection and reconnects without setting the CleanSession flag, it may need information about the topic names and IDs it previously registered or subscribed to. The gateway notifies the client of the topic names and their respective identifiers using a REGISTER message. The client uses the received topic identifier to publish its

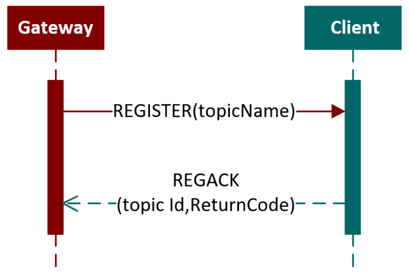


FIGURE 28. Procedure to register a topic name, executed by a client and a gateway.

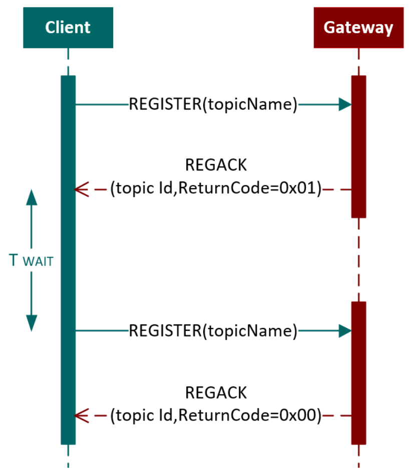


FIGURE 29. Rejection due to congestion during a registration procedure.

messages, as shown in Fig. 28. The gateway also uses this registration procedure when the client has subscribed to topic names using wildcard characters. This is because the message that accepts a subscription cannot contain more than one topic identifier. In case of rejection due to congestion, the procedure will be restarted after waiting a TWAIT time, either by the client or the gateway, as shown in Fig. 29.

1) DESIGN OF THE MEALY MACHINE FOR GATEWAY OPERATION

After receiving a connection, the gateway waits for a registration message sent by the newly connected client. Therefore, the initial state called WAIT REGISTRATION is established for this procedure. During this state, the node responds with a *REACK* message upon receipt of a *REGISTER* message and then returns to the same state. Since the gateway can also initiate a registration procedure, it can send a *REGISTER* message once it receives the *readyRegister* = 1 signal from the initial state. After sending the mentioned message, the node goes to the RX_REGACK state, where it waits for a *REACK* message sent by the client. Once the *REACK* message is received, its ReturnCode field is checked. If the registration is accepted or rejected (ReturnCode=invalid or unsupported topic identifier), the node returns to the initial state to execute another procedure or restart the registration procedure. Fig. 30 illustrates the state diagram

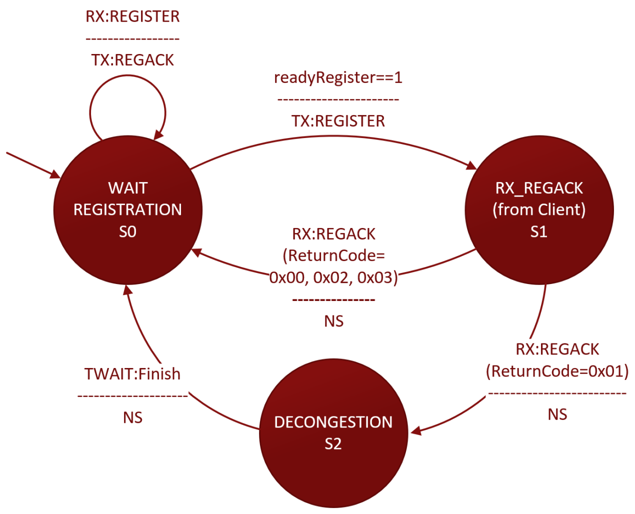


FIGURE 30. Diagrama de estados para el gateway, correspondiente al procedimiento para registrar nombres de temas.

of this procedure. If there is a rejection due to congestion, the gateway node must go to the DECONGESTION state, remaining until the *TWAIT* timer expires. Afterward, the node will return to the initial state to register the topic name again.

This SDL process is similar to that indicated on the Mealy machine, as shown in Fig. 31, with the addition of the variable `proc4Count`. In addition, the timer *TWAIT* and the variable `twait` are incorporated to control its duration. The timer is activated when receiving a rejection due to congestion. Once a value is assigned to the variable `proc4Count`, the procedure can be considered to have fulfilled its function. If necessary, it can also be restarted. The variable does not take on a value when rejections are received for invalid topic identifiers or unsupported messages.

2) DESIGN OF THE MEALY MACHINE FOR THE CLIENT'S OPERATION

After the client has established a successful connection, it is considered an active client. However, before publication, the client must initiate a registration procedure. For this reason, although the state diagram for the client is similar to that of the gateway, the initial state is called ACTIVE REGISTRATION. During this initial state, the client sends a *REGISTER* message when it receives the *readyRegister* == 1 signal. After sending this message, the node goes to the RX_REGACK state, where it waits for a *REACK* message sent by the gateway. Upon receipt of the *REACK* message, its ReturnCode field is examined. If the registration is accepted or rejected, the node returns to the initial state to execute another procedure or restart the registration procedure. If there is a rejection due to congestion, the client node goes to the DECONGESTION state, which will remain until the *TWAIT* timer expires. Afterward, the node must return to the initial state to register the topic name again. If the client node is in the initial state, it can respond with a *REACK* message

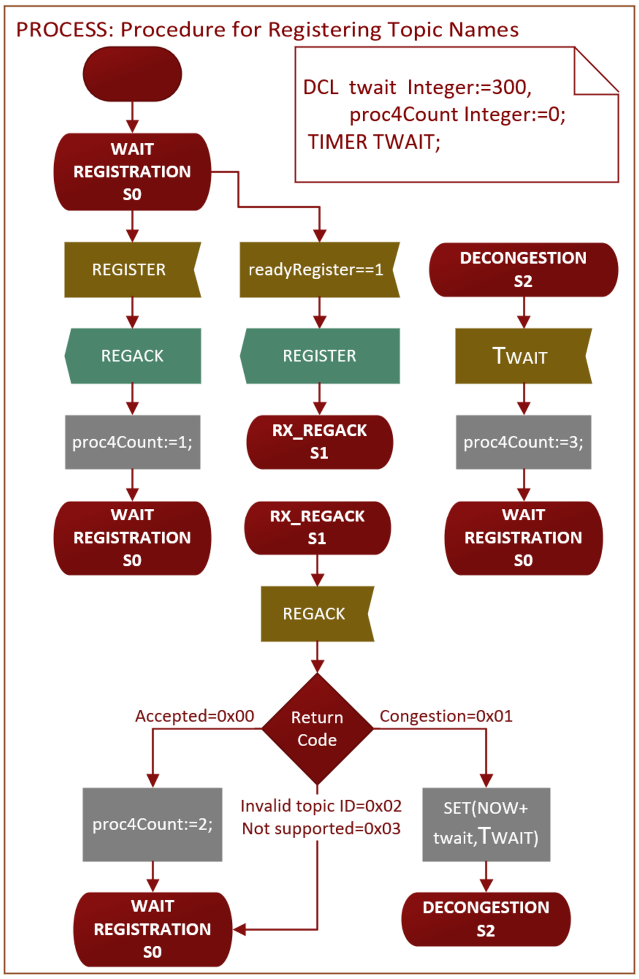


FIGURE 31. SDL process for the gateway, corresponding to registering topic names.

upon receipt of a *REGISTER* message sent by the gateway node. Fig. 32 shows the state diagram of this procedure.

This SDL diagram for the client, as seen in Fig. 33, is practically identical to the one used by the gateway, with the only difference being certain state names. The variable `proc4Count` is also included as in the previous diagram. This variable indicates that the successful exchange of message sequences between the client and the gateway has been completed. Once a value has been assigned to this variable, the procedure can be considered to have fulfilled its function.

E. CLIENT'S PUBLICATION PROCEDURE

Clients can publish information associated with a topic if they have previously completed the registration procedure. The payload and its topic identifier are sent within a PUBLISH message. This procedure covers three of the four quality of service levels (QoS 0, QoS 1, and QoS 2) supported by MQTT-SN. Therefore, a gateway will respond differently upon receiving a publication. The behavior of the gateway at the three QoS levels mentioned is described below:

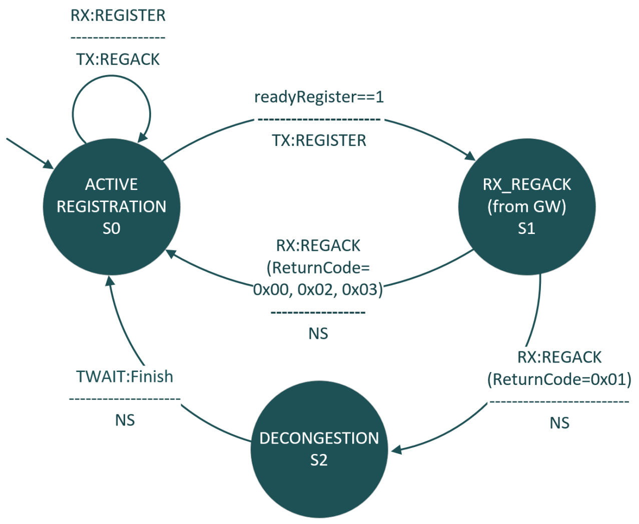


FIGURE 32. State diagram for the client, corresponding to registering topic names.

- If the gateway receives a publication with QoS 0, it does not send any message to confirm receipt.
- If the gateway receives a publication with QoS 1, it responds with a PUBAC message.
- If the gateway receives a publication with QoS 2, it responds with a PUBREC message. The client receives this message and responds with a PUBREL message. Finally, the gateway sends the PUBCOMP message in response to the client's last message.

If a client makes a publication with quality of service 1 or 2, it must wait for the message exchange to complete before starting a new publication. The sequence diagram in Fig. 34 shows a client sending publication messages with the different quality of service levels mentioned in this procedure. When a post is rejected, the returnCode field indicates the reason for the rejection. In case of rejection due to an invalid topic identifier (returnCode=0×02), the client must register the topic name again, as shown in Fig. 35. In publishes with QoS 1 and QoS 2, if there is a rejection due to congestion (returnCode=0×01), the client will try to publish after waiting for a TWAIT time, as shown in the diagrams in Fig. 36.

1) DESIGN OF THE MEALY MACHINE FOR GATEWAY OPERATION

If the gateway has completed a registration procedure initiated by the client, it can initiate a publication procedure. Therefore, an initial state called WAIT PUBLICATION is established for the gateway node. During this state, the node can receive *PUBLISH* messages with various quality of service variants. The following describes how the node should respond to each QoS level:

- **PUBLISH (QoS 0)**: When a publication with this quality of service is received, the node does not issue any response and returns to the initial state.

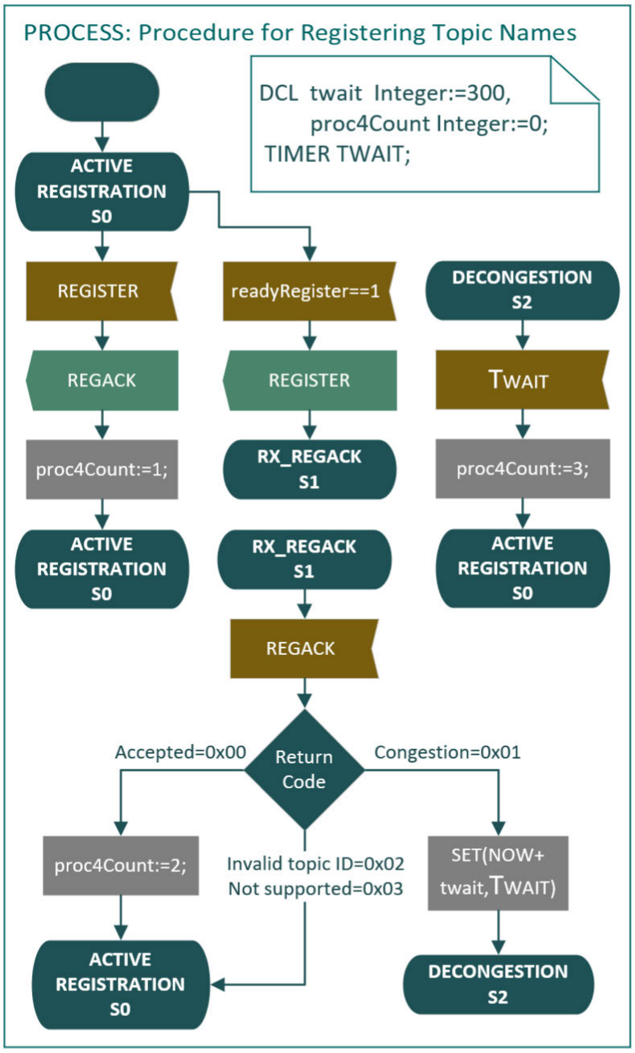


FIGURE 33. SDL process for the client, corresponding to registering topic names.

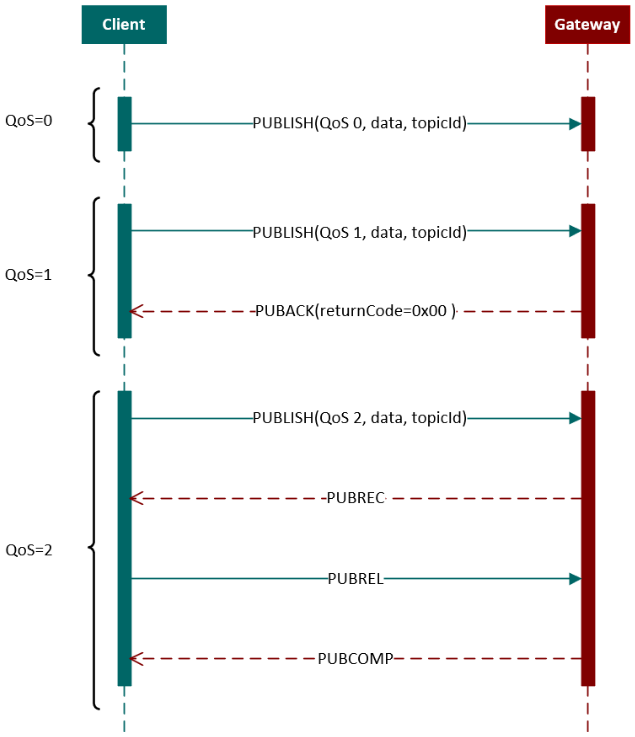

Client
Gateway
QoS=0
PUBLISH(QoS 0, data, topicId)
QoS=1
PUBLISH(QoS 1, data, topicId)
PUBACK(returnCode=0x00)
QoS=2
PUBLISH(QoS 2, data, topicId)
PUBREC
PUBREL
PUBCOMP

FIGURE 34. Client sending PUBLISH messages, with QoS 0, QoS 1, and QoS 2, to a gateway.

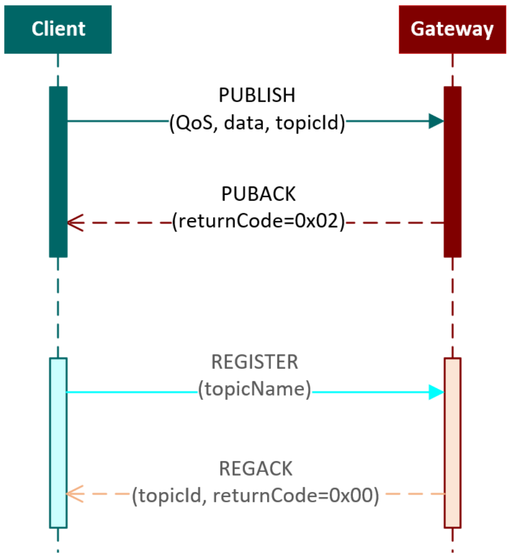


FIGURE 35. Exchange of messages between client and gateway when errors occur due to invalid identifiers.

- **PUBLISH (QoS 1):** Upon receipt of a publication with this quality of service, the node responds with a *PUBACK* message and returns to the initial state.
- **PUBLISH (QoS 2):** Upon receiving a publication with this quality of service, the node responds with a *PUBREC* message and enters the RX_PUBREL state. During this state, the node waits for the arrival of the *PUBREL* message to respond with a *PUBCOMP* and return to the initial state.

If the node is in the initial state and receives a *PUBLISH* message with an invalid topic identifier, it will respond with a *PUBACK* message indicating the reason for the rejection (ReturnCode=0 × 02). Finally, the node must return to the initial state to handle new publications or initiate new procedures. Fig. 37 shows the state diagram of this procedure.

The SDL process in Fig. 38 shows the state the node should remain in when receiving a *PUBLISH* message with a valid or invalid topic identifier. Fig. 38 also shows the verification of the QoS level of the received message after confirming the validity of the topic identifier. The change of state of the node and the assignment of a value to the variable `proc5Count` will depend on the QoS level of the *PUBLISH* message. This mentioned variable indicates that the correct exchange of messages between the gateway and the client has occurred. Additionally, the *iniRegister* variable is available to indicate that the procedure has failed.

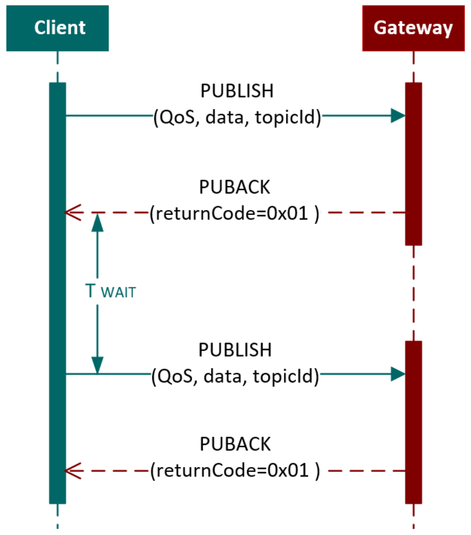


FIGURE 36. Exchange of messages between client and gateway when errors occur due to congestion problems.

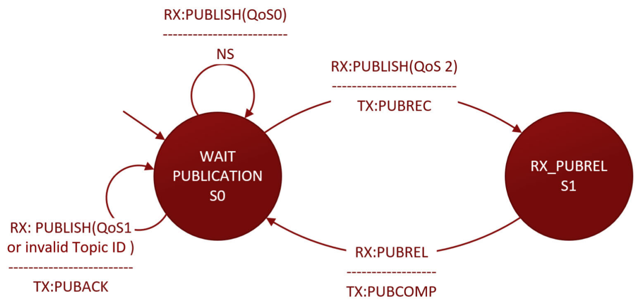


FIGURE 37. State diagram for the gateway, corresponding to the client publication procedure.

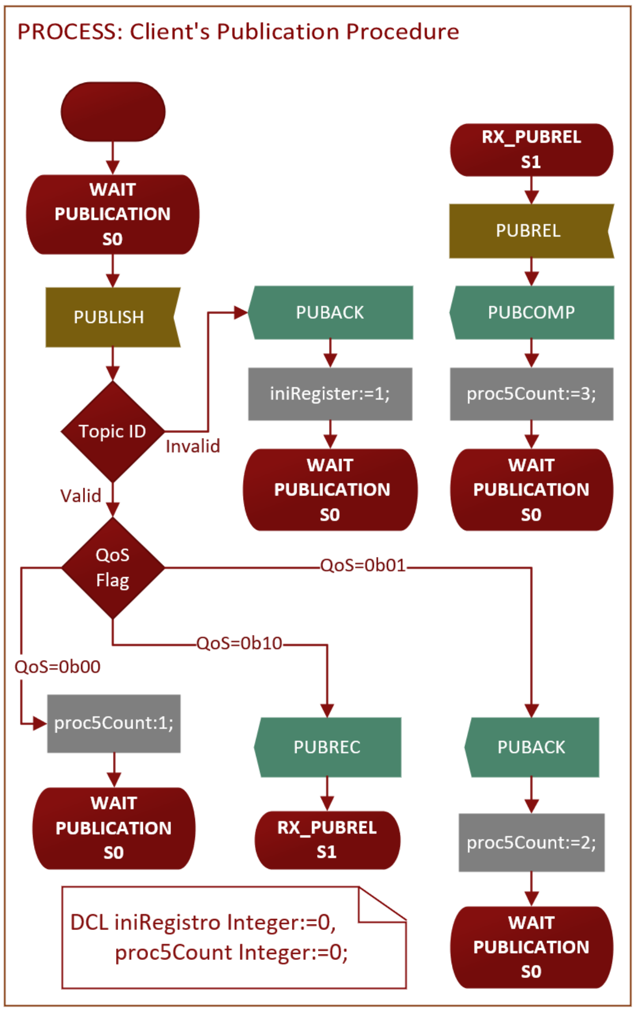


FIGURE 38. Proceso SDL para el gateway, correspondiente al procedimiento de publicacion del cliente.

2) DESIGN OF THE MEALY MACHINE FOR THE CLIENT'S OPERATION

If the client has completed a registration procedure, it is enabled to initiate a publication procedure. Therefore, an initial state called ACTIVE PUBLICATION is established. During this initial state, the node can send *PUBLISH* messages. The behavior of the client node when sending publications with different levels of QoS is described below:

- **PUBLISH (QoS 0):** When the node receives the signal $PublicationReady(QoS0) == 1$, it transmits a message *PUBLISH* and remains in the initial state.
- **PUBLISH (QoS 1):** When the node receives the signal $PublicationReady(QoS1) == 1$, it transmits a message *PUBLISH* and enters the RX_ACK state. During this state, the node waits for a *PUBACK* message to check the ReturnCode field. If the publication is accepted or rejected for a reason other than congestion, the node returns to the initial state to execute another procedure or restart the publication procedure. In case of rejection due to congestion, the node enters the DECONGESTION state, where it remains until the *TWAIT* timer expires. The node then returns to the initial state to try to send the publish message again.
- **PUBLISH (QoS 2):** When the node receives the signal $PublicationReady(QoS2) == 1$, it transmits a message *PUBLISH* and enters the RX_ACK state. During this state, the node waits to receive a *PUBREC* message to transmit a *PUBREL* message and enter the RX_PUBCOMP state. The publication is considered accepted if a *PUBCOMP* message is received during this last state.

A *PUBACK* message can also be received during the initial state if it contains the field ReturnCode$=0\times02$. Fig. 39 shows the state diagram of this procedure.

The SDL process in Fig. 40 shows the state changes the node must make after sending a publish message. The state change depends on the QoS level, which is analyzed before sending the *PUBLISH* message. Figure shows the analysis of the returnCode field and its corresponding change of states due to the different values contained in this field. Additionally, the variable `proc5Count` is

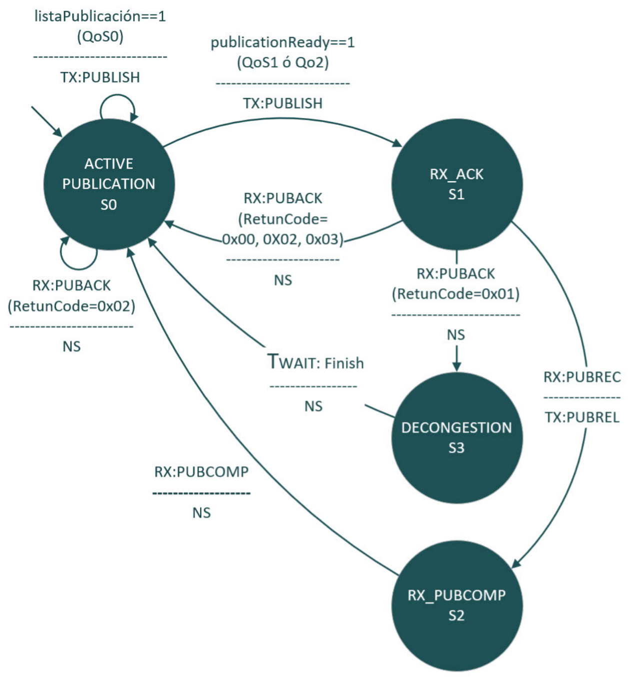

listaPublicación==1 (QoS0)
TX:PUBLISH
publicationReady==1 (QoS1 ó Qo2)
TX:PUBLISH
ACTIVE PUBLICATION S0
RX_ACK S1
RX:PUBACK (RetunCode= 0x00, 0X02, 0x03)
NS
RX:PUBACK (RetunCode=0x02)
NS
RX:PUBACK (RetunCode=0x01)
NS
TWAIT: Finish
NS
DECONGESTION S3
RX:PUBREC
TX:PUBREL
RX:PUBCOMP
NS
RX_PUBCOMP S2

FIGURE 39. State diagram for the client, corresponding to the client publication procedure.

introduced, which indicates that the correct exchange of messages between the client and the gateway has been carried out. Likewise, the variable `iniRegister` is available to indicate that the procedure has failed.

F. PUBLISHING WITH QOS LEVEL -1

The QoS -1 publishing procedure is helpful for simple MQTT-SN clients requiring this procedure. No connection, registration, subscription, or discovery procedures are necessary; the client only needs prior knowledge of the gateway address. The client sends PUBLISH messages to the gateway regardless of the state in which the latter is, nor does it verify whether the message arrives, as shown in Fig. 41. PUBLISH messages with QoS -1 can only have the following values:

- **QoS flag:** “0b11” for service quality level -1.
- **TopicIdType flag:** “0b01” for a predefined topic identifier or “0b10” for a short topic name.
- **TopicId field:** A predefined topic identifier or short topic name.
- **Data field:** The payload of the message.

1) DESIGN OF THE MEALY MACHINE FOR GATEWAY OPERATION

In this procedure, as shown in Fig. 42, only two states are defined: INACTIVE and WAIT PUBLICATION, because no other procedures are required to receive this type of publication, and it must also be considered that the gateway must be connected to a broker to start its operation. During the INACTIVE state, the node waits for the signal (*brokerReady* $=$ 1) that indicates that the broker is ready to operate and

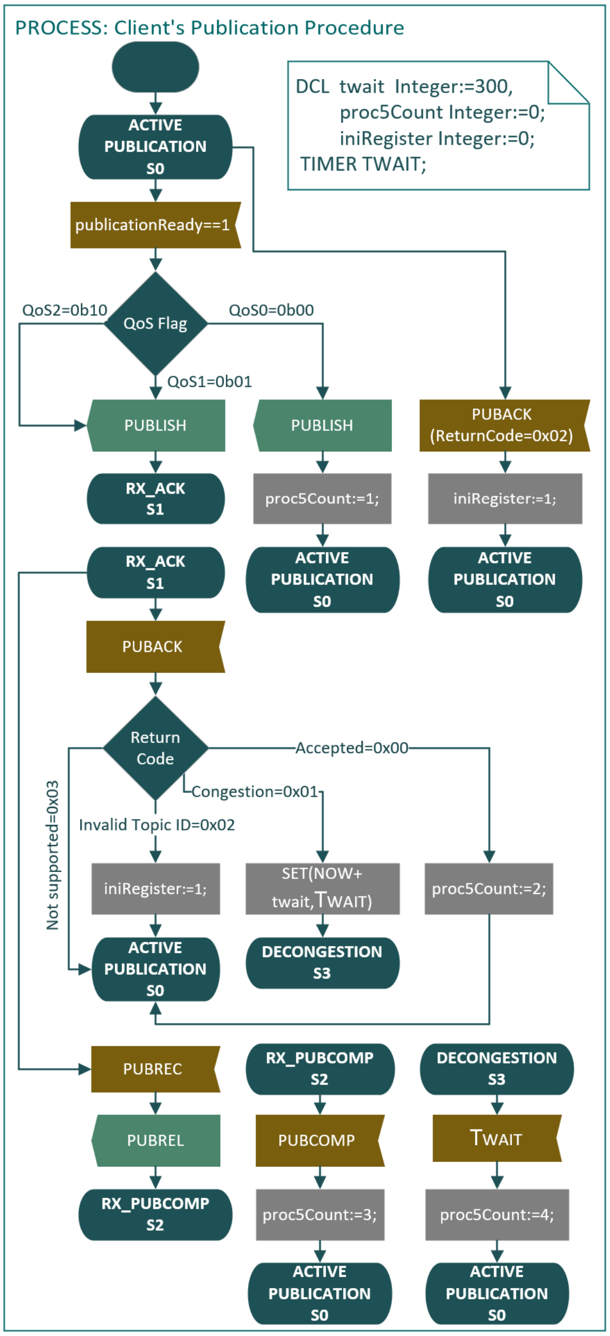


FIGURE 40. SDL process for the client, corresponding to the client's publication procedure.

then goes to the WAIT PUBLICATION state. In the WAIT PUBLICATION state, the node waits to receive *PUBLISH* messages with quality of service QoS-1, without issuing any confirmation. If the signal *brokerReady* $= 0$ is received due to any failure in the broker, the node will return to the INACTIVE state.

The SDL process in Fig. 43 incorporates the variable `proc6Count` that indicates the exchange of the correct sequence of messages between the gateway and the client. After assigning a value to this variable, it can be considered

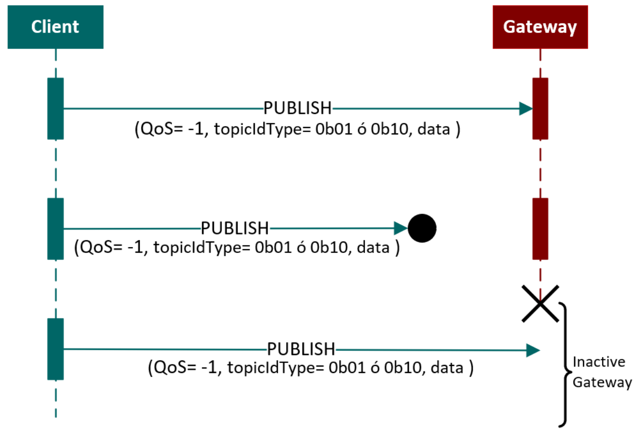

Client
Gateway
PUBLISH
(QoS= -1, topicIdType= 0b01 ó 0b10, data)
PUBLISH
(QoS= -1, topicIdType= 0b01 ó 0b10, data)
PUBLISH
(QoS= -1, topicIdType= 0b01 ó 0b10, data)
Inactive Gateway

FIGURE 41. Client sending PUBLISH messages with QoS −1 to the gateway.

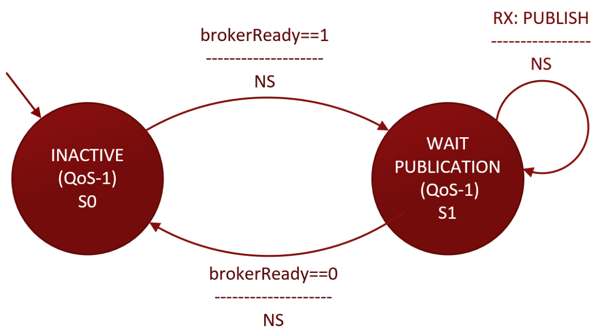


FIGURE 42. State diagram for the gateway, corresponding to the publication procedure with QoS -1.

that the procedure has fulfilled its function and that no timer or additional variable is added to the one indicated.

2) DESIGN OF THE MEALY MACHINE FOR THE CLIENT'S OPERATION

As shown in Fig. 44, a client will remain exclusively in the ACTIVE PUBLICATION state since the client can only send publication messages. In this state, it waits for the $publicationReady(QoS - 1) = 1$ signal to transmit the corresponding *PUBLISH* message and then returns to the same state.

In the SDL process, as shown in Fig. 45, the variable `proc6Count` is added. This variable indicates that the correct exchange of messages between the client and the gateway has been carried out.

G. CLIENT'S TOPIC SUBSCRIBE OR UNSUBSCRIBE PROCEDURE

To subscribe to a topic name, a client sends a SUBSCRIBE message to the gateway, indicating the topic of interest. If the gateway can accept the subscription, it assigns a topic identifier to the received topic name. It returns it within a SUBACK message to the client, as illustrated in the diagram in Fig. 46. If the client subscribes to a topic name containing

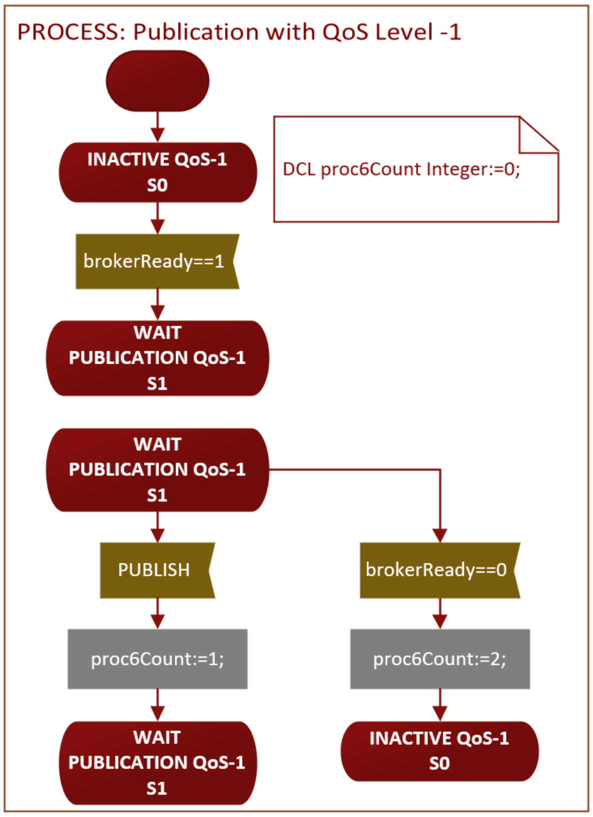

PROCESS: Publication with QoS Level -1
INACTIVE QoS-1 S0
DCL proc6Count Integer:=0;
brokerReady==1
WAIT PUBLICATION QoS-1 S1
WAIT PUBLICATION QoS-1 S1
PUBLISH
brokerReady==0
proc6Count:=1;
proc6Count:=2;
WAIT PUBLICATION QoS-1 S1
INACTIVE QoS-1 S0

FIGURE 43. SDL process for the gateway, corresponding to the publication procedure with QoS -1.

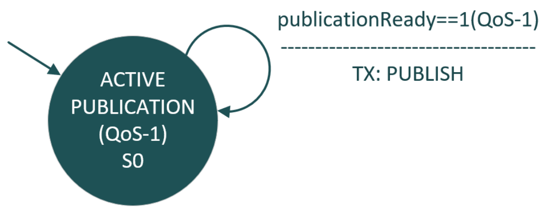


FIGURE 44. State diagram for the client, corresponding to the publication procedure with QoS −1.

a wildcard character (+ or #), the SUBACK message will include the topic ID value 0×0000. When a client subscribes to multiple topics using wildcards, it will be informed of the ID of all topics involved before sending the first publish message for a specific topic. To communicate the value of the topic identifier to the client, the gateway must use the registration procedure. Similar to the publishing procedure, topic identifiers can be predefined for specific topic names; You can also have short theme names. In both cases, the client must subscribe to these identifiers or topic names to receive posts related to the latter. If the subscription cannot be accepted, a SUBACK message will be sent back to the client with the reason for the rejection encoded in the returnCode field. If the cause of the rejection is "rejected: congestion", the client must wait the TWAIT time before forwarding the SUBSCRIBE message to the gateway, as shown in Fig. 47.

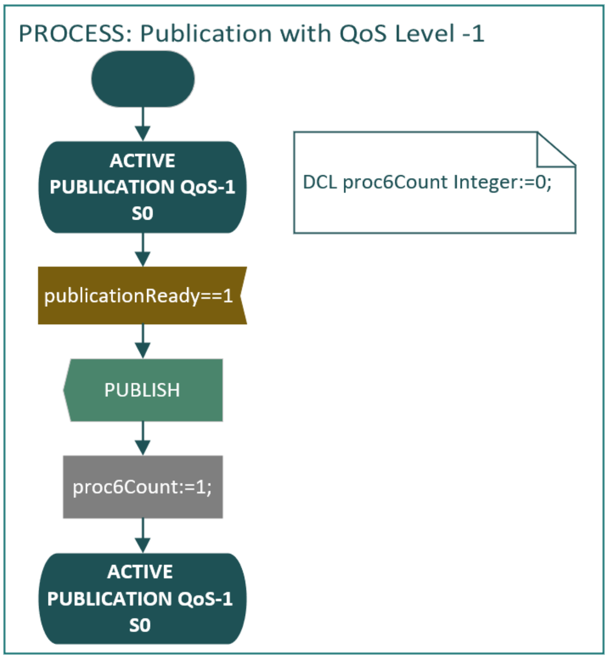

PROCESS: Publication with QoS Level -1
ACTIVE PUBLICATION QoS-1 S0
DCL proc6Count Integer:=0;
publicationReady==1
PUBLISH
proc6Count:=1;
ACTIVE PUBLICATION QoS-1 S0

FIGURE 45. SDL process for the client, corresponding to the publication procedure with QoS −1.

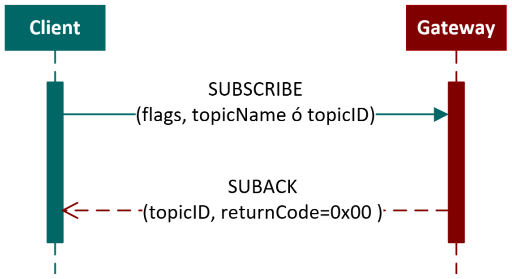


FIGURE 46. Exchange of subscription messages between client and gateway.

To unsubscribe, a client sends a UNSUBSCRIBE message to the gateway, which must respond with a UNSUBACK message. The client indicates the name or topic identifier from which they wish to unsubscribe, as shown in the diagram in Fig. 48.

1) DESIGN OF THE MEALY MACHINE FOR GATEWAY OPERATION

In this procedure, as shown in Fig. 49, the gateway node needs to remain in a single state called WAIT SUBSCRIPTION OR UNSUBSCRIPTION, where it constantly waits for *SUBSCRIBE* messages and then sends a *SUBACK* message indicating whether the subscription is accepted or rejected. The node can also receive *UNSUBSCRIBE* messages from clients that wish to unsubscribe, which are responded to with a *UNSUBACK*. After responding to received messages, the node returns to the same state.

The diagram in Fig. 50 includes the variable `proc7Count`, which indicates that the correct exchange of messages has been carried out between the gateway and the client when the latter has requested a connection or when the of a topic and will take the value of 1 or 2, respectively.

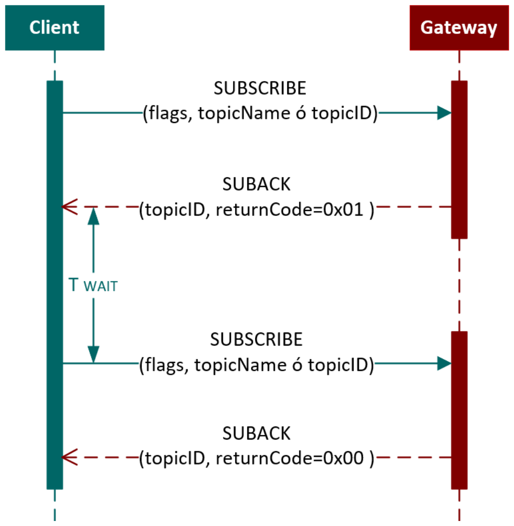

Client
Gateway
SUBSCRIBE
(flags, topicName ó topicID)
SUBACK
(topicID, returnCode=0x01)
T WAIT
SUBSCRIBE
(flags, topicName ó topicID)
SUBACK
(topicID, returnCode=0x00)

FIGURE 47. Interaction between client and gateway when the subscription is not accepted due to congestion.

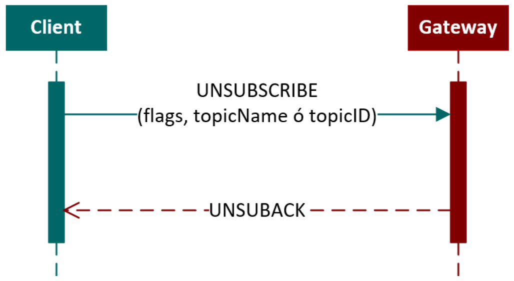


FIGURE 48. Exchange of messages, to unsubscribe from a topic, between client and gateway.

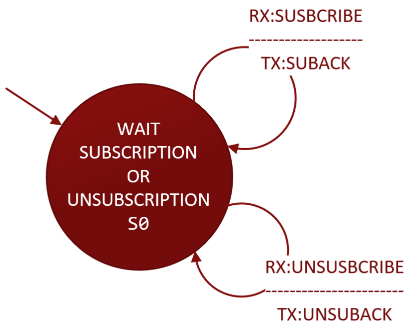


FIGURE 49. State diagram for the gateway, corresponding to the procedure to Subscribe or Unsubscribe from a topic.

2) DESIGN OF THE MEALY MACHINE FOR THE CLIENT’S OPERATION

When a client has established a connection to a gateway, it is considered in ACTIVE state, allowing it to subscribe to a topic of interest. This procedure starts with the node in the state

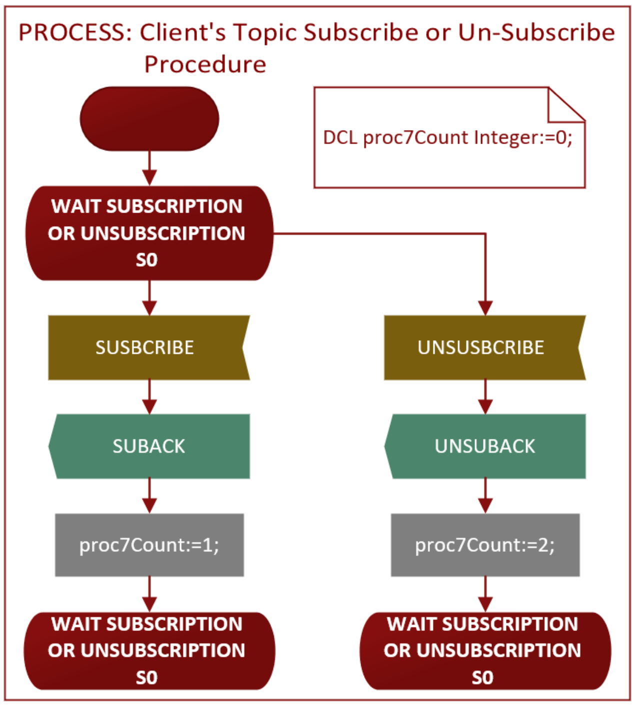

PROCESS: Client's Topic Subscribe or Un-Subscribe Procedure
DCL proc7Count Integer:=0;
WAIT SUBSCRIPTION OR UNSUBSCRIPTION S0
SUSBCRIBE
UNSUSBCRIBE
SUBACK
UNSUBACK
proc7Count:=1;
proc7Count:=2;
WAIT SUBSCRIPTION OR UNSUBSCRIPTION S0
WAIT SUBSCRIPTION OR UNSUBSCRIPTION S0

FIGURE 50. SDL process for the gateway, corresponding to the procedure to subscribe or unsubscribe from a topic.

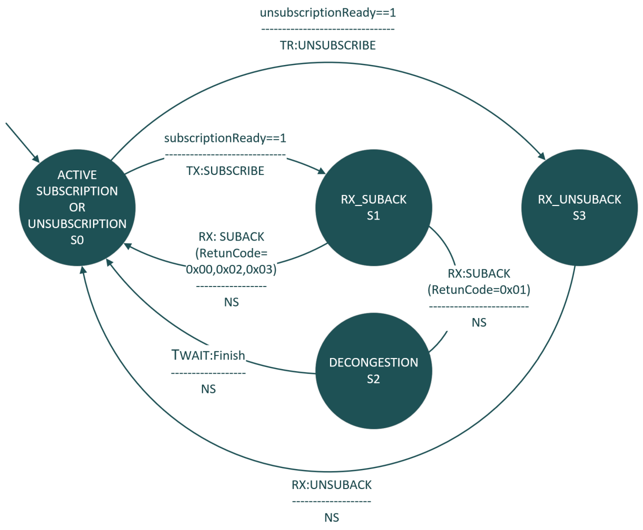


FIGURE 51. State diagram for the client, corresponding to the procedure to subscribe or unsubscribe from a topic.

named ACTIVE SUBSCRIPTION OR UNSUBSCRIPTION. During this state, the node waits for a signal to subscribe to or unsubscribe from a topic. If the signal *subscriptionReady* $==$ 1 is received in the initial state, a message *SUBSCRIBE* is sent, and then the state RX_SUBACK is passed, where the node waits for a message *SUBACK* sent by the gateway. Upon receiving the *SUBACK* message, its ReturnCode field is verified. If the subscription is accepted or rejected for a reason other than congestion, the node returns to the initial state to execute another procedure or restart the process. If the subscription is rejected due to congestion, the client node goes to the DECONGESTION state, remaining until the *TWAIT* timer expires. The node must then return to the initial state to try to subscribe again. If in the initial state, the signal *unsubscriptionReady* $==$ 1 is received, a message *UNSUBSCRIBE* is sent, and then the state is passed to RX_UNSUBACK, where the node waits for a UNSUBACK message sent by the gateway. In this way, a customer unsubscribes from a previously subscribed topic. Fig. 51 shows the state diagram of this procedure.

In the SDL process in Fig. 52, it is observed that the state changes caused by the confirmation messages sent by the gateway are similar to what is shown in the state diagrams of this procedure, with the particularity that the variables are incorporated. texttiniRegister and `proc7Count`. These variables indicate the proper message exchange between the client and the gateway has occurred. If the subscription is accepted, the topicID field must be verified; this way, the node will determine if it should initiate the registration procedure due to the use of wildcards within its subscription message. If yes, the node will assign a value to the variable `iniRegister`, in the same way it would if it receives a rejection for an invalid topic identifier in the returnCode field.

H. GATEWAY'S PUBLICATION PROCEDURE

For a gateway to publish on a specific topic, a client must first subscribe. Posts use the topic ID and QoS set during subscription. As in a publication by the client, the gateway sends a PUBLISH message, from which a response can be obtained from the client depending on the level of quality of service (QoS 1 and QoS 2) or not received in the case of QoS 0 The sequence diagrams in Fig. 53 illustrate various publications made by the gateway. Publications can be rejected using PUBACK messages, similar to the client's publishing procedure.

1) DESIGN OF THE MEALY MACHINE FOR GATEWAY OPERATION

Figure 54 presents the state diagram of this procedure. Before preparing posts for submission, the node must check to see if it has any pending posts. Therefore, the node starts in the WAIT state, where, as its name indicates, it waits for the signal *pendingPublications* $>=$ 1 to advance to the PUBLICATION state. From the second state, the gateway node follows the same behavior as that of a client when it requires sending *PUBLISH* messages, with the distinction that once all pending publications have been sent, the node returns to the initial state. If necessary, the *pendingPublications* $==$ 0 signal must return to the WAIT state and start another procedure.

In the SDL process in Fig. 55, the gateway initiates the first state change when the gateway node has at least one pending message to publish. Likewise, the variable `proc8Count` is used, indicating that the correct message exchange between

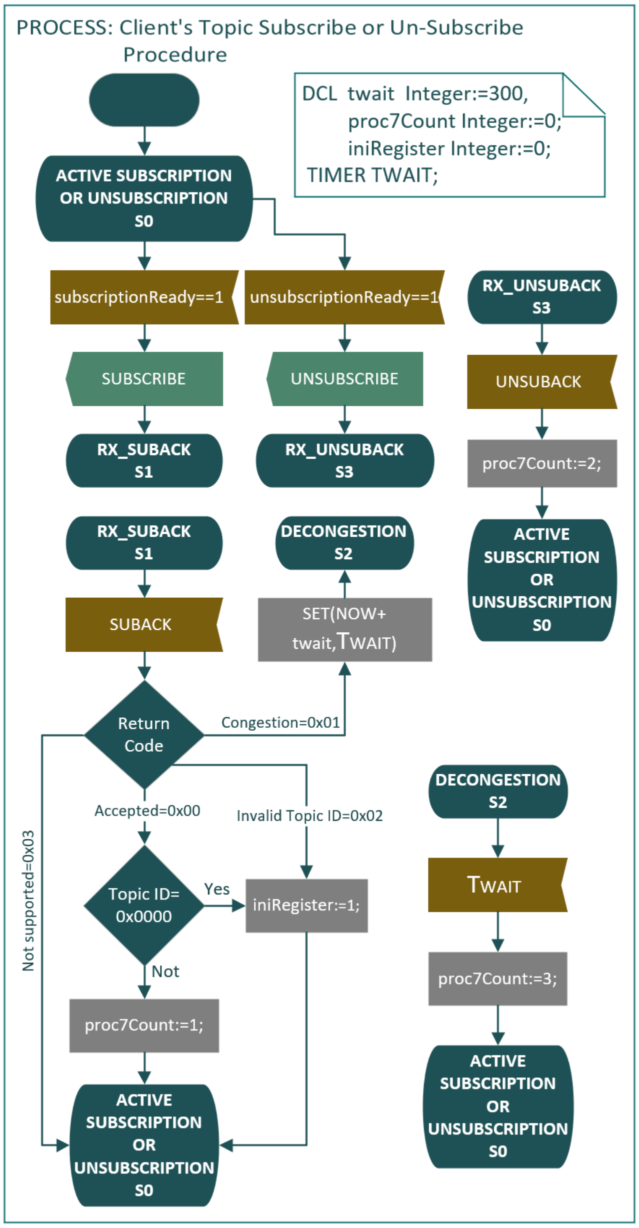


FIGURE 52. SDL process for the client, corresponding to the procedure to subscribe or unsubscribe from a topic.

the gateway and the client has been carried out. If the sending of pending messages goes smoothly, the number of pending posts decreases until it reaches zero, at which point it returns to the initial state. In addition, the variable `iniRegister` is used, which indicates that the procedure has failed.

2) DESIGN OF THE MEALY MACHINE FOR THE CLIENT'S OPERATION

During this procedure, the client node behaves similarly to the gateway node when executing the client's publishing procedure. The only difference lies in the name of the initial state, which will now be named ACTIVE WAIT PUBLICATION because the client must be active to start this procedure. From this initial state, the client will attend to the

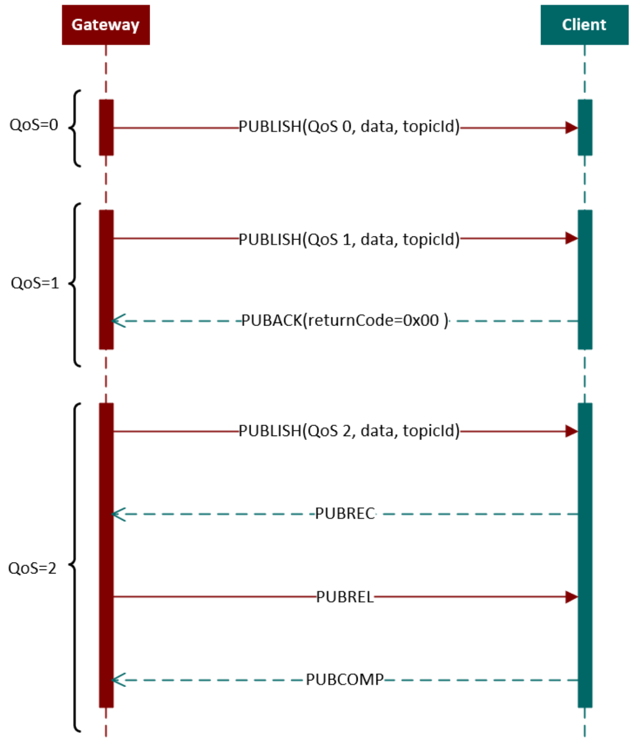

Gateway
Client
QoS=0
PUBLISH(QoS 0, data, topicId)
QoS=1
PUBLISH(QoS 1, data, topicId)
PUBACK(returnCode=0x00)
PUBLISH(QoS 2, data, topicId)
QoS=2
PUBREC
PUBREL
PUBCOMP

FIGURE 53. Gateway sending PUBLISH messages, with QoS 0, QoS 1, and QoS 2, to a client.

PUBLISH messages sent by the gateway with their respective QoS level, as shown in Fig. 56.

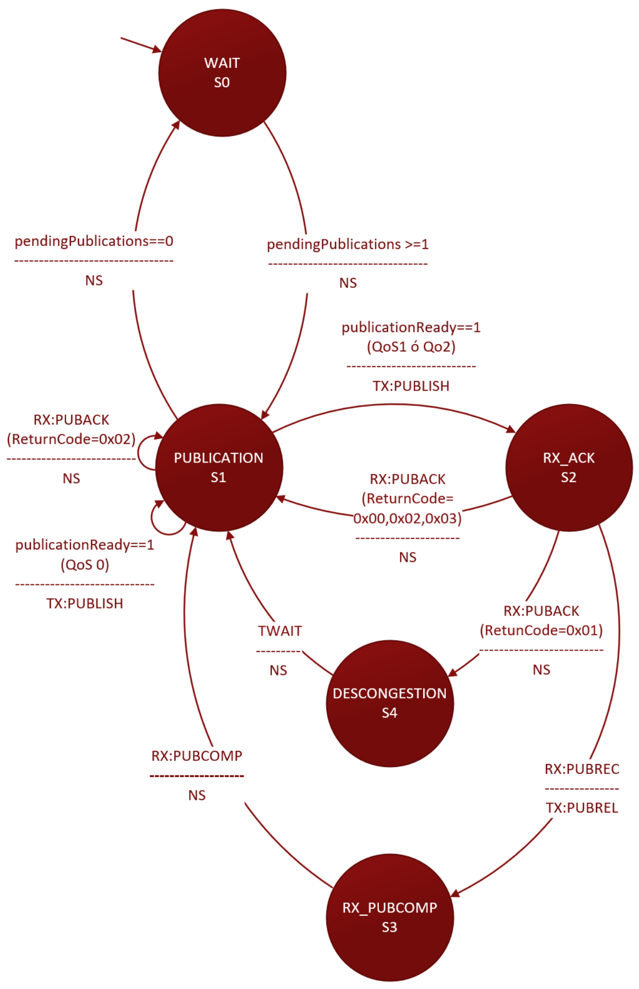
The SDL process in Fig. 57 presents the variable `proc8Count`, which indicates the correct exchange of messages during the procedure. It is observed which state the node should transition to when receiving a publication message with a specific QoS level and topic identifier. Additionally, the variable `iniRegistro` is used, indicating that the procedure failed due to an invalid topic identifier.

I. KEEP ALIVE AND PING PROCEDURE

In an MQTT-SN network, clients execute this procedure to verify the gateway's correct operation. The client must send a PINGREQ message within each keep-alive time interval set by the client during the connection, and the gateway must acknowledge this message by sending a PINGRESP in response. Similarly, a client will respond with a PINGRESP message if it receives a PINGREQ message from the gateway to which it is connected. Fig. 58 shows the sending and receiving of the mentioned messages.

1) DESIGN OF THE MEALY MACHINE FOR GATEWAY OPERATION

As shown in Fig. 59, during this procedure, the gateway node can remain in only two states: i) the WAIT state and ii) the RX_PINGRESP state. When the node is in the initial state, it can receive a message *PINGREQ*, to which it immediately responds with a message *PINGRESP*. Additionally, it may receive the signal that the timer *TKA* (Keep Alive) has



WAIT S0
pendingPublications==0 / NS
pendingPublications >=1 / NS
publicationReady==1 (QoS1 ó Qo2) / TX:PUBLISH
RX:PUBACK (ReturnCode=0x02) / NS
PUBLICATION S1
RX_ACK S2
RX:PUBACK (ReturnCode= 0x00,0x02,0x03) / NS
publicationReady==1 (QoS 0) / TX:PUBLISH
TWAIT / NS
RX:PUBACK (RetunCode=0x01) / NS
DESCONGESTION S4
RX:PUBCOMP / NS
RX:PUBREC / TX:PUBREL
RX_PUBCOMP S3

FIGURE 54. State diagram for the gateway, corresponding to the procedure for publication by the gateway.

expired, which occurs when the client stops sending its message *PINGREQ*. The node enters the RX_PINGRESP state when it receives the *requestReady* = 1 signal, indicating its intention to verify whether the client node is working. The gateway waits for the client's response in the second state and returns to the initial state.

The SDL process in Fig. 60 states that the timer *TKA* must be started before the procedure begins its execution. The variable `tKeepAlive` controls the timer's duration, which is assigned the value contained in the duration field, which must be assigned when the client receives a connection message. The variable is also assigned a 50 percent tolerance, as indicated in the MQTT-SN specification. Likewise, the variable `lostClient` is incorporated and activated when the timer ends. Additionally, the variable `proc9Count` indicates that the correct exchange of messages between the gateway and the client has occurred.

2) DESIGN OF THE MEALY MACHINE FOR THE CLIENT'S OPERATION

In this procedure, as shown in Fig. 61, two states are managed: i) ACTIVE SUPERVISION and ii) RX_PINGRESP.

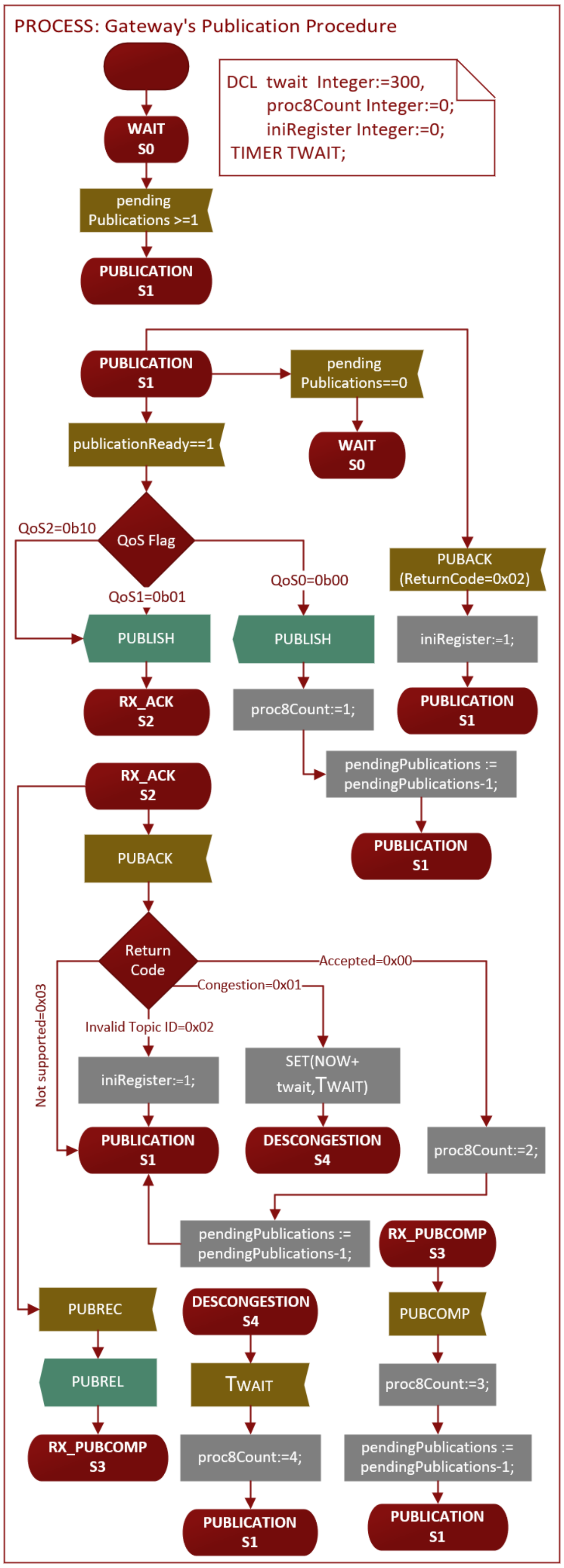


FIGURE 55. SDL process for the gateway, corresponding to the procedure for publication by the gateway.

In the initial state, the node waits for the timer *TKA* to expire and then sends the message *PINGREQ*. The response to the sent message is expected in the state RX_PINGRESP. Once

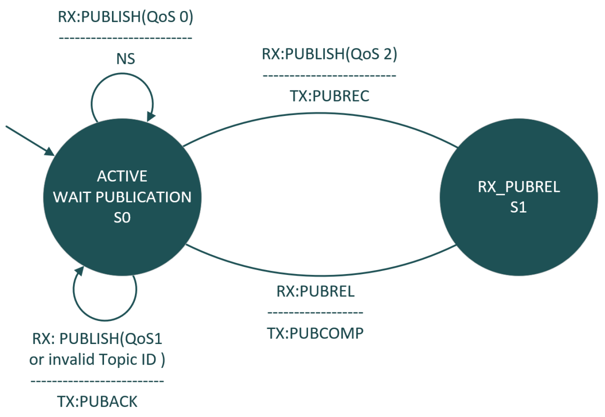


FIGURE 56. State diagram for the client, corresponding to the gateway's publication procedure.

the node receives the *PINGRESP* message, it returns to the initial state, indicating that the gateway operates normally. Additionally, in the initial state, the node must respond to a *PINGREQ* message sent by the gateway.

In the diagram in Fig. 62, the timer *TKA* must also be initialized before starting the procedure. The variable `tKeepAlive` is added, which stores the timer's duration, which must be assigned when the client receives a connection message during the respective procedure. Additionally, the variable `proc9Count` indicates that the correct exchange of messages between the client and the gateway has occurred.

J. CLIENT'S DISCONNECT PROCEDURE

If a client wants to disconnect from the MQTT-SN network, it must send a DISCONNECT message. The gateway then sends another DISCONNECT message to confirm the disconnection, as shown in Fig. 63. In this type of disconnection, initiating a registration procedure is unnecessary if the client must connect to the network again. When a client disconnects, the client's will data and subscriptions persist unless otherwise indicated in the cleanSession flag during a new connection. The gateway sends a DISCONNECT message when a client cannot be recognized. The client that receives this message must establish the connection again, as shown in Fig. 64.

1) MEALY MACHINE DESIGN FOR GATEWAY OPERATION

As shown in Fig. 65, the node must be maintained exclusively in a single state called WAIT DISCONNECTION during this state. In this state, the gateway waits for a *DISCONNECT* message sent by the client when the client wants to disconnect. This message is responded to with another *DISCONNECT* message and then returns to the initial state. A *DISCONNECT* message can be sent when the signal *unknowClient* $==1$ is received. This signal must be set when the gateway cannot identify a received message.

In the SDL process in Fig. 66, the variable `proc10Count` is used to signal that the proper exchange of messages between the gateway and the client has occurred. The variable

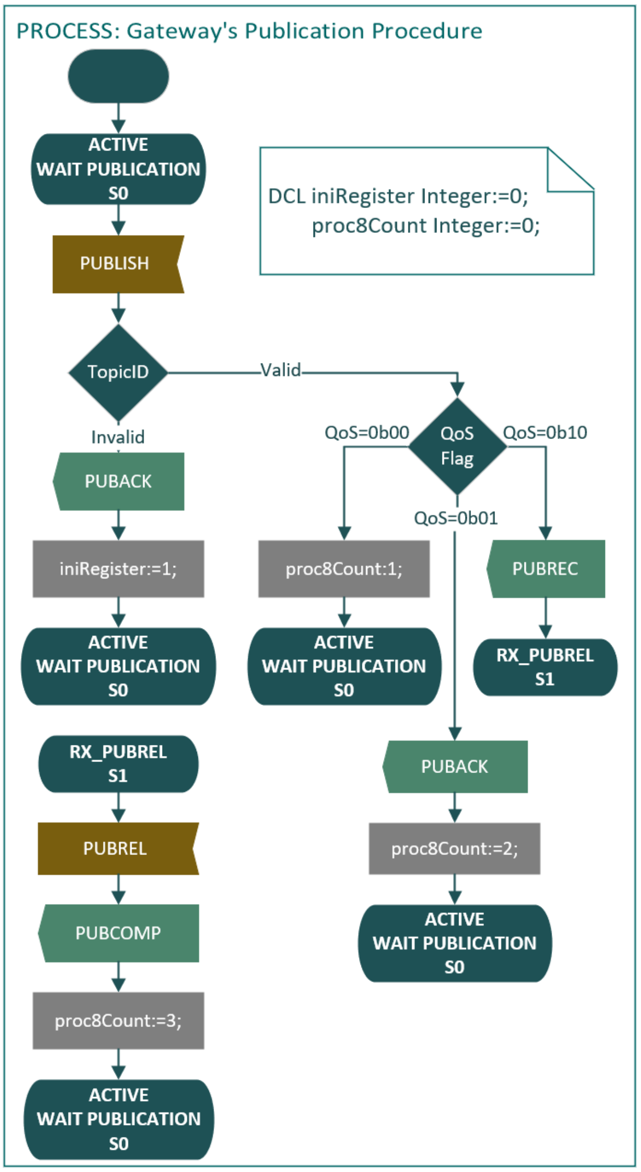


FIGURE 57. SDL process for the client, corresponding to the gateway's publication procedure.

will take one value when a disconnection message is sent due to a customer's request and will take another value when you want to know if a customer has been lost.

2) DESIGN OF THE MEALY MACHINE FOR THE CLIENT'S OPERATION

A client must be found and active to carry out this procedure. In this sense, the client node starts the procedure in an ACTIVE DISCONNECTION state. In this initial state, if the client wants to disconnect, it must wait to receive a *closeConnection* $==1$ signal and then send a *DISCONNECT* message. Subsequently, since the client waits for the gateway to confirm its disconnection, it transitions to the RX_DISCONNECT state. After receiving the disconnection confirmation, the node finally reaches

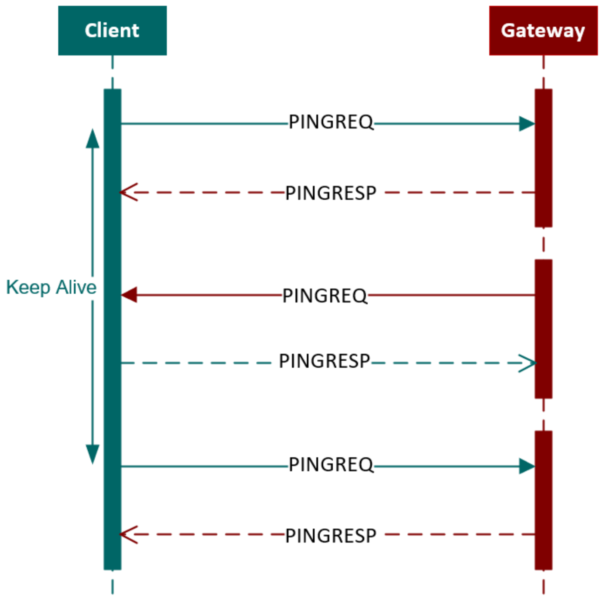


FIGURE 58. Keep alive and PING procedure.

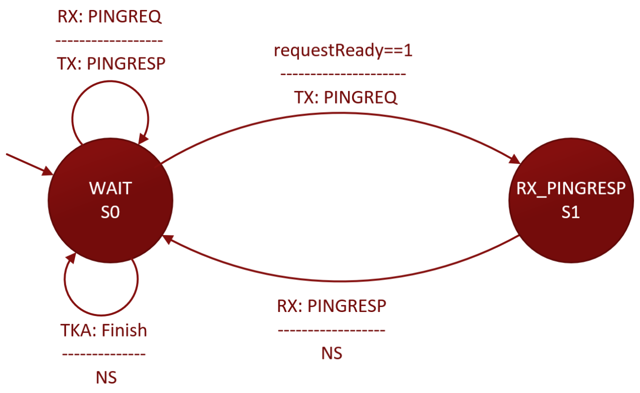


FIGURE 59. State diagram for the gateway, corresponding to the keep alive and ping procedure.

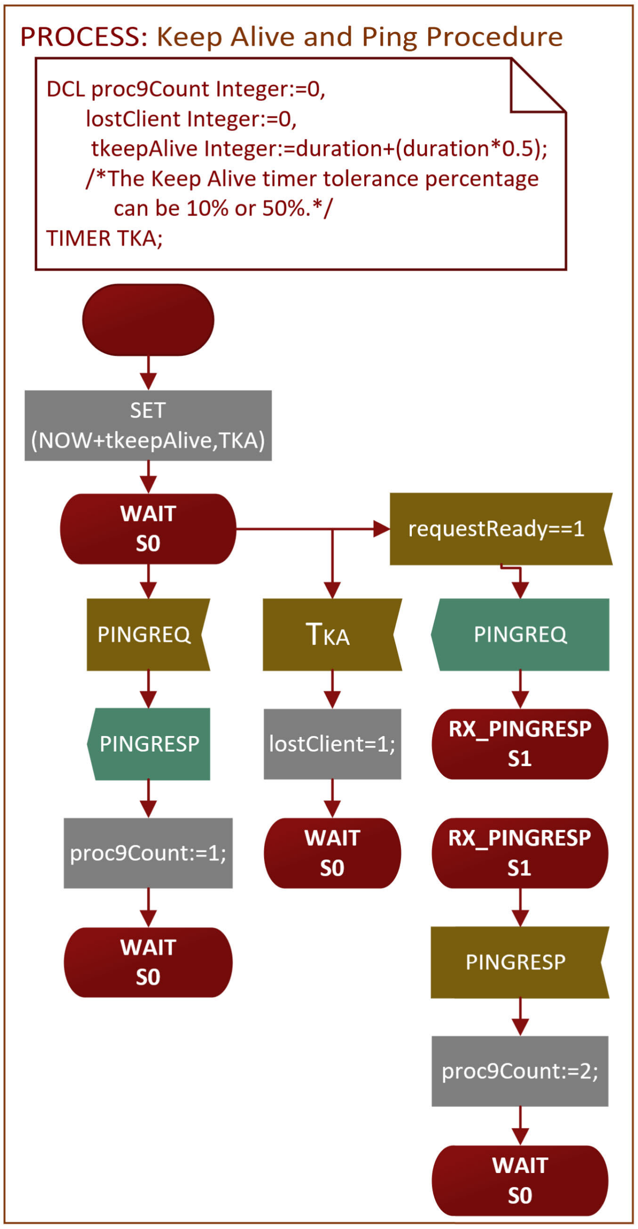


FIGURE 60. SDL process for the gateway, corresponding to the keep alive and ping procedure.

the *DISCONNECTED* state. The client will remain in this state until it establishes a new connection, indicated by the signal *iniConnection* = 1. Additionally, the client may receive a *DISCONNECT* message from the initial state, indicating that the gateway cannot recognize it. Upon receiving the *DISCONNECT* message, the node goes to the state ESTABLISH CONNECTION, referring to the state handled in the client connection configuration procedure. Because if a client receives a *DISCONNECT* message, it must try to connect to the gateway again. Fig. 67 shows the state diagram of this procedure.

In the SDL process in Fig. 68, the variable proc10Count indicates that the correct sequence of messages has been exchanged between the client and gateway.

K. CLIENT'S RETRANSMISSION PROCEDURE

Clients use this procedure when sending unicast messages to the gateway, which require a response. After sending the message and not receiving a response, a retry timer (Tretry) and a retry counter (Nretry) are activated. If no response is received during the timer period, the client retransmits the unicast message, resets the timer, and increments the counter by one. If the expected response is received, the client stops the timer and counter, as indicated in the sequence diagram in Fig. 69. There is a maximum limit for allowed retransmissions; once this limit is exceeded, the client aborts the procedure, assuming that the MQTT-SN connection has been lost. In this case, a new connection must be started, either with the same gateway or another.

1) DESIGN OF THE MEALY MACHINE FOR THE CLIENT'S OPERATION

As shown in Fig. 70, this procedure is exclusive to the client. When the node is in its initial state, designated

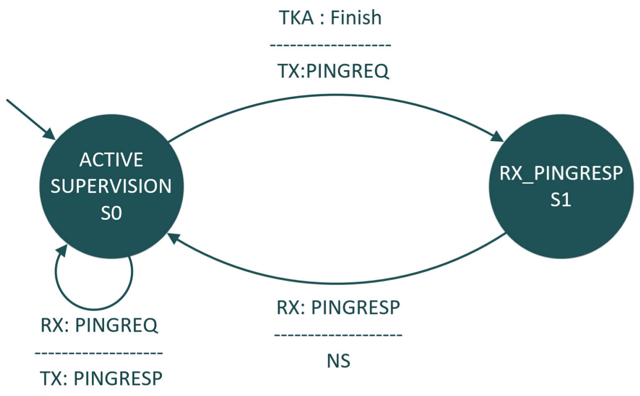


FIGURE 61. State diagram for the client, corresponding to the keep alive and ping procedure.

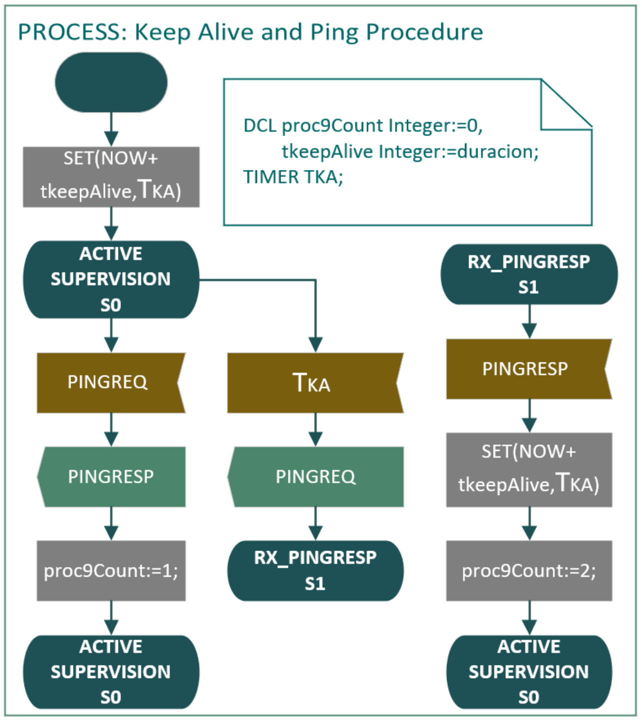


FIGURE 62. SDL process for the client, corresponding to the keep alive and ping procedure.

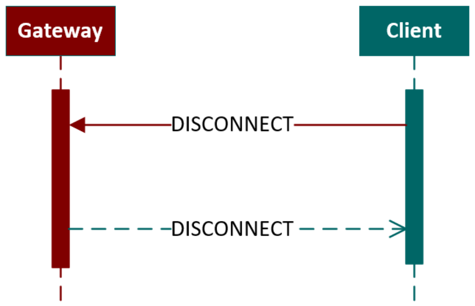


FIGURE 63. Procedure for client disconnection.

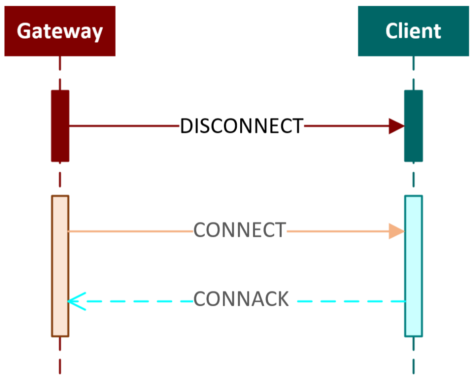


FIGURE 64. Gateway sending a disconnection message to the client.

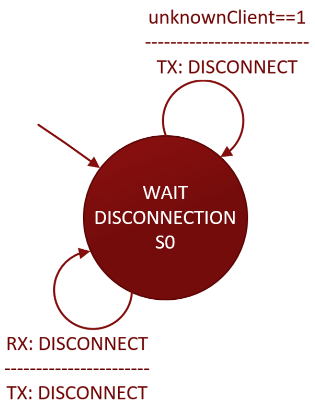


FIGURE 65. State diagram for the gateway, corresponding to the client disconnection procedure.

ACTIVE, it can send a unicast message to the client once it receives the corresponding signal (in this case, the signal $msgUnicastReady == 1$ is expected). If the sent message requires confirmation, the node goes to the RX_ACK state, where it waits to receive the expected message; In that case, it returns to the initial state. If the client returns to the ACTIVE state, you should be able to continue with other procedures. However, if the acknowledgment message is not received, the node must wait for the *TRETRY* timer to expire and retransmit the sent unicast message. The node enters the RX_ACK state until it receives the confirmation message or, failing that, reaches the maximum number of retries and must enter the ESTABLISH CONNECTION state using the $Nretry == Nmaxsignal$.. The client connection configuration procedure must be started if the node reaches the mentioned state.

The SDL process in Fig. 71 shows the activation, start, and subsequent reset of timer *TRETRY*, which occurs when initially sending or retransmitting a Unicast message. The variable `tretry` will control the duration of this timer. Additionally, the variables `Nmax` and `Nretry` are used to keep track of the maximum allowed number of retransmission attempts. The variable `proc11Count` is also used to indicate that the procedure has been completed and that a new connection must begin.

L. SUPPORT FOR ENERGY SAVING

This procedure supports clients that need to conserve power and remain in sleep mode until they have information to

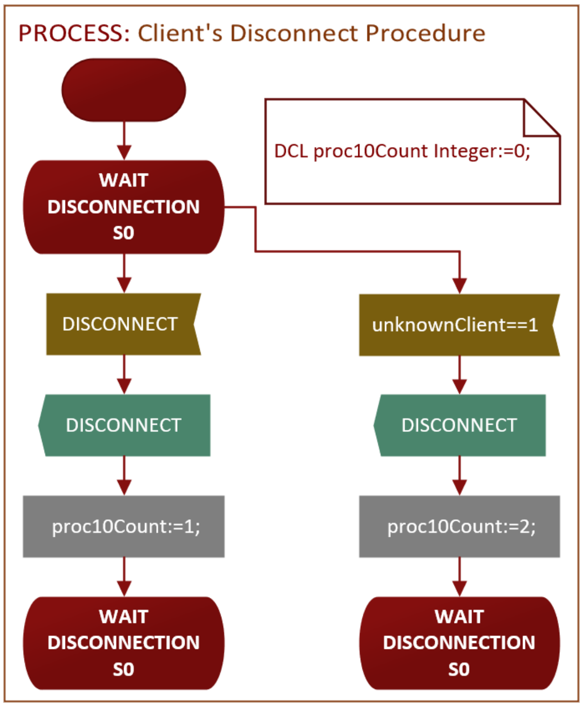


FIGURE 66. SDL process for the gateway, corresponding to the client disconnection procedure.

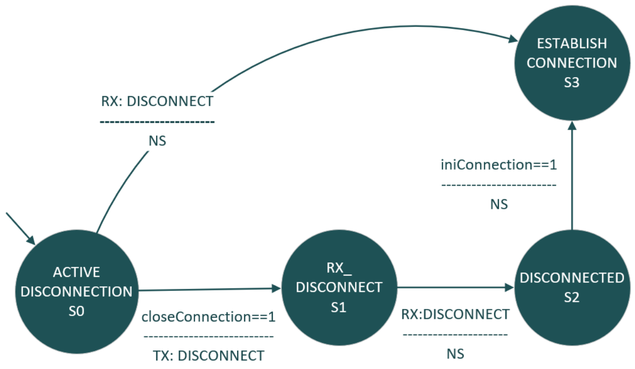


FIGURE 67. State diagram for the client, corresponding to the client disconnection procedure.

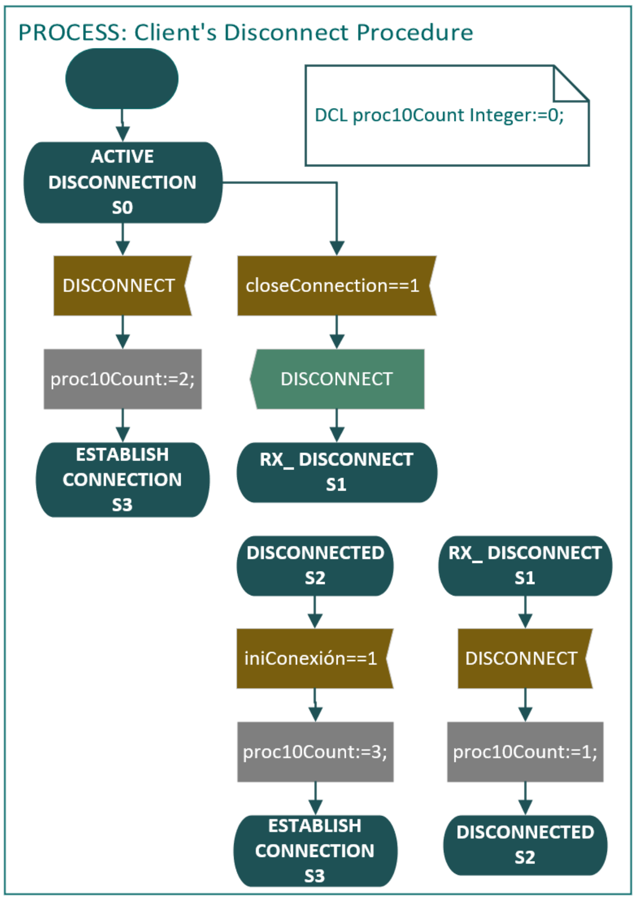


FIGURE 68. SDL process for the client, corresponding to the client disconnection procedure.

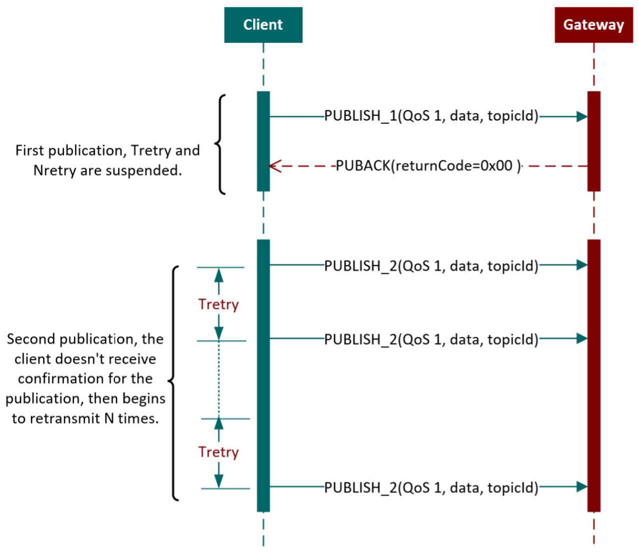

Client
Gateway
First publication, Tretry and Nretry are suspended.
PUBLISH_1(QoS 1, data, topicId)
PUBACK(returnCode=0x00)
Second publication, the client doesn't receive confirmation for the publication, then begins to retransmit N times.
Tretry
Tretry
PUBLISH_2(QoS 1, data, topicId)
PUBLISH_2(QoS 1, data, topicId)
PUBLISH_2(QoS 1, data, topicId)

FIGURE 69. A client making two publications. The first is carried out successfully, while the second receives no response. In this situation, the client retransmit the message until the maximum number of retries allowed is reached.

send or receive. Clients are only activated when they have data to exchange. The gateway stores messages destined for the client in a buffer until the client is activated. From the gateway's perspective, a client can be in one of the following states: active, sleeping, awake, lost, and offline. All states except the offline state are monitored continuously. When a client wants to go to sleep, it must send a DISCONNECT message that includes the duration of the Tsleep timer associated with that client. This message is acknowledged with another DISCONNECT message, followed by the timer's start. The gateway monitors the client's state using its own Tsleep timer, the duration of which is indicated in the client's DISCONNECT message. The client is considered lost if the client does not send any messages before the gateway's Tsleep timer expires. If the gateway receives a PINGREQ before its Tsleep timer expires, it identifies the client that sent it and considers it awake. On the client side,

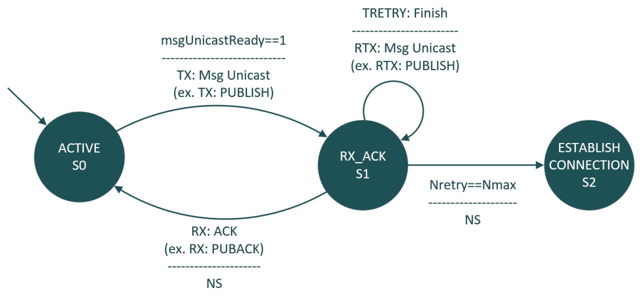


FIGURE 70. State diagram for the client, corresponding to the retransmission procedure.

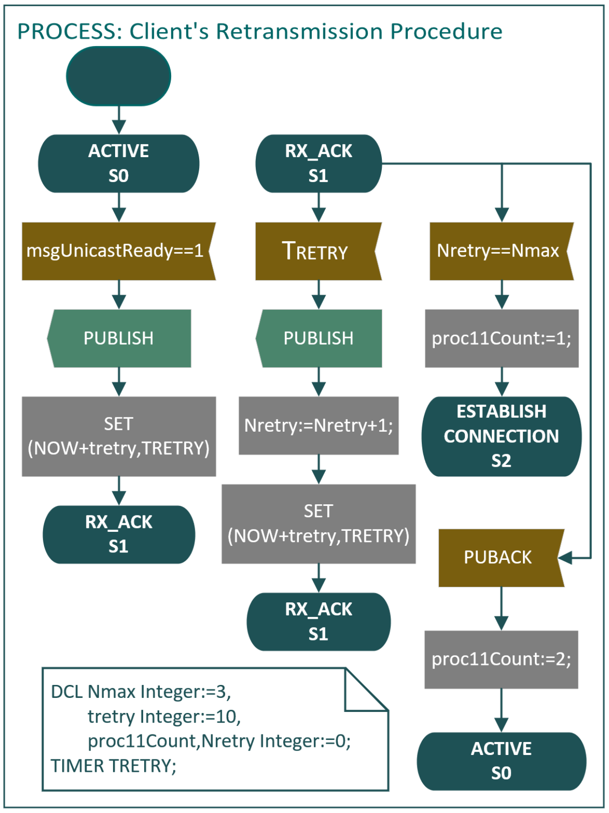


FIGURE 71. SDL process for the client, corresponding to the retransmission procedure.

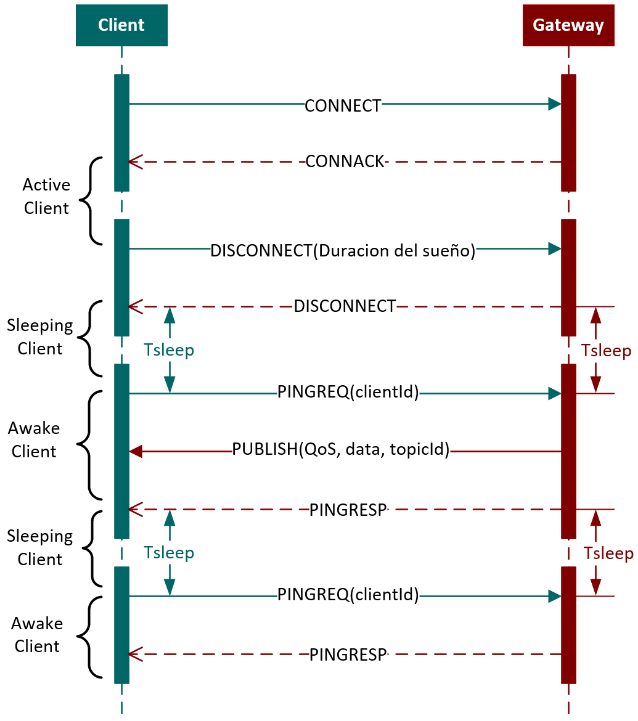


FIGURE 72. Client changing state when it needs to conserve energy.

it switches to the awake state when it sends the PINGREQ message with its respective identifier once its Tsleep ends. During the awake state of the client, a transfer of pending messages occurs, which the gateway completes by sending a PINGRESP message. If there are no pending messages, the gateway immediately sends the PINGRESP message. Upon receipt of the PINGRESP message, the client resets its Tsleep timer and returns to sleep. Simultaneously, the gateway considers the client asleep after sending the PINGRESP and restarts its timer, as shown in Fig. 72.

1) MEALY MACHINE DESIGN FOR GATEWAY OPERATION

For the gateway to support clients that require power savings, the node must manage two states: i) WAIT SUPERVISION (initial state) and ii) PUBLICATION. In the initial state, the node waits for the reception of different signals to determine the client's state. Suppose the gateway receives a *DISCONNECT* message (either with or without the duration field) while in the initial state, it responds with a *DISCONNECT* message to confirm receipt. Once the node has received a *DISCONNECT* message that includes the duration for which the client will remain asleep, it can also receive the timer signal *Ts* (Tsleep). This allows the gateway node to monitor the client while sleeping or in a lost state. The active state of the client is not considered, as this aspect is addressed in the client connection procedure.

The node enters the PUBLICATION state if, while in the initial state, it receives a *PINGREQ* message. This reception assumes the client is awake and can send or receive pending publications using the corresponding procedure. If there are no pending publications, the node receives the signal $pendingPublications == 0$ and immediately sends a message *PINGRESP*. The client is then considered asleep, and the gateway node returns to the initial state, where it continues to monitor the client's state. Fig. 73 shows the state diagram of this procedure. The gateway does not monitor the disconnected state. A client enters this state after sending a *DISCONNECT* message and receiving confirmation, as outlined in the client disconnection procedure.

The SDL process in Fig. 74 illustrates the establishment of the timer *Ts*, whose duration depends on the information contained in the Duration field of the *DISCONNECT* message when a client indicates that it will go to sleep. The variable `durationS` will contain the value of the mentioned

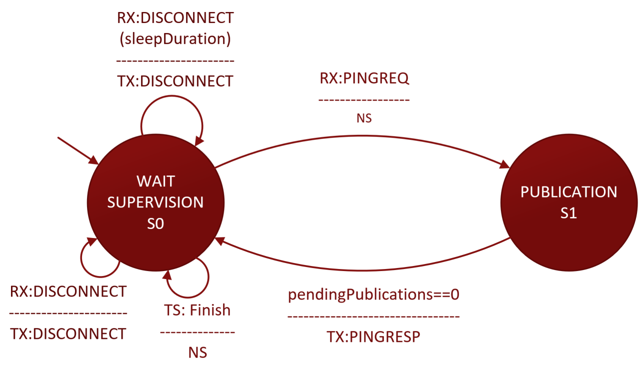


FIGURE 73. State diagram for the gateway, corresponding to the energy saving support procedure.

field, to which its respective tolerance will be added. Therefore, the timer will operate shortly after receiving the *DISCONNECT* message. Additionally, the timer will reset when the gateway verifies that it has no pending publications. Also, the different messages that the gateway can send to know the client's status are presented. For this reason, the variable `staClient` assigns a value that indicates the client's state: 1 if it is asleep, 2 if it is awake, and 3 if lost. In this case, the variable `proc12Count` indicates whether a client has disconnected.

2) DESIGN OF THE MEALY MACHINE FOR THE CLIENT'S OPERATION

As shown in Fig. 75, when a client node can manage its energy consumption, it can enter various states. For this purpose, a diagram is considered that includes the following states for said node: ACTIVE, SLEEP CONFIRMATION, SLEEP, AWAKE, RX_DISCONNECT, DISCONNECTED, and ESTABLISH CONNECTION. From its initial state, the node can receive the signals $saveEnergy == 1$ or $closeConnection == 1$, and send a message *DISCONNECT* if it seeks to move to the state SLEEP CONFIRMATION or RX DISCONNECT, respectively. It is important to note that the *DISCONNECT* message sent to save energy must include the 'Duration' field, indicating the time the node will remain sleeping. In case of receiving a *DISCONNECT* message from the state SLEEP CONFIRMATION, the node will go to the state SLEEP. During this state, the node waits for the timer that controls the sleep duration, Ts, to expire and then enters the AWAKE state after sending the *PINGREQ* message. During the AWAKE state, the node can receive or send publications until it receives the *PINGRESP* message from the gateway. Subsequently, the node returns to the SLEEP state, where it waits for timer Ts to expire again. During the SLEEP state, the node can adjust the sleep duration by sending another message *DISCONNECT*, and the gateway must confirm the new duration of timer Ts. Additionally, the node can return to the ACTIVE or DISCONNECT state upon receiving the signals $saveEnergy == 0$ or $closeConnection == 1$, respectively. To return to the ACTIVE state, you must first go through the ESTABLISH CONNECTION state, where the client connection procedure should be executed.

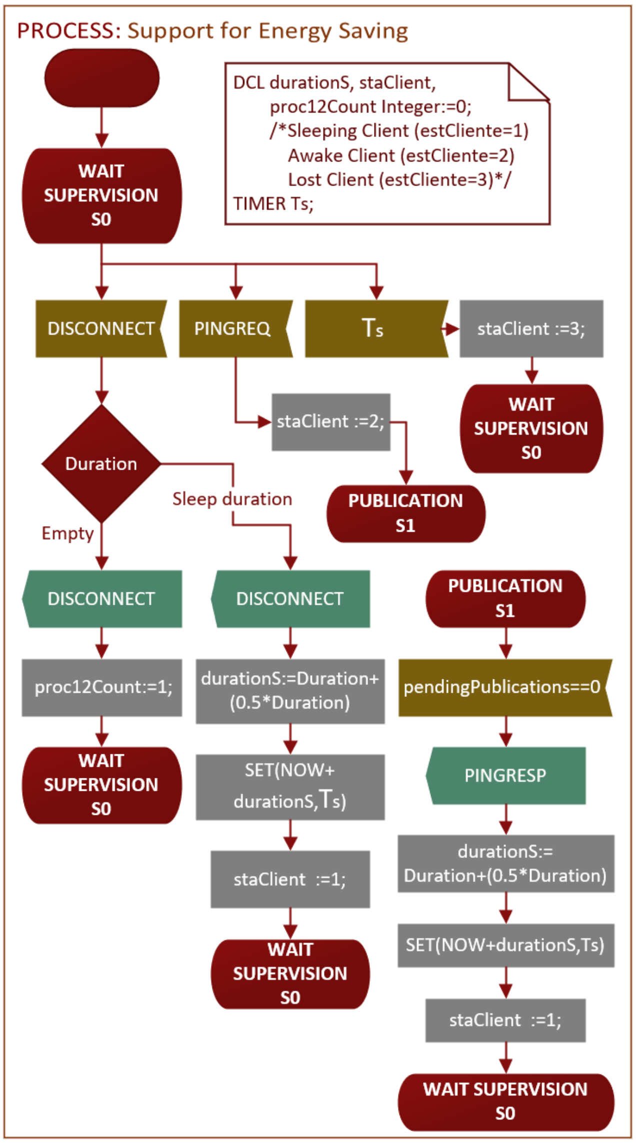


FIGURE 74. SDL process for the gateway, corresponding to the energy saving support procedure.

The node returns to the active state if the connection is accepted. Because the procedure above was already explained previously, only the state ESTABLISH CONNECTION is used, during which the signal $proc2Count = 1$ is expected to be received, which indicates that the client has established a successful connection. Finally, to move to the DISCONNECT state, the node must receive a confirmation *DISCONNECT* message sent by the gateway during the RX_DISCONNECT state. If the node requires to return to the ACTIVE state, it must initiate a new connection using the signal $iniConnection == 1$.

The SDL process in Fig. 76, shows the node responds to various inputs, whether messages or internal signals. The timer Ts and the variable `durationS` are introduced to control the time the node will remain in the SLEEP state. The variable `durationS` stores the duration of the timer, which starts when the node enters the SLEEP state and

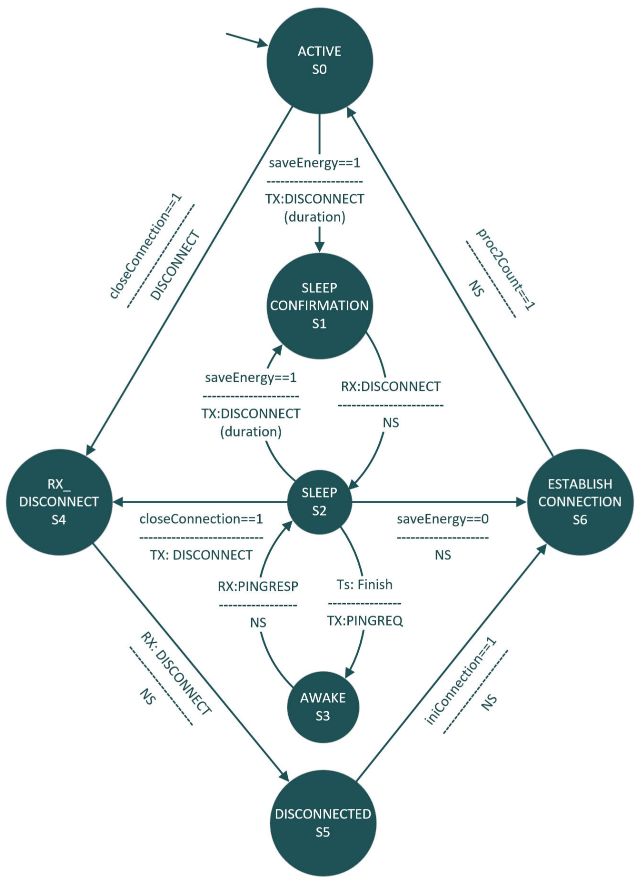


FIGURE 75. State diagram for the client, corresponding to the energy-saving support procedure.

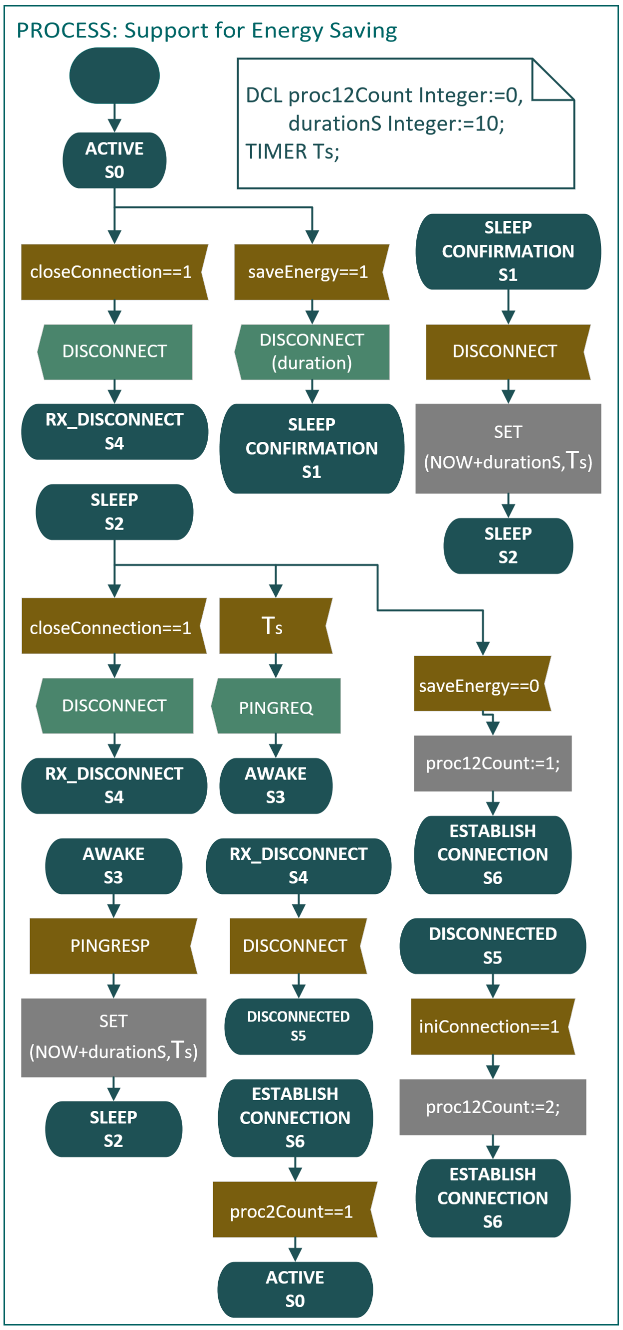


FIGURE 76. SDL process for the client, corresponding to the support procedure for energy savings.

restarts every time it transitions from the AWAKE state to the SLEEP state again. A constant state change is shown between SLEEP and AWAKE, as well as the transition process of these states through the signals $closeConnection == 1$ and $saveEnergy == 0$. The first signal brings the node to the DISCONNECTED state, while the second returns the node to the ACTIVE state. In both cases, the variable `proc12Count` is used to indicate that the node no longer needs to conserve energy, assigning it the value 1 when it goes from the SLEEP state to the ESTABLISH CONNECTION state, or the value 2 when it goes from the DISCONNECTED state to the ESTABLISH CONNECTION state. The procedure is considered to have met its objective when the variable takes on any of these values. From the ESTABLISH CONNECTION state, the node returns to the initial state as soon as it receives the signal indicating that it has successfully established a connection.

V. REPRESENTATION OF THE MQTT-SN PROTOCOL OPERATING ON THE IEEE 802.15.4 STANDARD WITHIN THE NODE USING SDL BLOCKS AND PROCESSES

The Mealy machines obtained are beneficial for developing SDL diagrams of the protocol procedures because the SDL diagrams handle the same states and signals. Each obtained SDL diagram represents an SDL process type entity (SDL PROCESS). Once the SDL PROCESS entities for each MQTT-SN procedure have been obtained, starting from their respective Mealy machine, and clear about the different signals that each node needs to achieve a state transition, it is possible to represent an SDL system that better shows how the exchange of messages between the client node and gateway node works through several block-type entities (BLOCK) connected through channels. Each of the blocks obtained

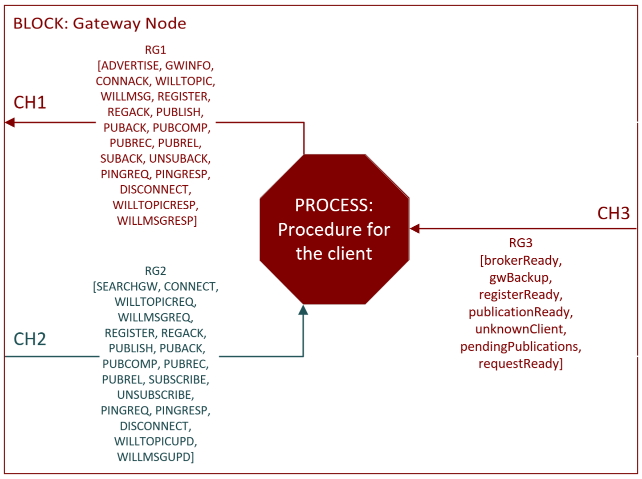

BLOCK: Gateway Node
RG1
[ADVERTISE, GWINFO, CONNACK, WILLTOPIC, WILLMSG, REGISTER, REGACK, PUBLISH, PUBACK, PUBCOMP, PUBREC, PUBREL, SUBACK, UNSUBACK, PINGREQ, PINGRESP, DISCONNECT, WILLTOPICRESP, WILLMSGRESP]
CH1
PROCESS: Procedure for the client
CH3
RG3
[brokerReady, gwBackup, registerReady, publicationReady, unknownClient, pendingPublications, requestReady]
RG2
[SEARCHGW, CONNECT, WILLTOPICREQ, WILLMSGREQ, REGISTER, REGACK, PUBLISH, PUBACK, PUBCOMP, PUBREC, PUBREL, SUBSCRIBE, UNSUBSCRIBE, PINGREQ, PINGRESP, DISCONNECT, WILLTOPICUPD, WILLMSGUPD]
CH2

FIGURE 77. **SDL block type entity representing the gateway node.**

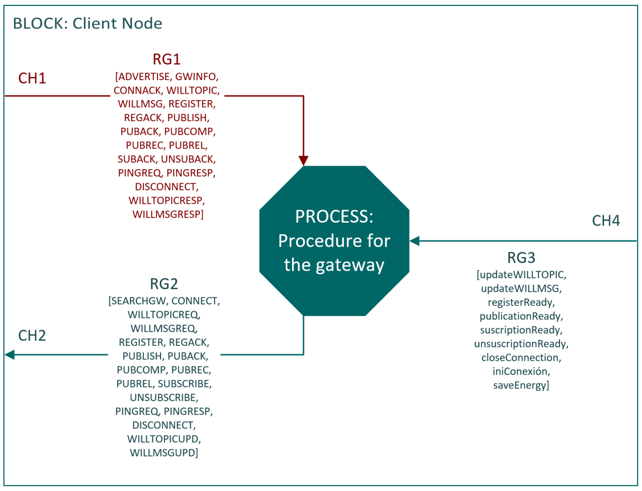

BLOCK: Client Node
RG1
CH1
[ADVERTISE, GWINFO, CONNACK, WILLTOPIC, WILLMSG, REGISTER, REGACK, PUBLISH, PUBACK, PUBCOMP, PUBREC, PUBREL, SUBACK, UNSUBACK, PINGREQ, PINGRESP, DISCONNECT, WILLTOPICRESP, WILLMSGRESP]
PROCESS: Procedure for the gateway
CH4
RG3
[updateWILLTOPIC, updateWILLMSG, registerReady, publicationReady, suscriptionReady, unsuscriptionReady, closeConnection, iniConexión, saveEnergy]
RG2
[SEARCHGW, CONNECT, WILLTOPICREQ, WILLMSGREQ, REGISTER, REGACK, PUBLISH, PUBACK, PUBCOMP, PUBREC, PUBREL, SUBSCRIBE, UNSUBSCRIBE, PINGREQ, PINGRESP, DISCONNECT, WILLTOPICUPD, WILLMSGUPD]
CH2

FIGURE 78. **SDL block type entity representing the client node.**

contains an SDL process inside. Each entity that will be part of the system is explained below.

A. PROCESS

As seen in Fig. 77 and Fig. 78, a process is part of an SDL block. The process within the block refers to each of the previously designed MQTT-SN procedures and its finite state machine. In the case of the process called gateway procedure, it is connected to two input channels called RG2 and RG3 and one output channel called RG1. The RG1 and RG2 channels send and receive MQTT-SN messages, while the RG3 channel receives the signals generated within the gateway node. The client procedure entity contains two input channels called RG1 and RG3 and an output channel called RG2.

B. GATEWAY NODE BLOCK

This entity, as shown in Fig. 77, represents the gateway node and is composed of a process entity (gateway procedure), three channels (RG1, RG2, and RG3), and three inputs (CH1, CH2, and CH3), which allow the process with the block and later with the system. This block connects with the block that represents the client node or with entities external to the system through the channels of the main entity.

C. CLIENT NODE BLOCK

This entity, as shown in Fig. 78, represents the client node and is composed of a process entity (client procedure), three channels (RG1, RG2, and RG3), and three inputs (CH1, CH2, and CH4) that allow the process to interact with the block and later with the system. This block connects with the block that represents the gateway node or with entities external to the system through the channels of the main entity.

D. SYSTEM

The main system or entity, shown in Fig. 79, consists of a text box, four channels (CH1, CH2, CH3, and CH4), and two blocks (CLIENT NODE and GATEWAY NODE). All signals passing through all channels in the system are declared within the text box. Signals include MQTT-SN messages and other events that cause a change of states within the respective entities. The blocks representing the client node and gateway node communicate through channels CH1 and CH2, which carry MQTT-SN messages and refer to the wireless medium through which the RCB256RFR2 nodes communicate. Finally, channels CH3 and CH4 are responsible for transporting the rest of the signals the nodes will use. It is considered that these signals originate outside the system since they depend on an external agent that must interact with the node's button.

1) IMPLEMENTATION OF STATES IN THE NODES

The states described in Section I have been coded using the Atmel Studio 7 integrated development environment. Each node's complete code is found in this repository [26].

2) STATE MACHINE SIMULATIONS

The JFLAP state machine simulator offers the ability to simulate various types of state machines, including the Mealy machine, thus allowing the different state changes of the nodes to be recreated [27]. An input string (entered manually or via a text file) that represents the various MQTT-SN message sequences that the node must interact with is required to perform the simulation. The interface provided by JFLAP, as shown in Fig. 80, facilitates the creation and simulation of a state machine, which in turn allows verifying the validity of the state machines defined for each procedure.

VI. RESULTS

The SmartRF and packet sniffer tools are used to verify the operation of the stated and encoded states in the nodes. These tools capture messages exchanged between the nodes in the WSN.

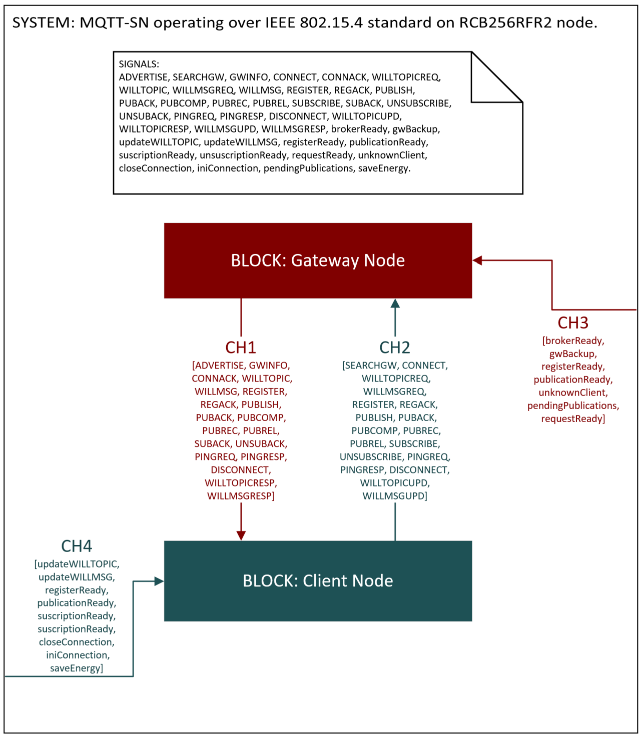

SYSTEM: MQTT-SN operating over IEEE 802.15.4 standard on RCB256RFR2 node.
SIGNALS:
ADVERTISE, SEARCHGW, GWINFO, CONNECT, CONNACK, WILLTOPICREQ, WILLTOPIC, WILLMSGREQ, WILLMSG, REGISTER, REGACK, PUBLISH, PUBACK, PUBCOMP, PUBREC, PUBREL, SUBSCRIBE, SUBACK, UNSUBSCRIBE, UNSUBACK, PINGREQ, PINGRESP, DISCONNECT, WILLTOPICUPD, WILLTOPICRESP, WILLMSGUPD, WILLMSGRESP, brokerReady, gwBackup, updateWILLTOPIC, updateWILLMSG, registerReady, publicationReady, suscriptionReady, unsuscriptionReady, requestReady, unknownClient, closeConnection, iniConnection, pendingPublications, saveEnergy.
BLOCK: Gateway Node
CH1
[ADVERTISE, GWINFO, CONNACK, WILLTOPIC, WILLMSG, REGISTER, REGACK, PUBLISH, PUBACK, PUBCOMP, PUBREC, PUBREL, SUBACK, UNSUBACK, PINGREQ, PINGRESP, DISCONNECT, WILLTOPICRESP, WILLMSGRESP]
CH2
[SEARCHGW, CONNECT, WILLTOPICREQ, WILLMSGREQ, REGISTER, REGACK, PUBLISH, PUBACK, PUBCOMP, PUBREC, PUBREL, SUBSCRIBE, UNSUBSCRIBE, PINGREQ, PINGRESP, DISCONNECT, WILLTOPICUPD, WILLMSGUPD]
CH3
[brokerReady, gwBackup, registerReady, publicationReady, unknownClient, pendingPublications, requestReady]
CH4
[updateWILLTOPIC, updateWILLMSG, registerReady, publicationReady, suscriptionReady, suscriptionReady, closeConnection, iniConnection, saveEnergy]
BLOCK: Client Node

FIGURE 79. SDL main entity representing the MQTT-SN protocol on the node.

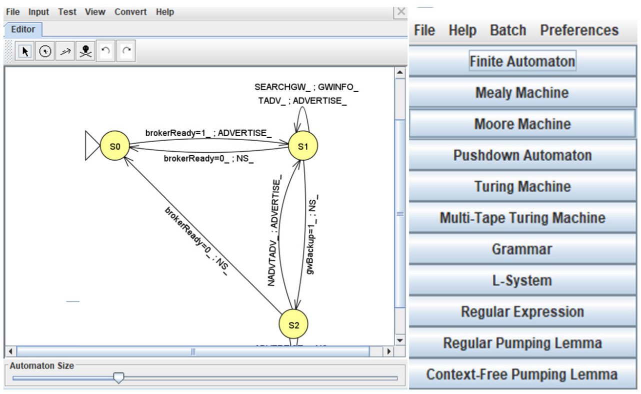


FIGURE 80. Simulation of the procedure for gateway announcement and discovery.

A. TEST SCENARIO

The network consists of four RCB256RFR2 nodes, the edge nodes containing the code necessary to represent their respective state machines. The two internal nodes carry the MQTT-SN messages generated by the edge nodes (client and gateway), thus forming a linear topology. Each node has a unique address that it uses to communicate with each other, as shown in Fig. 81. The client node (SRC_ADDR=0 × 004) communicates with the nearest intermediate node (SRC_ADDR=0 × 003), while the gateway node (SRC_ADDR=0 × 001) communicates with the nearest intermediate node (SRC_ADDR =0 × 002).

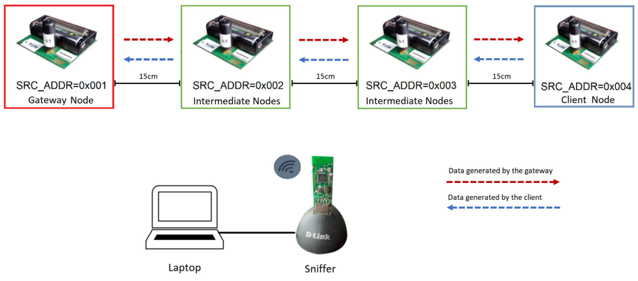
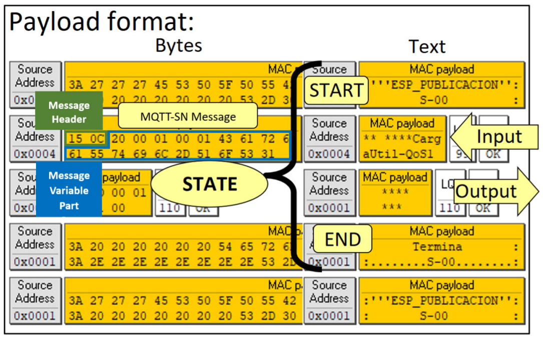


FIGURE 81. Test scenario.



```
Source   MAC p                                  Source   MAC payload
Address  3A 27 27 27 45 53 50 5F 50 55 4        Address  '''ESP_PUBLICACION'':
0x0         20 20 20 20 20 20 53 2D 3                      S-00         :

Sou                                             Source   MAC payload
Address  15 0C 20 00 01 00 01 43 61 72 6        Address  ** ****Carg        9   OK
0x0004   61 55 74 69 6C 2D 51 6F 53 31          0x0004   aUtil-QoS1

Sou         payload                             Source   MAC payload   LQ
Add          00 01                              Address    ****
0x0          00         110   OK                0x0001     ***         110   OK

Source   MAC p                                  Source   MAC payload
Address  3A 20 20 20 20 20 20 54 65 72 6        A           Termina       :
0x0001   3A 2E 2E 2E 2E 2E 2E 2E 2E 53 2D       0x0001   :........S-00........:

Source   MAC p                                  Source   MAC payload
Address  3A 27 27 27 45 53 50 5F 50 55 42       Address  :'''ESP_PUBLICACION'':
0x0001   3A 20 20 20 20 20 20 20 53 2D 30       0x0001   :        S-00         :
```

FIGURE 82. Status verification by SmartRF packet sniffer.

B. MONITORING TOOL CONFIGURATION

The SmartRF Packet Sniffer monitoring tool monitors a node's status. This tool captures and displays the fields of 802.15.4 frames in a graphical interface, including the MQTT-SN messages generated and sent by the nodes and the frame's payload field in text or hexadecimal format. Due to the inclusion of messages indicating the node's status, these messages are only visible when the payload field is displayed in text format in the Sniffer application. On the other hand, MQTT-SN messages are only visible when the payload field is displayed in hexadecimal format.

C. STATUS VERIFICATION BY SMARTRF PACKET SNIFFER

To identify the state a node is in, the node sends a message indicating that state and the node remains in this state until it receives an IEEE 802.15.4 frame containing an MQTT-SN message. The node may transmit a response message and perform a state transition depending on the case. Before performing the state transition, the node sends another message indicating that the previous state has ended. Fig. 82 shows the messages that mark the start and end of a state.

D. VERIFICATION OF MQTT-SN MESSAGES USING SMARTRF PACKET SNIFFER

When MQTT-SN messages are used as inputs and outputs of a state machine, they must be placed between the messages that mark the start and end of a state. The messages are only noticeable if the IEEE 802.15.4 frame payload display format is set to hexadecimal. Because the Sniffer cannot categorize the fields of MQTT-SN messages, it is necessary to manually review the relevant fields to verify which messages

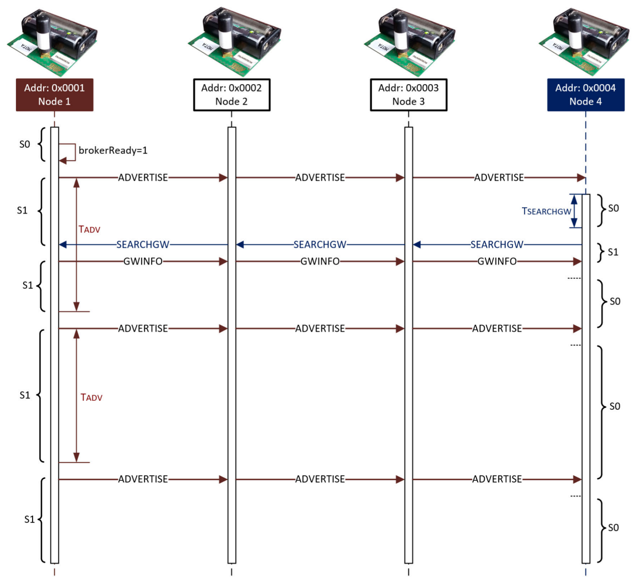


FIGURE 83. MQTT-SN messages used in the test of the gateway announcement and discovery procedure.

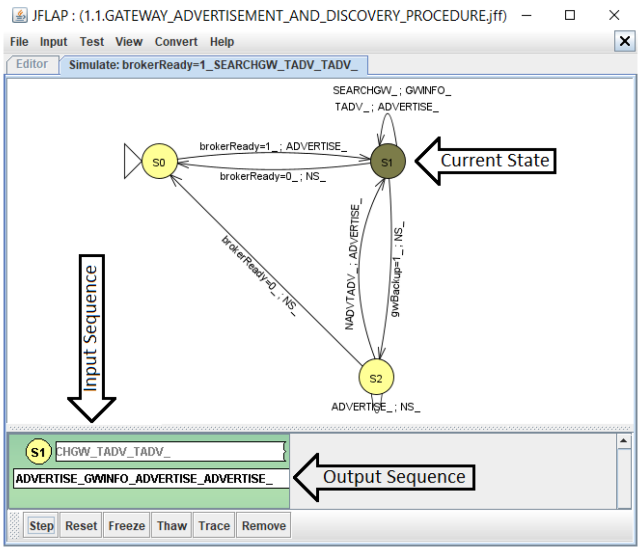


FIGURE 84. Simulation (Gateway) of the gateway announcement and discovery procedure.

enter and leave the node. Only the first two bytes of the frame's payload field containing the MQTT-SN message must be examined to determine the message type. The first byte indicates the length of the message, while the second indicates its type. These two bytes comprise the fixed header, while the remaining bits represent the variable part of the MQTT-SN message. Fig. 82 illustrates how to identify the fields of MQTT-SN messages.

E. TESTS PERFORMED

The client and gateway nodes generate messages between both devices to verify their correct operation. These sequences are the same as those entered into the simulator. Although the simulator returns a sequence of messages according to the procedure, a state transition does not always generate an output message. In such cases, a pair of characters (NS) indicates the absence of output or response upon receipt of a specific message or signal. The output sequences of the simulator are similar to the MQTT-SN message sequences sent by the nodes. Next, the results of the sniffer captures are presented, showing the inputs, outputs, and states managed by the client and gateway nodes. The captures made are compared with the results of the simulations.

1) TESTING OF THE GATEWAY ANNOUNCEMENT AND DISCOVERY PROCEDURE

Figure 88 shows the MQTT-SN messages used in the test of the gateway announcement and discovery procedure. The analysis was performed from both the gateway and the client side.

- **Gateway side:** For this procedure, the following input sequence has been selected:

```
brokerReady=1\_SEARCHGW\_TADV\_TADV\_
```

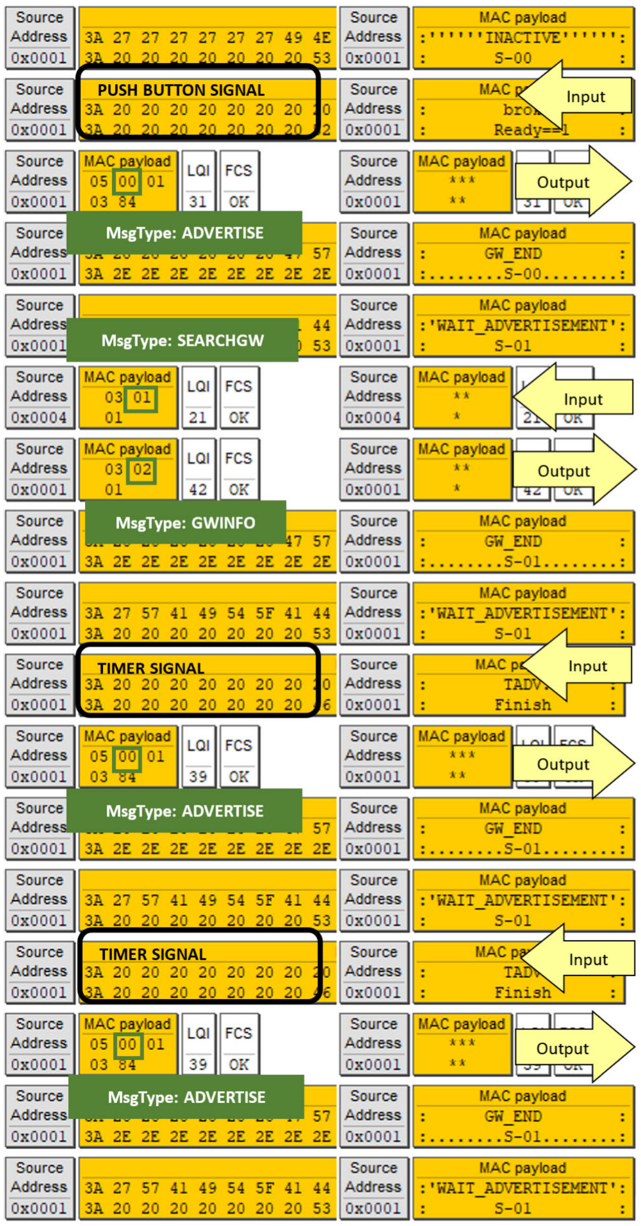
This sequence consists of a signal indicating that the broker is ready to operate, a message querying the existence of a gateway, and two timer signals that send warning messages.

The simulation of this procedure shows the various states the node goes through upon receiving the input sequence, which includes several signals that the node must generate internally. The signal brokerReady=1 in the sequence allows the node to advance from the initial state (S0) to the WAITING ADVERTISEMENT state (S1), where it will continue to send several ADVERTISE messages upon receiving signals indicating the expiration of a TADV timer. The state machine remains in state S1 upon receiving the sequence above. In this state, during which ADVERTISE messages can be sent periodically, the device must be maintained to fulfill its objective in this procedure. The sequence was selected to illustrate this aspect. Fig. 84 shows the output sequence emitted by the simulator.

Figure 85 shows that the node behaves consistently with the simulation when receiving the first signal of the input sequence. The node goes from the INACTIVE state (S0) to the WAITING ADVERTISEMENT state (S1), which remains while receiving the TADV timer signals. State S2 is only reached when the corresponding configuration of the gateway node is performed. This configuration allows the signal gwBackup=1 to be sent to the sensor node through the button, but this state was not reached in this test. The internal signal brokerReady=1, which the broker should send, is generated by the node push button, while the TADV signals are generated automatically at the end of the previously configured timers. The sequences of signals, messages received, and messages leaving the node are shown below.

Input sequence:

```
brokerReady=1, SEARCHGW, TADV, TADV
```



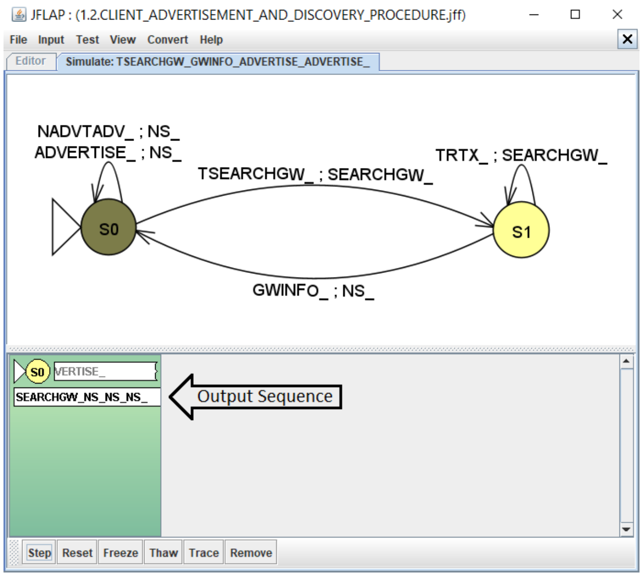
```
Source Address 0x0001 | 3A 27 27 27 27 27 27 49 4E | Source Address 0x0001 | MAC payload :''''''INACTIVE'''''':
                      | 3A 20 20 20 20 20 20 20 53 |                       | :        S-00        :
Source Address 0x0001 | 3A 20 20 20 20 20 20 20 20 | Source Address 0x0001 | MAC p : bro
                      | 3A 20 20 20 20 20 20 20 2  |                       | :     Ready==1       :
Source Address 0x0001 | MAC payload 05 00 01 03 84 | LQI 31 | FCS OK | Source Address 0x0001 | MAC payload *** **
Source Address 0x0001 | 57                         | Source Address 0x0001 | MAC payload :      GW_END        :
                      | 3A 2E 2E 2E 2E 2E 2E 2E 2E |                       | :........S-00........:
Source Address 0x0001 | 44                         | Source Address 0x0001 | MAC payload :'WAIT_ADVERTISEMENT':
                      | 53                         |                       | :        S-01        :
Source Address 0x0004 | MAC payload 03 01 01       | LQI 21 | FCS OK | Source Address 0x0004 | MAC payload ** *
Source Address 0x0001 | MAC payload 03 02 01       | LQI 42 | FCS OK | Source Address 0x0001 | MAC payload ** *
Source Address 0x0001 | 47 57                      | Source Address 0x0001 | MAC payload :      GW_END        :
                      | 3A 2E 2E 2E 2E 2E 2E 2E 2E |                       | :........S-01........:
Source Address 0x0001 | 3A 27 57 41 49 54 5F 41 44 | Source Address 0x0001 | MAC payload :'WAIT_ADVERTISEMENT':
                      | 3A 20 20 20 20 20 20 20 53 |                       | :        S-01        :
Source Address 0x0001 | 3A 20 20 20 20 20 20 20 20 | Source Address 0x0001 | MAC p : TADV
                      | 3A 20 20 20 20 20 20 20 6  |                       | :      Finish       :
Source Address 0x0001 | MAC payload 05 00 01 03 84 | LQI 39 | FCS OK | Source Address 0x0001 | MAC payload *** **
Source Address 0x0001 | 57                         | Source Address 0x0001 | MAC payload :      GW_END        :
                      | 3A 2E 2E 2E 2E 2E 2E 2E 2E |                       | :........S-01........:
Source Address 0x0001 | 3A 27 57 41 49 54 5F 41 44 | Source Address 0x0001 | MAC payload :'WAIT_ADVERTISEMENT':
                      | 3A 20 20 20 20 20 20 20 53 |                       | :        S-01        :
Source Address 0x0001 | 3A 20 20 20 20 20 20 20 20 | Source Address 0x0001 | MAC pa : TADV
                      | 3A 20 20 20 20 20 20 20 6  |                       | :      Finish       :
Source Address 0x0001 | MAC payload 05 00 01 03 84 | LQI 39 | FCS OK | Source Address 0x0001 | MAC payload *** **
Source Address 0x0001 | 57                         | Source Address 0x0001 | MAC payload :      GW_END        :
                      | 3A 2E 2E 2E 2E 2E 2E 2E 2E |                       | :........S-01........:
Source Address 0x0001 | 3A 27 57 41 49 54 5F 41 44 | Source Address 0x0001 | MAC payload :'WAIT_ADVERTISEMENT':
                      | 3A 20 20 20 20 20 20 20 53 |                       | :        S-01        :
```

FIGURE 85. Sniffing (gateway-side) of the gateway announcement and discovery procedure.

Output sequence:

```
ADVERTISE, GWINFO, ADVERTISE, ADVERTISE
```

- **Client side:** The simulation of this procedure shows the various states the node goes through when receiving the input sequence. The first received signal allows the device state machine to advance from the DISCOVERY state (S0) to the RX_GWINFO state (S1). In this last state, the device waits for the arrival of the GWINFO message. Once this message is received, the node returns to the initial state S0. Once it has returned to the initial state, two ADVERTISE messages are received, and upon receiving these messages, the node remains in the



JFLAP : (1.2.CLIENT_ADVERTISEMENT_AND_DISCOVERY_PROCEDURE.jff)

Simulate: TSEARCHGW_GWINFO_ADVERTISE_ADVERTISE_

SEARCHGW_NS_NS_NS_

FIGURE 86. Simulation (Client) of the gateway announcement and discovery procedure.

initial state. This sequence allows for the verification of the operation of the procedure when the node receives several ADVERTISE messages. Fig. 86 shows the output sequence emitted by the simulator, where only one SEARCHGW message is emitted, followed by several "NS" symbols that indicate the absence of signals to emit.

The sniffing procedure shows that the node behaves similarly to what was indicated in the simulation upon receiving the same sequence. Upon receiving the timer signal, the node advances from the initial state (S0) to the RX_GWINFO state (S1) after sending a SEARCHGW message. Subsequently, it returns to the initial state after receiving the GWINFO message. The TSEARHGW signal is generated automatically at the end of the previously configured timer. Once the node returns to the initial state, it can remain in this state, waiting for more ADVERTISE messages. Because the client node should not respond to the arrival of ADVERTISE messages, it only issued a SEARCHGW message during this test, similar to what was indicated in the simulation. The input and output sequences are shown below and are depicted in Fig. 87.

Input sequence:

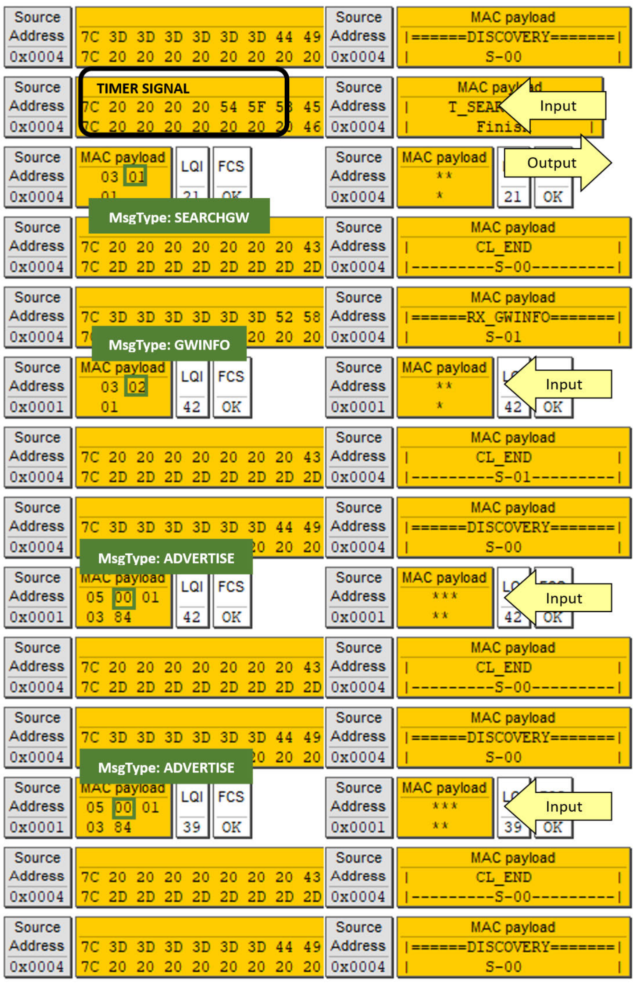
```
TSEARCHGW, GWINFO, ADVERTISE, ADVERTISE
```

Output sequence:

```
SEARCHGW
```

2) TESTING THE PROCEDURE FOR CLIENT CONNECTION CONFIGURATION

Figure 88 shows the MQTT-SN messages used to test the client connection configuration procedure. The analysis was performed from both the gateway and the client side.



```
Source Address 0x0004 | 7C 3D 3D 3D 3D 3D 3D 44 49 | Source Address 0x0004 | MAC payload
                      | 7C 20 20 20 20 20 20 20 20 |                       | |======DISCOVERY=======|
                      |                            |                       | |        S-00          |
Source Address 0x0004 | 7C 20 20 20 20 54 5F 53 45 | Source Address 0x0004 | MAC payload
                      | 7C 20 20 20 20 20 20 20 46 |                       | |    T_SEAR            |
                      |                            |                       | |         Finis        |
Source Address 0x0004 | MAC payload 03 01 01 | LQI 21 | FCS OK | Source Address 0x0004 | MAC payload ** * | 21 | OK
Source Address 0x0004 | 7C 20 20 20 20 20 20 20 43 | Source Address 0x0004 | MAC payload
                      | 7C 2D 2D 2D 2D 2D 2D 2D 2D |                       | |        CL_END        |
                      |                            |                       | |---------S-00---------|
Source Address 0x0004 | 7C 3D 3D 3D 3D 3D 3D 52 58 | Source Address 0x0004 | MAC payload
                      | 7     20 20 20             |                       | |=====RX_GWINFO=======|
                      |                            |                       | |        S-01          |
Source Address 0x0001 | MAC payload 03 02 01 | LQI 42 | FCS OK | Source Address 0x0001 | MAC payload ** * | 42 | OK
Source Address 0x0004 | 7C 20 20 20 20 20 20 20 43 | Source Address 0x0004 | MAC payload
                      | 7C 2D 2D 2D 2D 2D 2D 2D 2D |                       | |        CL_END        |
                      |                            |                       | |---------S-01---------|
Source Address 0x0004 | 7C 3D 3D 3D 3D 3D 3D 44 49 | Source Address 0x0004 | MAC payload
                      |                 20 20 20   |                       | |======DISCOVERY=======|
                      |                            |                       | |        S-00          |
Source Address 0x0001 | MAC payload 05 00 01 03 84 | LQI 42 | FCS OK | Source Address 0x0001 | MAC payload *** ** | 42 | OK
Source Address 0x0004 | 7C 20 20 20 20 20 20 20 43 | Source Address 0x0004 | MAC payload
                      | 7C 2D 2D 2D 2D 2D 2D 2D 2D |                       | |        CL_END        |
                      |                            |                       | |---------S-00---------|
Source Address 0x0004 | 7C 3D 3D 3D 3D 3D 3D 44 49 | Source Address 0x0004 | MAC payload
                      |                 20 20 20   |                       | |======DISCOVERY=======|
                      |                            |                       | |        S-00          |
Source Address 0x0001 | MAC payload 05 00 01 03 84 | LQI 39 | FCS OK | Source Address 0x0001 | MAC payload *** ** | 39 | OK
Source Address 0x0004 | 7C 20 20 20 20 20 20 20 43 | Source Address 0x0004 | MAC payload
                      | 7C 2D 2D 2D 2D 2D 2D 2D 2D |                       | |        CL_END        |
                      |                            |                       | |---------S-00---------|
Source Address 0x0004 | 7C 3D 3D 3D 3D 3D 3D 44 49 | Source Address 0x0004 | MAC payload
                      | 7C 20 20 20 20 20 20 20 20 |                       | |======DISCOVERY=======|
                      |                            |                       | |        S-00          |
```

FIGURE 87. Sniffing (client-side) of the gateway announcement and discovery procedure.

- **Gateway side:** For this procedure, the following input sequence has been selected:

  CONNECT(willFlag=1)_WILLTOPIC_WILLMSG_

  This sequence consists of a connection message sent by the client, which contains the will flag with a value of one. It also includes two messages that assign a topic and a will message.

The simulation shows the state changes necessary for a gateway to respond to a connection request required by a client, in addition to the fact that the latter needs to configure topics and will messages. The simulation receives an input stream that contains only MQTT-SN messages; however, the CONNECT message is specified to contain a will field set to one. This *will* field facilitates the transition from the WAIT CONNECTION state (S0) to the RX_TOPIC state (S1) and subsequently to the RX_MSG state (S2). Finally, the simulation state machine returns to the initial state because it can wait for another connection sequence from this state. In each state transition, a message is issued corresponding to

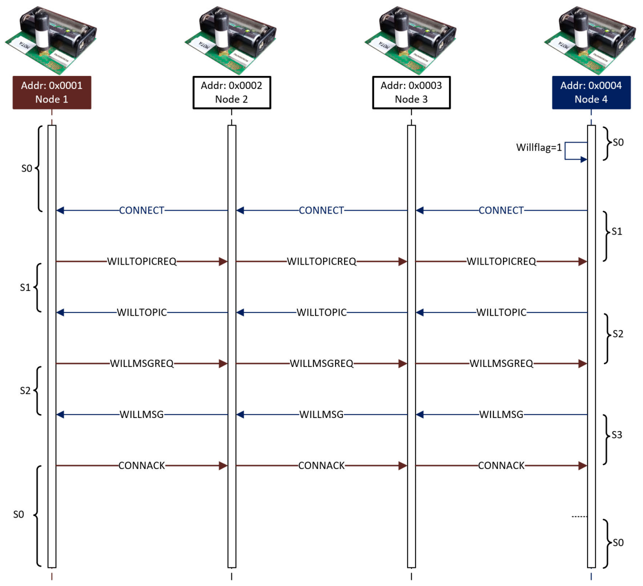
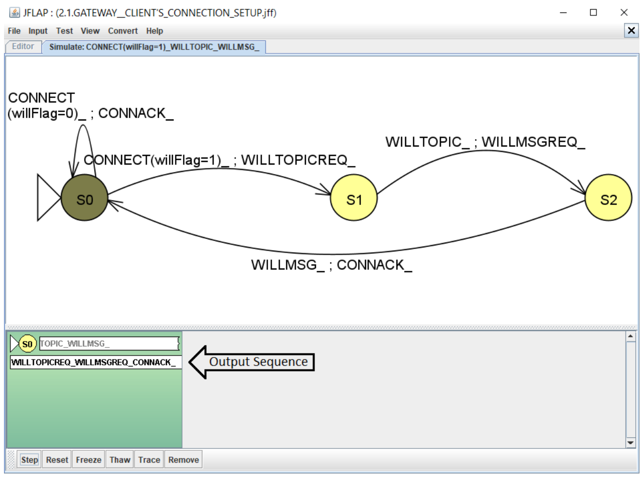


FIGURE 88. MQTT-SN messages used in testing the procedure for client connection configuration.

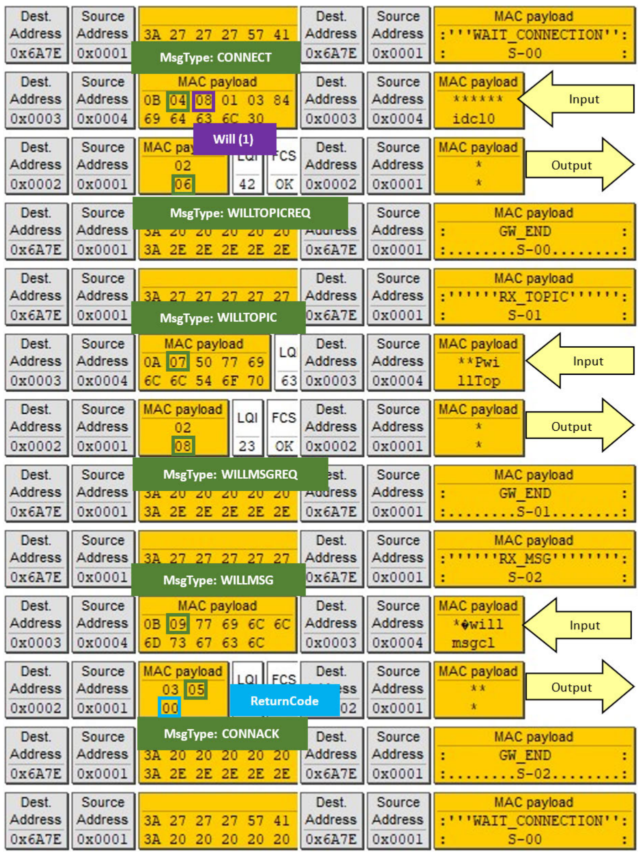


JFLAP : (2.1.GATEWAY_CLIENT'S_CONNECTION_SETUP.jff)
File Input Test View Convert Help
Editor
Simulate: CONNECT(willFlag=1)_WILLTOPIC_WILLMSG_
CONNECT
(willFlag=0)_ ; CONNACK_
CONNECT(willFlag=1)_ ; WILLTOPICREQ_
WILLTOPIC_ ; WILLMSGREQ_
WILLMSG_ ; CONNACK_
S0
S1
S2
S0
TOPIC_WILLMSG_
WILLTOPICREQ_WILLMSGREQ_CONNACK_
Output Sequence
Step
Reset
Freeze
Thaw
Trace
Remove

FIGURE 89. Simulation (gateway-side) of the procedure for client connection configuration.

the output sequence, as shown in Fig. 89. If the simulation returns to the initial state, it can be considered that a successful connection has been established.

Using the sniffer, it can be observed that the node behaves consistently with the simulation if it receives the same sequence. The node starts in the WAIT CONNECTION state (S0) and, upon receiving a CONNECT message with the will flag set to one (will field $=0b00001000$ or 0×08), enters the RX_TOPIC state (S1) after transmitting the message WILLTOPICREQ. Subsequently, the node enters the RX_MSG state (S2) upon receiving the WILLTOPIC message while issuing a WILLMSGREQ message. Finally, the node returns to the initial state after accepting the connection using the CONNACK message with the field returnCode$=0 \times 00$. Upon reaching this state, it is ready to



```
Dest.   Source  |                    | Dest.   Source  | MAC payload
0x6A7E  0x0001  | 3A 27 27 27 57 41  | 0x6A7E  0x0001  | :'''WAIT_CONNECTION'': 
                |                    |                 | :        S-00        :
0x0003  0x0004  | MAC payload        | 0x0003  0x0004  | ******
                | 0B 04 08 01 03 84  |                 | idcl0
                | 69 64 63 6C 30     |                 |
0x0002  0x0001  | MAC payload 02 06  | LQI 42  FCS OK  | 0x0002 0x0001 | * *
0x6A7E  0x0001  | 3A 20 20 20 20 20  | 0x6A7E  0x0001  | :       GW_END       :
                | 3A 2E 2E 2E 2E 2E  |                 | :........S-00........:
0x6A7E  0x0001  | 3A 27 27 27 27 27  | 0x6A7E  0x0001  | :''''''RX_TOPIC'''''':
                |                    |                 | :        S-01        :
0x0003  0x0004  | MAC payload        | LQ 63           | 0x0003 0x0004 | **Pwi
                | 0A 07 50 77 69     |                 |               | llTop
                | 6C 6C 54 6F 70     |                 |
0x0002  0x0001  | MAC payload 02 08  | LQI 23  FCS OK  | 0x0002 0x0001 | * *
0x6A7E  0x0001  | 3A 20 20 20 20 20  | 0x6A7E  0x0001  | :       GW_END       :
                | 3A 2E 2E 2E 2E 2E  |                 | :........S-01........:
0x6A7E  0x0001  | 3A 27 27 27 27 27  | 0x6A7E  0x0001  | :''''''RX_MSG'''''''':
                |                    |                 | :        S-02        :
0x0003  0x0004  | MAC payload        | 0x0003  0x0004  | *�will
                | 0B 09 77 69 6C 6C  |                 | msgcl
                | 6D 73 67 63 6C     |                 |
0x0002  0x0001  | MAC payload 03 05  | LQI FCS 02      | 0x0002 0x0001 | ** *
                | 00                 |                 |
0x6A7E  0x0001  | 3A 20 20 20 20 20  | 0x6A7E  0x0001  | :       GW_END       :
                | 3A 2E 2E 2E 2E 2E  |                 | :........S-02........:
0x6A7E  0x0001  | 3A 27 27 27 57 41  | 0x6A7E  0x0001  | :'''WAIT_CONNECTION'':
                | 3A 20 20 20 20 20  |                 | :        S-00        :
```

FIGURE 90. Sniffing (gateway-side) of the procedure for client connection configuration.

wait for another sequence of messages sent by the client. The input and output sequences are presented below and are depicted in Fig. 90.

Input sequence:

```
CONNECT(willFlag=1), WILLTOPIC, WILLMSG
```

Output sequence:

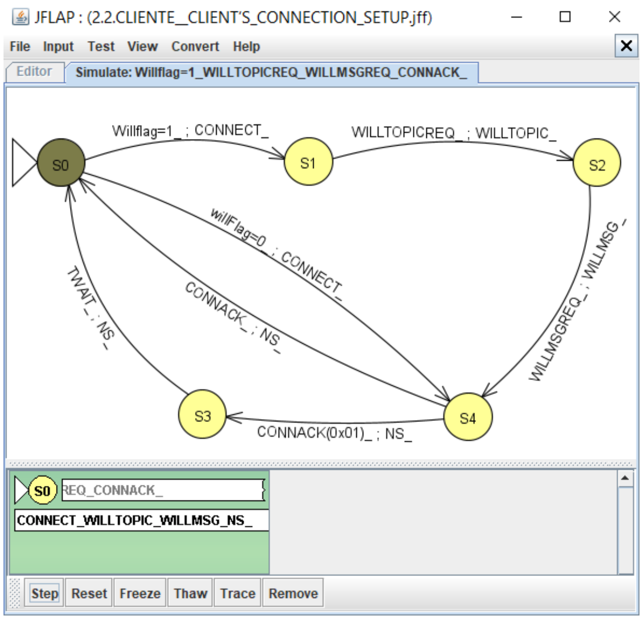
```
WILLTOPICREQ, WILLMSGREQ, CONNACK
```

- **Client side:** For this procedure, the following input sequence has been selected:

  `Willflag=1\_WILLTOPICREQ \_WILLMSGREQ\_CONNACK\_`

  This sequence consists of a signal that allows a connection message to be sent with a will flag with a value of one. It also consists of two messages: the gateway, which requests a topic, and the last will message, which accepts or rejects the connection.

The simulation shows a client's different state transitions when connecting with a gateway. The message sequence contains a signal, which indicates the value of the will flag before sending the CONNECT message. Depending on the value of the flag, a state change will occur. In this case, the sequence indicates that the will flag was set to one, so a state change will occur from the initial state S0 to the RX_TOPIC_REQ state (S1) after sending a connection message. Having reached state S1, it must go to state RX_MSG_REQ (S2) when a WILLTOPICREQ message is received, and the WILLTOPIC message is transmitted. To go from state S2 to state RX_CONNACK (S3), the node must receive a WILLMSGREQ message and respond with WILLMSG. Finally, the state machine returns to the initial state from state S3 once the CONNACK message is received. From the initial state, another connection sequence can be received. Fig. 91 shows the change of states and the output sequence.



JFLAP : (2.2.CLIENTE_CLIENT'S_CONNECTION_SETUP.jff)
File Input Test View Convert Help
Editor
Simulate: Willflag=1_WILLTOPICREQ_WILLMSGREQ_CONNACK_
S0
S1
S2
S3
S4
Willflag=1_ ; CONNECT_
WILLTOPICREQ_ ; WILLTOPIC_
WILLMSGREQ_ ; WILLMSG_
willFlag=0_ ; CONNECT_
CONNACK_ ; NS_
TWAIT_ ; NS_
CONNACK(0x01)_ ; NS_
Step
Reset
Freeze
Thaw
Trace
Remove

```
S0  REQ_CONNACK_
CONNECT_WILLTOPIC_WILLMSG_NS_
```

FIGURE 91. Simulation (client-side) of the procedure for client connection configuration.

The sniffing process shows that the node starts its operation in the ESTABLISH CONNECTION state (S0) and then goes to the RX_TOPIC_REQ(S1) state after sending a CONNECT message with the will flag set to one (field will= 0b00001000 or 0 × 08). The signal that indicates the flag's value is recreated using the node's push button, and the node also indicates that the signal has been received. This way, the node knows which state to advance to. According to the sequence received, the node goes from state S1 to state RX_MSG_REQ(S2) until reaching state RX_CONNACK(S3), where it waits for the CONNACK message with the field returnCode= 0 × 00. The node returns to the initial state, where it can restart the connection process. If the message received in state S3 is a CONNACK with a returnCode= 0 × 01 field, the node will go to the DECONGESTION state (S4), where it will activate a timer and wait for it to end to return to the initial state and restart the procedure. The latter state is not covered in this test. The entry and exit sequences are shown below.

Sequence of signals and messages received:

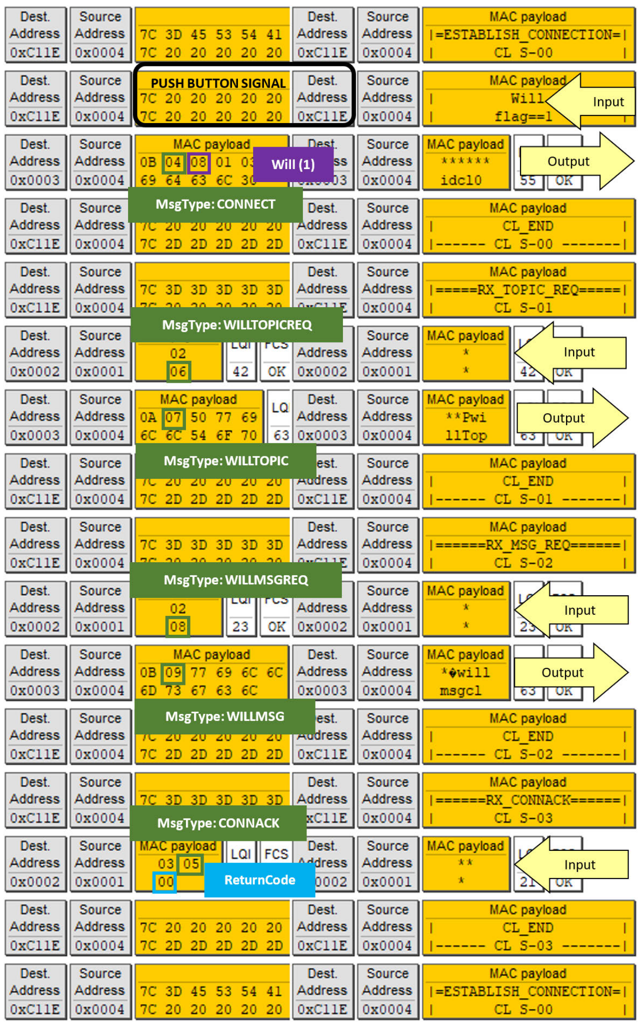


FIGURE 92. Sniffing (client-side) of the procedure for client connection configuration.

```
Willflag=1, WILLTOPICREQ, WILLMSGREQ,
CONNACK
```

Output sequence:

```
CONNECT, WILLTOPIC, WILLMSG
```

Figure 92 allows verifying the input and output sequences and the will flags and returnCode fields used to change states.

VII. CONCLUSION

This paper details the development of FSMs so that the MQTT-SN protocol can operate over IEEE 802.15.4 in linear topologies. The Mealy machine is explicitly used for this purpose because it allows a better representation of the operation of each procedure of the MQTT-SN protocol. This type of FSM considers the reception and transmission of MQTT-SN messages in each state transition. In addition, the number of states that would be used with another type of state machine is considerably reduced.

Representing an MQTT-SN procedure through a Mealy machine is comparable to using SDL. However, the latter allows the representation of the operation of variables, comparators, timers, and additional signals necessary for coding the nodes. Furthermore, the simplicity of MQTT-SN messages allows their direct encapsulation in an IEEE802.15.4 frame across the node (for example, the RCB256RFR2). This enables their subsequent use as inputs and outputs of the client node and the gateway. The simulations and captures confirmed that the WSNs can respond to the various message sequences generated for each MQTT-SN procedure.

REFERENCES

[1] A. Al-Fuqaha, M. Guizani, M. Mohammadi, M. Aledhari, and M. Ayyash, "Internet of Things: A survey on enabling technologies, protocols, and applications," *IEEE Commun. Surveys Tuts.*, vol. 17, no. 4, pp. 2347–2376, 4th Quart., 2015.

[2] U. Hunkeler, H. L. Truong, and A. Stanford-Clark, "Mqtt-s—A publish/subscribe protocol for wireless sensor networks," in *Proc. 3rd Int. Conf. Commun. Syst. Software Middleware and Workshops (COMSWARE)*, 2008, pp. 791–798.

[3] C. Egas Acosta, F. Gil-Castiñeira, and E. Costa-Montenegro, "Red inalámbrica de sensores con topología lineal sin capa de red," *Revista de Investigación en Tecnologías de la Información*, vol. 9, no. 17, pp. 56–65, Jan. 2021. [Online]. Available: https://riti.es/index.php/riti/article/view/73

[4] F. Belina and D. Hogrefe, "The CCITT-specification and description language SDL," *Comput. Netw. ISDN Syst.*, vol. 16, no. 4, pp. 311–341, Mar. 1989.

[5] C. Egas Acosta, L. Criollo, C. Tipantuña, and J. Carvajal-Rodriguez, "Software-defined networking-enabled efficient default route configuration in IEEE 802.15.4 protocol: A smart algorithmic approach," *Electronics*, vol. 13, no. 8, p. 1537, Apr. 2024. [Online]. Available: https://www.mdpi.com/2079-9292/13/8/1537

[6] I. Jawhar, N. Mohamed, and D. P. Agrawal, "Linear wireless sensor networks: Classification and applications," *J. Netw. Comput. Appl.*, vol. 34, no. 5, pp. 1671–1682, Sep. 2011. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S1084804511001081

[7] E. Ndoye, F. Jacquet, M. Misson, and I. Niang, "Evaluation of RTS/CTS with unslotted CSMA/CA algorithm in linear sensor networks," NICST, France, Tech. Rep. 1, 2013.

[8] C. E. Acosta., F. Gil-Castiñeira, E. Costa-Montenegro, and J. S. Silva, "Reliable link level routing algorithm in pipeline monitoring using implicit acknowledgements," *Sensors*, vol. 21, no. 3, p. 968, Feb. 2021. [Online]. Available: https://www.mdpi.com/1424-8220/21/3/968

[9] L. Vera Sánchez and C. Egas Acosta, "Algoritmo para el monitoreo de estructuras lineales a gran escala," *Revista Tecnológica ESPOL*, vol. 34, no. 3, pp. 58–71, Nov. 2022.

[10] C. E. Acosta, F. Gil-Castiñeira, and C. E. Gualotuña, "Optimization of delays and power consumption in large-scale linear networks using iACK," in *Proc. IEEE ANDESCON*, Oct. 2020, pp. 1–5.

[11] A. C. Egas, F. Gil-Castiñeira, E. Costa-Montenegro, and J. S. Silva, "Automatic allocation of identifiers in linear wireless sensor networks using link-level processes," in *Proc. 8th IEEE Latin-American Conf. Commun. (LATINCOM)*, Nov. 2016, pp. 1–6.

[12] A. Stanford-Clark and H. L. Truong. (Nov. 2013). *MQTT For Sensor Networks (MQTT-SN) Protocol Specification*. [Online]. Available: https://www.oasisopen.org/committees/download.php/66091/MQTT-SN_spec_v1.2.pdf

[13] P. Lea, *Internet of Things for Architects: Architecting IoT Solutions by Implementing Sensors, Communication Infrastructure, Edge Computing, Analytics, and Security*. Birmingham, Packt, 2018.

[14] P. Egli, "MQTT—Message Queueing Telemetry Transport Introduction to MQTT, a protocol for M2M and IoT applications," in *Proc. Enero*, 2017, pp. 1–33. [Online]. Available: https://www.researchgate.net/publication/320126053_MQTT_-_Message_Queueing_Telemetry_Transport_Introduction_to_MQTT_a_protocol_for_M2M_and_IoT_applications

[15] D. Brand and P. Zafiropulo, "On communicating finite-state machines," *J. ACM*, vol. 30, no. 2, pp. 323–342, Apr. 1983, doi: 10.1145/322374.322380.

[16] W. Grieskamp, Y. Gurevich, W. Schulte, and M. Veanes, ''Generating finite state machines from abstract state machines,'' *ACM SIGSOFT Softw. Eng. Notes*, vol. 27, no. 4, pp. 112–122, Jul. 2002, doi: 10.1145/566171.566190.

[17] M. Veanes, P. Hooimeijer, B. Livshits, D. Molnar, and N. Bjorner, ''Symbolic finite state transducers: Algorithms and applications,'' *ACM SIGPLAN Notices*, vol. 47, no. 1, pp. 137–150, Jan. 2012, doi: 10.1145/2103621.2103674.

[18] D. Lee and M. Yannakakis, ''Principles and methods of testing finite state machines–A survey,'' *Proc. IEEE*, vol. 84, no. 8, pp. 1090–1123, 1996.

[19] M. Kaloper and P. Rudnicki, ''Minimization of finite state machines,'' in *Proc. Mizar User's Assoc.*, 1996, pp. 1–11.

[20] Z. Zhang, C. Xia, J. Fu, and Z. Chen, ''Initial-state observability of mealy-based finite-state machine with nondeterministic output functions,'' *IEEE Trans. Syst., Man, Cybern., Syst.*, vol. 52, no. 10, pp. 6396–6405, Oct. 2022.

[21] A. Rockstrom and R. Saracco, ''SDL-CCITT specification and description language,'' *IEEE Trans. Commun.*, vol. COM-30, no. 6, pp. 1310–1318, Jun. 1982.

[22] B. Hogrefe. (May 2013). *SDL-88 Tutorial*. [Online]. Available: http://www.sdl-forum.org/sdl88tutorial/index.html

[23] S. Ruehrup. (2009). *Network Protocol Design and Evaluation*. [Online]. Available: http://hondo.informatik.uni-freiburg.de/teaching/vorlesung/protocol-design-s09/slides/04-Protocol_Specification_2.pdf

[24] Atmel. *Atmel AVR10004: RCB256RFR2—Hardware User Manual, Application Note ed. Microchip*. Accessed: Jan. 20, 2024. [Online]. Available: https://ww1.microchip.com/downloads/en/AppNotes/Atmel-42081-RCB256RFR2-Hardware-User-ManualApplication-NoteAV R10004.pdf

[25] D. Guha Roy, B. Mahato, D. De, and R. Buyya, ''Application-aware end-to-end delay and message loss estimation in Internet of Things (IoT)—MQTT-SN protocols,'' *Future Gener. Comput. Syst.*, vol. 89, pp. 300–316, Dec. 2018. https://www.sciencedirect.com/science/article/pii/S0167739X17329990

[26] C. T. C. Criollo, L. E. Acosta, and J. Carvajal. (Mar. 2024). *MQTT-SN Finite State Machine Implementation Repository*. [Online]. Available: https://github.com/criolloluis410/FSM_MQTT-SN_Over_IEEE_802.15.4.git

[27] S. H. Rodger and T. W. Finley, *JFLAP: An Interactive Formal Languages and Automata Package*. Sudbury, MA, USA: Jones & Bartlett Learning, 2006.

LUIS CRIOLLO CAJAMARCA received the degree in electronics and information networks engineering from the Escuela Politecnica Nacional, Ecuador, in 2024. His research interests include linear wireless sensor networks, embedded software development, general-purpose software development, the implementation of IoT solutions, and software-defined radio.

CARLOS EGAS ACOSTA received the degree in electronics and telecommunications from the Escuela Politecnica Nacional, in 1986, and the master's degree in computer science from the Universidad Andina Simn Bolivar, in 1996. He is currently the Director of the Internet of All Things Research Group, National Polytechnic School. His research interest includes wireless sensor networks with LoRa technology.

CHRISTIAN TIPANTUÑA received the bachelor's degree in telecommunications engineering from the Escuela Politecnica Nacional, Ecuador, in 2011, the M.Sc. degree in wireless systems and related technologies from the Politecnico di Torino, Turin, Italy, in 2013, and the Ph.D. degree in network engineering from the Universitat Politecnica de Catalunya, Barcelona, Spain, in 2022. He is a member of the Grupo de investigacion en Redes Inalambricas, Escuela Politecnica Nacional. His current research interests include UAV-enabled communications, wireless networks, software-defined radio (SDR), optical networks, and machine learning applied to communications systems and networks.

JORGE CARVAJAL-RODRIGUEZ received the degree in electronics and telecommunication engineering from the Escuela Politecnica Nacional, Ecuador, in 2010, and the M.Sc. degree in information technologies from the Mannheim University of Applied Sciences, Germany, in 2013. His research interests include UAV-enabled communications, wireless networks, and software-defined radio. He is a member of the Grupo de investigacion en Redes Inalambricas, Escuela Politecnica Nacional.

CARLA PARRA received the bachelor's degree in electronics and information networks from the Escuela Politecnica Nacional, Ecuador, in 2018, and the joint M.Sc. degree in business management and project management from the Universitat Politecnica de Catalunya and EAE Business School, Barcelona, Spain, in 2020. She is currently pursuing the Ph.D. degree with the Technology Management Program, Escuela Politecnica Nacional. She was the CTO of Nuevas Comunicaciones Iberia, Barcelona, Spain, a company dedicated to commercializing FTTH products.

• • •