ESCUELA POLITÉCNICA NACIONAL

FACULTAD DE INGENIERÍA ELÉCTRICA Y ELECTRÓNICA

DESARROLLO DE LA MÁQUINA DE ESTADOS FINITOS DEL PROTOCOLO MQTT-SN PARA SU OPERACIÓN SOBRE IEEE 802.15.4 EN TOPOLOGIAS LINEALES

TRABAJO DE TITULACIÓN PREVIO A LA OBTENCIÓN DEL TÍTULO DE INGENIERO EN ELECTRÓNICA Y REDES DE INFORMACIÓN

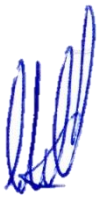
LUIS EDUARDO CRIOLLO CAJAMARCA

DIRECTOR: ING. CARLOS ROBERTO EGAS ACOSTA M.Sc.

Quito, Enero 2023

AVAL

Certifico que el presente trabajo fue desarrollado por Luis Eduardo Criollo Cajamarca, bajo mi supervisión.



ING. CARLOS EGAS, M.Sc.

DIRECTOR DEL TRABAJO DE TITULACIÓN

DECLARACIÓN DE AUTORÍA

Yo, Luis Eduardo Criollo Cajamarca, declaro bajo juramento que el trabajo aquí descrito es de mi autoría; que no ha sido previamente presentado para ningún grado o calificación profesional; y, que he consultado las referencias bibliográficas que se incluyen en este documento.

A través de la presente declaración dejo constancia de que la Escuela Politécnica Nacional podrá hacer uso del presente trabajo según los términos estipulados en la Ley, Reglamentos y Normas vigentes.

LUIS EDUARDO CRIOLLO CAJAMARCA

DEDICATORIA

Este trabajo está dedicada a mis padres Manuel y Magdalena quienes con su amor, paciencia y esfuerzo me han ayudado a cumplir esta importante meta, a mis tías Carmen, Piedad, Delia, y a mi tío Luis que siempre han sido un apoyo fundamental en mi proceso de formación personal y profesional.

A mis abuelitos Andrés, Petrona, Alberto y Mercedes por darme ánimo para seguir adelante, por siempre tener una lección que enseñarme y en general ser un ejemplo para seguir; aunque ya no estén en este plano, todo su amor incondicional seguirá presente en mi vida.

A mis hermanos Diego, Alex y Angelica, por estar conmigo en todo momento y brindarme su apoyo incondicional.

AGRADECIMIENTO

Le agradezco a toda mi familia en especial a mis padres, hermanos, tías, y abuelos. Por estar siempre a mi lado y apoyarme día a día para alcanzar una de las metas más importantes en mi vida.

A todos los amigos que me acompañaron durante toda mi etapa universitaria.

Agradezco también a mi tutor Ing. Carlos Roberto Egas Acosta M.Sc., por brindarme todo su conocimiento, paciencia y apoyo incondicional durante el desarrollo de este proyecto.

ÍNDICE DE CONTENIDO

ÍNDICE DE FIGURAS

ÍNDICE DE TABLAS

RESUMEN

El presente trabajo de titulación tiene como objetivo, desarrollar la máquina de estados finitos(MEF) del protocolo MQTT-SN para su operación sobre IEEE 802.15.4 en topologías lineales. Se logró obtener varias máquinas de estado finito para cada uno de los procedimientos indicados en la especificación MQTT-SN y el código que permite al nodo actuar de acuerdo a las máquinas de estados finitos. Se concluyó que es posible implementar el protocolo MQTT-SN sobre los nodos RCB256RFR2.

En el primer capítulo se describe el funcionamiento del estándar IEEE 802.15.4, el protocolo MQTT, el protocolo MQTT-SN y los diferentes tipos de máquinas de estados finitos. En el segundo capítulo se detalla el desarrollo de las máquinas de estados finitos y su posterior codificación en los nodos RCB256RFR2. También, se ingresa las máquinas de estado finitos al simulador JFLAP. Además, se desarrolla el código necesario para que los nodos tengan la capacidad de generar mensajes MQTT-SN y enviarlos a través de una red de sensores inalámbricos en topología lineal.

En el tercer capítulo se evalúa el código ingresado en los nodos RCB256RFR2, realizando una comparación de las secuencias de mensajes MQTT-SN recibidas y transmitidas y secuencias utilizadas en las simulaciones.

PALABRAS CLAVE: MQTT, MQTT-SN, IEEE 802.15.4, Máquina de estados finitos, Topología lineal y red inalámbrica de sensores.

ABSTRACT

The aim of the present work is to develop the finite state machine of the MQTT-SN protocol for its operation over IEEE 802.15.4 in linear topologies. Several finite state machines were obtained for each of the procedures indicated in the MQTT-SN specification and the code that allows the node to act in disagreement with the finite state machines. It was concluded that it is possible to implement the MQTT-SN protocol on RCB256RFR2 nodes.

The first chapter describes the operation of the IEEE 802.15.4 standard, the MQTT protocol, the MQTT-SN protocol and the different types of finite state machines. The second chapter details the development of the finite state machines and their subsequent coding in the RCB256RFR2 nodes. Also, the finite state machines are entered into the JFLAP simulator. In addition, the necessary code is developed for the nodes to be able to generate MQTT-SN messages and send them through a wireless sensor network in linear topology.

The third chapter evaluates the code entered in the RCB256RFR2 nodes, making a comparison of the MQTT-SN message sequences received and transmitted and the sequences used in the simulations.

KEYWORDS: MQTT, MQTT-SN, IEEE 802.15.4, Finite State Machine, Linear Topology and wireless sensor network.

1 INTRODUCCIÓN

En la actualidad el uso de redes inalámbricas de sensores (WSN-Wireless Sensor Network) ha tenido un incremento en su utilización, y con ello ha aparecido la necesidad de desarrollar tecnologías que requieren el uso del protocolo Message Queing Telemetry Transport for Sensor Networks (MQTT-SN) operando sobre la capa de enlace en topologías lineales conformadas por cientos de nodos [1]. Sin embargo, en la actualidad no se ha encontrado alguna implementación del protocolo MQTT-SN directamente sobre IEEE 802.15.4.

El protocolo MQTT-SN ha sido implementado en sistemas operativos tales como TinyOS [2] y Contiki [3] para operar sobre la capa de red, con nodos sensores de baja capacidad de procesamiento, con baterías y que funcionen en diferentes topologías. No obstante, para lograr que el protocolo MQTT-SN se aplique en redes inalámbricas de sensores con estructuras lineales a gran escala es necesario minimizar los procesos que se realizan en el nodo [1]. La operación del protocolo MQTT-SN directamente sobre el nivel de enlace en topologías lineales puede reducir el procesamiento de los dispositivos que conforman una WSN.

El desarrollo de la MEF(Maquina de Estados Finitos) de MQTT-SN operando sobre IEEE 802.15.4, permitirá conocer si es posible usar MQTT-SN sobre el nivel de enlace, en redes inalámbricas de sensores con bajos costos de implementación, operación y mantenimiento; considerando las características del nodo sensor RCB256RFR2 [4] y las características de una topología lineal donde las funciones de enrutamiento son mínimas [1].

El presente trabajo tiene como objetivo desarrollar la máquina de estados finitos del protocolo MQTT-SN para su operación sobre IEEE 802.15.4 en topologías lineales, para su posterior representación mediante el lenguaje de especificación y descripción (SDL-Specification and Description Language). Con lo cual se podrá codificar el nodo RCB256RFR2 de tal forma que pueda responder a diferentes secuencias de tramas 802.15.4 que contengan mensajes MQTT-SN, lo que permitirá conocer si el nodo RCB256RFR2 es capaz de trabajar con el protocolo MQTT-SN.

1.1 OBJETIVOS

El objetivo general de este Proyecto Técnico es: Desarrollar la máquina de estados finitos del protocolo MQTT-SN para su operación sobre IEEE 802.15.4 en topologías lineales.

Los objetivos específicos del Proyecto Técnico son:

- Analizar la base teórica para el desarrollo del proyecto.
- Definir los requerimientos para el desarrollo de la máquina de estados finitos del protocolo MQTT-SN sobre IEEE 802.15.4.
- Desarrollar la máquina de estados finitos del protocolo MQTT-SN sobre IEEE 802.15.4.
- Evaluar la codificación del pseudocódigo de la MEF en los nodos RCB256RF2.

1.2 ALCANCE

El presente trabajo de titulación pretende desarrollar la máquina de estados finitos del protocolo MQTT-SN para un nodo cliente y un nodo Gateway, tal como se observa en la Figura 1.1, y mantener una relación entre ellos a partir de los mensajes utilizados en el protocolo MQTT-SN los cuales serán transmitidos entre los nodos.

Figura 1.1 Cliente y gateway MQTT-SN.

Se estudiará el funcionamiento en detalle del protocolo MQTT-SN, el cual es una tecnología de transporte de datos importante para el desarrollo de aplicaciones de IoT con WSN generalmente implementado para operar sobre protocolos de red tales como 6LowPan y ZigBee [5].

Se desarrollará la máquina de estados finitos del protocolo MQTT-SN, tomando en cuenta las características del nodo RCB256RFR2, ya que de estas características del nodo sensor depende la implementación óptima de la Máquina de Estados Finitos (MEF). Esto se debe a que en el nodo RCB256RFR2, el conjunto de operaciones que se puede realizar en este tipo de nodo es mucho más restringido que un lenguaje de programación general. También se tomará en cuenta el funcionamiento de los nodos sobre una topología lineal.

Utilizando el Lenguaje de Especificación y Descripción (SDL) indicado en la recomendación ITU-T Z.100 [6] se obtendrá un diagrama de la MEF. SDL provee de herramientas que permiten especificar y describir sistemas de telecomunicaciones, servicios [7] y protocolos [8] utilizando máquina de estados finitos. El diseño de la máquina de estados finitos tanto para el gateway como para el cliente considera que el protocolo MQTT-SN será encapsulado sobre IEEE 802.15.4. Por lo tanto, se debe tener claro el funcionamiento del estándar IEEE 802.15.4, ya que de las características de comunicaciones de este protocolo dependerá la comunicación entre el protocolo MQTT-SN y, el sensor. Además, se debe tener en cuenta el funcionamiento del broker porque de este depende la operación del gateway y por consiguiente afectará al desarrollo de la MEF, a pesar de que el broker no estará presente en la MEF o en la codificación del pseudocódigo.

Estudiando cada procedimiento del protocolo MQTT-SN, primero se desarrollará la máquina de estados finitos, para luego, a partir de la MEF, elaborar el pseudocódigo del funcionamiento del protocolo MQTT-SN operando sobre IEEE 802.15.4. El pseudocódigo debe representar el funcionamiento de la máquina de estados finitos del protocolo MQTT-SN, de tal forma que éste se adapte al hardware del nodo RCB256RFR2, el cual, a diferencia de otras plataformas de hardware como el Raspberry y Arduino, se caracteriza porque los programas son de un solo hilo. Para la escritura del pseudocódigo se utilizarán sentencias del lenguaje de programación C. La MEF también será ingresada al simulador de máquina de estados finitos JFLAP [9] para comprobar su funcionamiento.

El pseudocódigo será codificado en el nodo cliente y nodo gateway de manera que se pueda verificar el cambio de estados ante la recepción de mensajes MQTT-SN que serán almacenados en cada nodo. Cada estado estará representado por una función que solo tendrá la capacidad de recibir y enviar mensajes. Cabe recalcar que no se implementarán los procesos a ser desarrollados en cada estado definido en la máquina de estados finitos obtenida, la codificación a realizar en los nodos es solo del pseudocódigo.

Adicionalmente, se debe mencionar que el trabajo de titulación contará con un producto final demostrable, tal como se observa en la Figura 1.2, que corresponde a un prototipo de dos nodos sensores en los cuales se realizará la codificación del pseudocódigo, utilizando el nodo RCB256RFR2. En el intercambio de mensajes entre los nodos que actuarán como cliente y gateway, además se asume que cada mensaje pasa por diferentes nodos de una topología lineal antes de llegar a su destino.

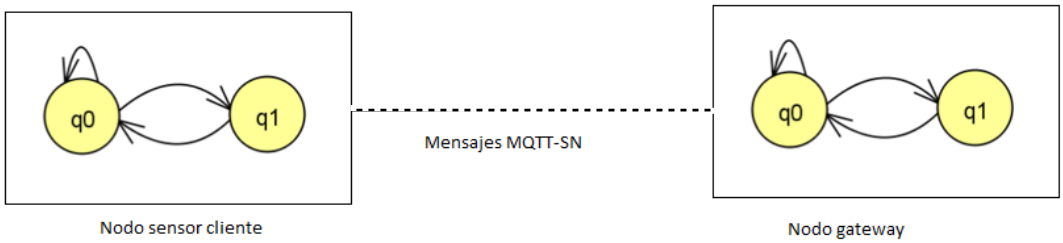


Figura 1.2 Interacción entre nodo cliente y nodo gateway.

Finalmente, se evaluará el pseudocódigo ingresado en los nodos revisando que los cambios de estado sean iguales a los indicados en el diagrama de máquina de estados finitos.

1.3 MARCO TEÓRICO

1.3.1 REDES INALAMBRICAS DE SENSORES

Las Wireless Sensor Networks (WSN) o redes inalámbricas de sensores son redes autoconfiguradas, sin infraestructura que monitorean condiciones físicas o ambientales, como temperatura, sonido, vibración, presión, movimiento o contaminantes, y pasan sus datos a través de la red a una ubicación central o estación base donde los datos pueden almacenarse, observarse, y analizarse. Una estación base sirve como enlace entre los usuarios y la red [10].

Una red de sensores inalámbricos normalmente consta de cientos de miles de nodos sensores, mediante transmisiones de radio los nodos sensores se comunican entre sí. Todos los dispositivos son unidades autónomas que constan de un microcontrolador, una fuente de energía (casi siempre una batería), un radio transceptor y un elemento sensor, tal y como se observa en la Figura 1.3 [10].

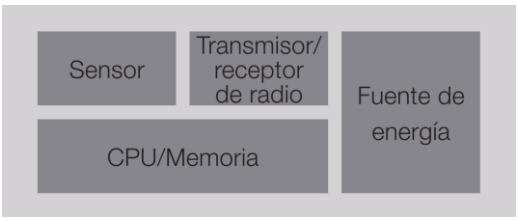


Figura 1.3 Dispositivo autónomo de una red de sensores inalámbricos[10].

Los nodos individuales de una WSN tienen velocidad de procesamiento, capacidad de almacenamiento y ancho de banda de comunicación restringidos. Una vez que se han instalado los nodos sensores, primero deben autoorganizarse para luego recopilar datos relevantes. Los dispositivos también pueden responder a comandos que permiten realizar

tareas específicas o recopilar datos. Además, cada nodo puede funcionar de forma continua o en función de eventos [10].

1.3.2 REDES INALÁMBRICAS DE SENSORES LINEALES

Las redes inalámbricas de sensores lineales son un tipo de WSN en el que los nodos están organizados en línea recta (Figura 1.4), o los nodos están ubicados entre dos líneas paralelas que se extienden por una distancia considerable en comparación con su rango de transmisión. El objetivo de esta topología es reducir los costos de instalación y mantenimiento, mejorar la estabilidad de la red y la tolerancia a fallas, extender la vida útil de la batería del sensor y disminuir la latencia de la comunicación de extremo a extremo para mejorar la calidad de servicio (QoS) de los datos confidenciales [10].

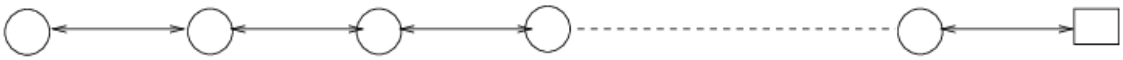


Figura 1.4 Dispositivo autónomo de una red de sensores inalámbricos [11].

La alineación lineal de los nodos sensores se puede emplear en una variedad de aplicaciones, incluido el monitoreo y la vigilancia de fronteras internacionales para cruces ilegales u operaciones de contrabando, el monitoreo de carreteras u oleoductos que transportan petróleo, gas o agua, el monitoreo ambiental de ríos, entre otros [11].

1.3.3 ESTÁNDAR IEEE 802.15.4

Es un estándar de conectividad inalámbrica, destinado a definir las capas utilizadas en redes inalámbricas con bajas velocidades de transmisión, limitado consumo de energía y bajos costos. Debido a que las capas MAC y PHY son proporcionadas por el estándar, las capas superiores serán especificados por protocolos específicos de nivel superior como Thread, Zigbee, 6LoWPAN, entre otros [12].

1.3.3.1 Arquitectura IEEE 802.15.4

El estándar IEEE 802.15.4 especifica un conjunto de capas, las cuales deben cumplir con el modelo de referencia OSI[1] (Open System Inter*connect*ion). Un dispositivo que maneje IEEE 802.15.4 consta de una capa física (PHY physical layer) y una subcapa de control de acceso al medio (MAC Media Access Control). La capa PHY incluye un transceptor de radiofrecuencia mientras que la subcapa MAC facilita el acceso al canal físico.

[1] OSI: Modelo de referencia planteado por la Organización Internacional para la Estandarización para protocolos de red con arquitectura en capas. [13]

En la Figura 1.5 Arquitectura IEEE 802.15.4 [4] se muestran las subcapas que se adhieren al estándar IEEE 802.15.4, las cuales pueden brindar servicio a las capas superiores a través de la subcapa de control lógico de enlace (LLC Logical Link Control)[13].

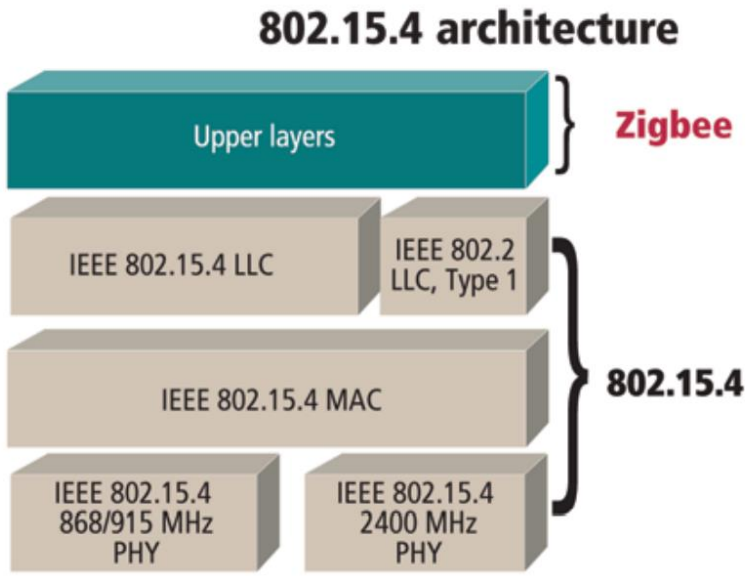


Figura 1.5 Arquitectura IEEE 802.15.4 [4].

1.3.3.2 Capa Física

La capa PHY es responsable de interactuar con el medio de transmisión físico, intercambiando información con él y con la capa superior (capa MAC). Sus responsabilidades en relación con el medio físico radioeléctrico son las siguientes [4]:

- Activación y desactivación del transceptor de radio.
- Detección de energía (ED- Energy detection).
- Indicación de calidad de enlace (LQI-Link Quality Indication).
- Evaluación de canal transparente (CCA- Clear Channel Assessment).

Antes de describir las características indicadas hay que mencionar las bandas de frecuencia especificadas por la capa física de IEEE 802.15.4.

1.3.3.2.1 Canales definidos en el estándar IEEE 802.15.4

La capa física de IEEE 802.15.4 especifica tres bandas de frecuencia operativas de 2,4 GHz, 915 MHz y 868 MHz. Entre 868 y 868,6 MHz se encuentra un solo canal, 10 canales entre 902 y 928 MHz y 16 canales entre 2,4 y 2,4835 GHz como se muestra en la Figura 1.6. [14]

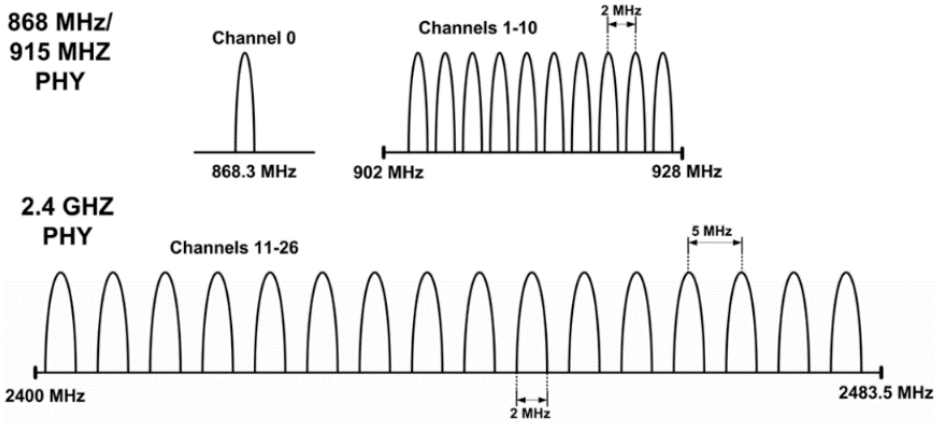


Figura 1.6 Bandas de frecuencia operativas [14].

A 2,4 GHz, la velocidad de datos es de 250 kbps; a 915 MHz, son 40 kbps; y a 868 MHz, son 20 kbps. Debido a la disminución de las pérdidas de propagación, las frecuencias más bajas son mejores para rangos de transmisión más largos. Las transmisiones de baja velocidad tienen un área de cobertura mayor y una sensibilidad mejorada. Cuanto mayor sea la tasa, mayor será el rendimiento, menor la latencia y menor el ciclo de trabajo. Todas las bandas de frecuencia utilizan la técnica de espectro ensanchado por secuencia directa (DSSS-Direct Sequence Spread Spectrum) para la transmisión [14]. En la Tabla 1.1 se resumen las características de cada banda de frecuencia.

Tabla 1.1 Características de las bandas de frecuencia utilizadas por 802.15.4 [14].

Banda de Frecuencia	N° de Canales	Técnica de Ensanchamiento	Modulación	Velocidad de símbolo por canal (kbaud)	Tasa de bits por canal (kbps)
868 MHz	1	Binary DSSS	BPSK	20	20
915 MHz	10	Binary DSSS	BPSK	40	40
2.4 GHz	16	16-array DSSS	O-QPSK	62.5	250

1.3.3.2.2 Activación y desactivación del transceptor de radio.

El transceptor de radio se puede cambiar entre tres estados: transmisión, recepción y reposo. La radio se enciende o se apaga en respuesta a las solicitudes de la subcapa MAC. De acuerdo con el estándar, el tiempo de respuesta entre la transmisión y la recepción no debe exceder los 12 períodos de símbolos (cada símbolo corresponde a 4 bits) [14].

1.3.3.2.3 Detección de energía (ED) dentro del canal actual

Es una estimación de la potencia de señal recibida dentro del ancho de banda de un canal IEEE 802.15.4, en dicho canal no realiza identificación o decodificación de señal. El tiempo de detección de energía debe ser de 8 períodos de símbolo, la capa de red a menudo usa

este valor como parte de un algoritmo de selección de canales o para evaluar si un canal está ocupado o inactivo mediante la evaluación de canal transparente (CCA)[14].

1.3.3.2.4 Indicación de calidad de enlace (LQI)

El LQI es una métrica que describe la calidad de un paquete recibido. Esta métrica evalúa la calidad de la señal después de haberla recibido de un enlace. Su valor puede obtenerse mediante la estimación de una señal de ruido, utilizando el valor de ED, o por medio de una combinación de ambos. Los niveles superiores pueden utilizar el valor de LQI[14].

1.3.3.2.5 Evaluación de canal transparente (CCA)

Esta operación se encarga de reportar el estado actual de actividad del medio: ocupado o inactivo. Se puede realizar de tres maneras diferentes [14]:

- **Modo detección de energía.** Solo indica que el canal está ocupado, si la energía observada excede el umbral ED.
- **Modo detección de portadora.** Indica un canal ocupado si detecta una señal con modulación y características de dispersión IEEE 802.15.4.
- **Modo detección de portadora con detección de energía.** Este es un híbrido de los dos métodos anteriores. Solo si CCA identifica una señal con modulación, características de dispersión IEEE 802.15.4 y energía por encima del umbral ED, indica un canal ocupado.

1.3.3.3 Subcapa MAC

La subcapa MAC del protocolo IEEE 802.15.4 sirve como enlace entre la capa física y los protocolos de capas superiores. La subcapa MAC proporciona los siguientes servicios [15]:

- El servicio de datos MAC: Permite transmitir datos de una capa a otra.
- Servicios de administración de MAC: Proporciona medios para que la capa superior aceda a los servicios brindados por la subcapa MAC.

La capa MAC de IEEE 802.15.4 al igual que otros protocolos de red inalámbricos utiliza CSMA/CA (acceso múltiple por detección de portadora y prevención de colisiones) como protocolo de acceso al canal, lo cual permite manejar los modos de operación beacon habilitado y beacon no habilitado.

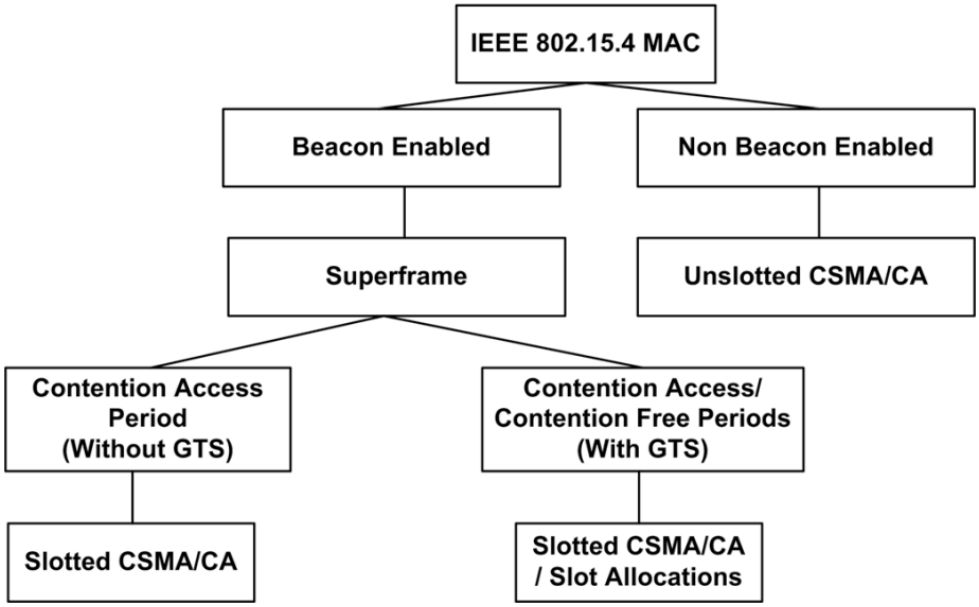


Figura 1.7. Modos de operación 802.15.4 [14].

1.3.3.3.1 Modo beacon habilitado

Cuando la red inalámbrica trabaja con el modo beacon habilitado por obligación utilizará una supertrama para gestionar la comunicación entre dispositivos asociados a la red inalámbrica. El formato de la supertrama es definido por un nodo coordinador y es transmitido entre dos tramas beacon, que son enviadas periódicamente por el Coordinador. La supertrama está divida por 16 ranuras de igual tamaño seguidas por un período inactivo predefinido (Figura 1.8). El intervalo de tiempo total de estos intervalos de tiempo es denominado período de acceso de contención (CAP-Contention Access Period), durante el cual los nodos pueden intentar comunicarse mediante CSMA/CA ranurado [15].

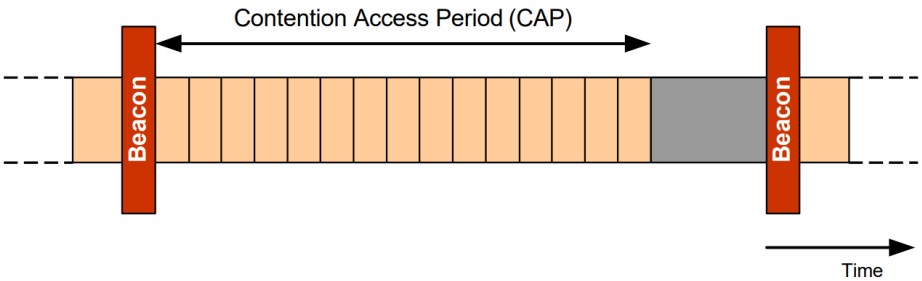


Figura 1.8 Supertrama [15]**.**

Un nodo puede solicitar la asignación de intervalos de tiempo particulares, estos intervalos de tiempo consecutivos son denominados intervalos de tiempo garantizados (GTS-Guaranteed Time Slots), están ubicados después de los intervalos CAP. El intervalo de tiempo total de todos los GTS se denomina período libre de contención (CFP-Contention

Free Period). La comunicación en el CFP no requiere el uso de CSMA/CA. El uso de GTS reduce el CAP, y la supertrama consiste entonces en un CAP seguido de un CFP seguido de un período muerto. Como se muestra en la Figura 1.9 [12].

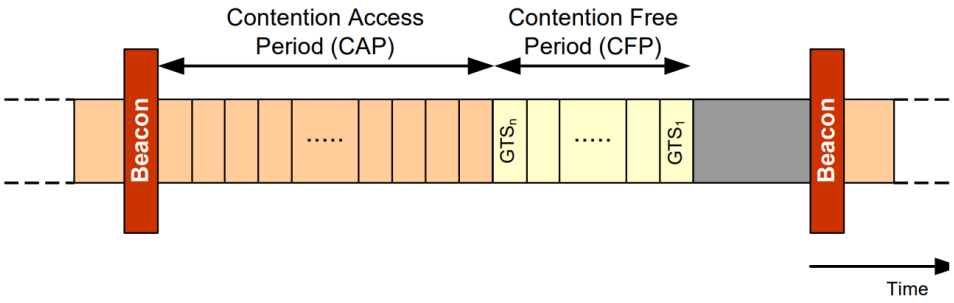


Figura 1.9 Supertrama con periodo de contención libre [15].

Los GTS son apropiados para aplicaciones que requieren una cantidad específica de ancho de banda y una latencia mínima.

1.3.3.3.2 Modo beacon no habilitado

Una red que trabaja en modo beacon no habilitado evita enviar tramas beacon de forma regular, solo lo hará si es necesario asociar un dispositivo con un nodo coordinador. Un dispositivo se comunica con el coordinador solo cuando lo necesita, esto implica una comunicación asincrónica y el ahorro de energía. Para determinar si hay datos pendientes para un nodo, el nodo tendrá que sondear al Coordinador [15].

El modo beacon no habilitado es útil en situaciones en las que solo se espera tráfico ligero entre los nodos de la red y el Coordinador. El uso de tramas beacon regulares puede desgastar energía de los nodos.

1.3.3.3.3 Acceso múltiple por detección de portadora y prevención de colisiones

También conocido como CSMA/CA es un mecanismo de acceso al medio, mediante el cual los nodos realizan un sondeo del canal para verificar que no exista una transmisión en curso. Se inicia la transmisión al instante si el canal se encuentra libre, por el contrario, si el canal está ocupado el nodo espera un intervalo de tiempo aleatorio (periodos de backoff) para reanudar la escucha del canal. Este mecanismo evita varios nodos transmitan al mismo tiempo [4].

1.3.3.3.4 CSMA-CA ranurado

Este mecanismo es empleado en redes inalámbricas que utilizan el modo beacon habilitado. CSMA/CA ranurado hace que los límites de los slots de la supertrama se alineen con los límites de los periodos de backoff de todos los dispositivos de la red inalámbrica,

cuando el coordinador inicia la trama beacon, éste se alinea con el inicio del primer periodo de backoff. Cuando se va a transmitir una trama dentro de una red beacon, el nodo debe ejecutar el algoritmo de backoff antes de transmitir en un slot del CAP de la supertrama, si el canal no se encuentra ocupado inicia la trama en el siguiente slot disponible y si se encuentra ocupado ejecuta el algoritmo de backoff antes de intentar transmitir. Las tramas ACK y beacon se transmiten sin utilizar CSMA-CA. [16]

1.3.3.3.5 CSMA-CA no ranurado

Este mecanismo es empleado, por redes no beacon, por lo cual un dispositivo que utilice CSMA/CA no ranurado pasara a ejecutar directamente el algoritmo de backoff cuando se requiera transmitir tramas. Cuando finalice el intervalo de tiempo aleatorio se iniciará la transmisión si el canal está disponible, de lo contrario el algoritmo de backoff se ejecutará de nuevo. Las tramas ACK se envían sin emplear el mecanismo CSMA/CA[16].

1.3.3.4 Transferencia de datos en 802.15.4

Para la transferencia de datos están definidos tres modelos: transmisión de datos desde un dispositivo hacia un coordinador, transmisión de datos desde el coordinador a un dispositivo y transmisión de datos punto a punto. Los primeros dos modelos de transmisión son ideales para utilizar en topologías estrella que requieren de un nodo coordinador y varios dispositivos RFD. Se puede emplear cualquiera de los tres modelos en topologías punto a punto. Los datos pueden intercambiarse entre cualquier par de dispositivos FFD o RFD. [4]

1.3.3.4.1 Transmisión de datos desde un dispositivo hacia un coordinador

En el momento que un dispositivo requiera transmitir información al coordinador en una red beacon, primero debe escuchar al beacon enviado por el coordinador y después sincronizarse con la supertrama. El dispositivo mediante CSMA/CA ranurado obtiene su turno para transmitir su información al coordinador [4]. El coordinador confirma la recepción mediante un acuse de recibo (ACK) como se muestra en la Figura 1.10.

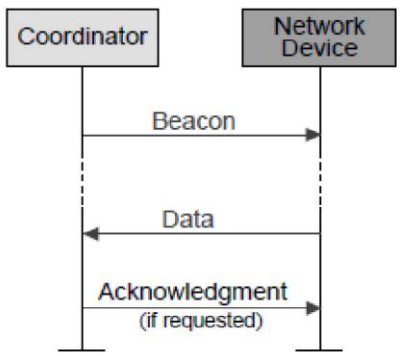


Figura 1.10. Transmisión de un dispositivo al coordinador [4].

Si el dispositivo utiliza el mecanismo CSMA/CA no ranurado para enviar información, solo envía la información y opcionalmente el coordinador puede utilizar ACKs para confirmar la recepción de información.

1.3.3.4.2 Transmisión de datos desde el coordinador a un dispositivo

En una red beacon, si el coordinador requiere enviar datos debe anunciar que tiene datos para enviar. Un dispositivo debe reconocer que el coordinador desea enviarle información, ya que todos los dispositivos de la red censan el canal periódicamente. Un nodo transmitirá un comando Data Request (Figura 1.11), cuando reconoce que el coordinador tiene información por enviarle, indicando al, coordinador que puede transmitir la información pendiente [4]. El coordinador enviara la información utilizando CSMA/CA ranurado. Opcionalmente se puede utilizar ACKs entre los dispositivos involucrados.

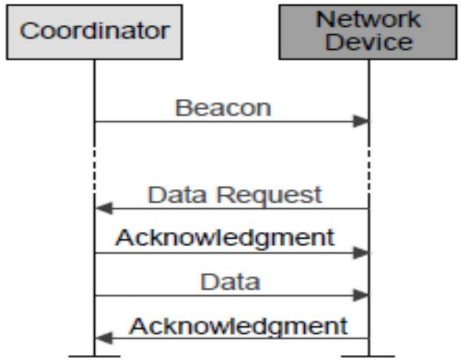


Figura 1.11. Transmisión de coordinador a un dispositivo [4].

Si el coordinador utiliza el mecanismo CSMA/CA no ranurado para enviar información, espera que un dispositivo dentro de la red envié un comando solicitando la transferencia de datos. Además, también se puede utilizar ACKs entre los dispositivos involucrados[4].

1.3.3.5 Tipos de dispositivos de una red IEEE 802.15.4

Se pueden configurar dos tipos de dispositivos en una red IEEE 802.15.4:

1.3.3.5.1 FDD (Dispositivo de función completa)

Este tipo de dispositivo admite todas las características indicadas por el estándar IEEE 802.15.4 y se puede utilizar de las siguientes maneras [16]:

- **Nodo Coordinador:** Es el encargado de coordinar y enlazar todos los nodos de su red.

- **Nodo simple:** Este nodo solo recopila datos de sus sensores y los envía a un nodo coordinador.

1.3.3.5.2 RFD (dispositivo de función reducida)

Un dispositivo de función reducida tipo de dispositivo tiene capacidades reducidas, simplemente realiza funciones básicas como recopilar datos de sensores para después transmitirlos a un nodo coordinador.

1.3.3.6 Estructura de las tramas MAC

La trama MAC IEEE 802.15.4 tiene una longitud máxima de 127 bytes y contiene tres partes básicas: MAC Header, MAC Service Data Unit y MAC Footer. La trama mostrada en la Figura 1.12 es una trama general IEEE 802.15.4, se puede apreciar las partes mencionadas de la trama MAC que a su vez contienen los campos que contienen. Todos los campos son descritos a continuación [16]:

- **MAC Header (MHR).** Está conformado por el campo de control de trama, un número de secuencia e información de direccionamiento.
  - **Control de Trama:** Contiene información que define el tipo de trama y otras banderas de control. Tiene una longitud de 16 bits.
  - **Número de secuencia:** Contiene un identificador de secuencia único para la trama, en tramas ACK es utilizado para determinar el éxito de una comunicación. Tiene una longitud de 8 bits.
  - **Identificador PAN destino:** Contiene el identificador único de red PAN perteneciente al receptor de la trama. Tiene una longitud de 16 bits.
  - **Dirección de destino:** Contiene la dirección del receptor de la trama. Tiene una longitud de 16 o 64 bits.
  - **Identificador PAN origen:** Contiene el identificador único de red PAN del transmisor de la trama, Es un campo con una longitud de 16 bits.
  - **Dirección de origen:** Contiene la dirección del transmisor de la trama. Tiene una longitud de 16 o 64 bits.
- **MAC Service Data Unit (MSDU).** Está conformado por el campo carga útil.

- **Carga útil:** Contiene información específica que depende del tipo de trama IEEE 802.15.4 utilizada. Tiene una longitud variable, pero hay que tener en cuenta el tamaño máximo de la trama MAC.

- **MAC Footer (MFR).** Está conformado solo por el campo Frame Check Sequence (FCS).
  - **FCS.** Contiene una comprobación de redundancia cíclica (CRC) de 16 bits. Tiene una longitud de 16 bits.

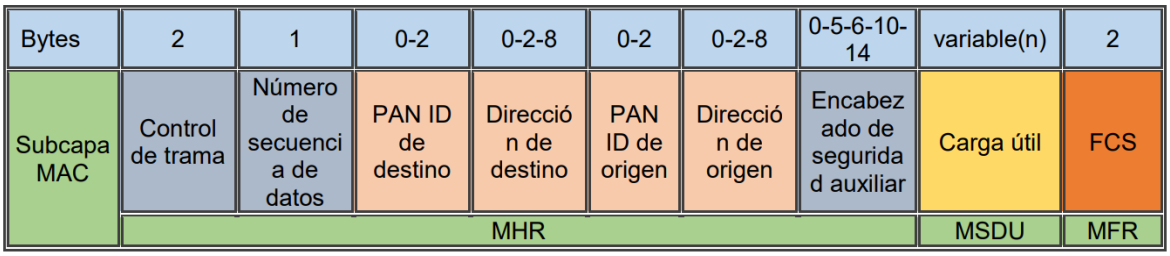


Figura 1.12. Trama general IEEE 802.15.4 [16].

1.3.3.7 Tipos de tramas utilizadas IEEE 802.15.4

El estándar IEEE 802.15.4 especifica 4 tipos de tramas [16]:

1.3.3.7.1 Trama de Datos.

Se utiliza para transportar la información útil del usuario, su estructura se observa en la Figura 1.13.

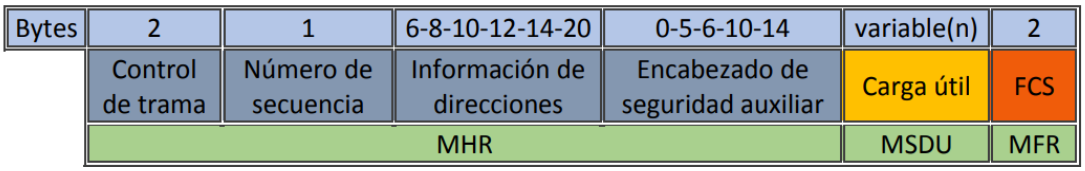


Figura 1.13. Trama de Datos IEEE 802.15.4 [16].

1.3.3.7.2 Trama beacon.

El coordinador de la red utiliza la trama beacon para sincronizar los relojes de todas las entidades de red. La Figura 1.14 muestra su estructura.

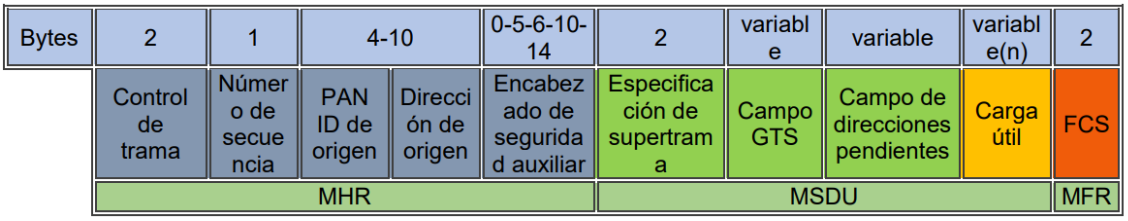


Figura 1.14. Trama beacon IEEE 802.15.4 [16].

La MAC Service Data Unit de este tipo de trama contiene tres nuevos campos, descritos a continuación:

- **Especificación de supertrama:** especifica parámetros afines a la supertrama, como el orden de supertrama y beacon, la ranura CAP final, la vida útil de la batería, el coordinador de red de área personal PAN, el permiso de asociación. El campo tiene una longitud de 16 bits.
- **El campo GTS.** Contiene información de los intervalos de tiempo GTS los cuales se asignan según una lista GTS y varias banderas de control. Este campo tiene una longitud de variable.
- **El campo Dirección Pendiente.** Contiene información de todos los dispositivos que tienen mensajes pendientes del coordinador. Este campo tiene una longitud de variable.

1.3.3.7.3 Trama ACK

La trama ACK tiene el propósito de validar la información que se recibe, esta trama no posee carga útil y su estructura se muestra en la Figura 1.15.

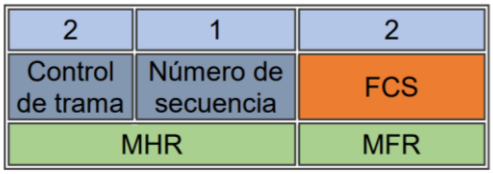


Figura 1.15. Trama ACK IEEE 802.15.4 [16].

1.3.3.7.4 Trama de Comandos

Las tramas de comandos son útiles para ejecutar funciones administrativas en la red, como son el envío de solicitudes, respuestas y notificaciones. La MAC Service Data Unit de este tipo de trama contiene dos campos, el primer campo tiene una longitud de 1 byte e indica el tipo de comando utilizado, el segundo campo es de longitud variable y contiene la

información del comando. La estructura de la trama de comandos se muestra en la Figura 1.16.

Bytes	2	1	6-8-10-12-14-20	0-5-6-10-14	1	variable(n)	2
	Control de trama	Número de secuencia	Información de direcciones	Encabezado de seguridad auxiliar	Tipo de comando	Carga útil	FCS
	MHR				MSDU		MFR

Figura 1.16. Trama de Comandos IEEE 802.15.4 [16].

1.3.4 MÓDULO TRANSCEPTOR RCB256RFR2

Como se muestra en el diagrama de bloques (Figura 1.17) el módulo transceptor está compuesto por el microcontrolador ATmega256RFR2, una memoria ID EEPROM10 (almacena un número de serie y una dirección MAC), un interruptor, diodos LED, una antena, un filtro/balun, un cristal SIWARD SX4025 de 16 MHz de alta precisión, pines de entrada y salida. Además, el transceptor de radio incorpora un hardware para gestión de modulación y demodulación de señales [16].

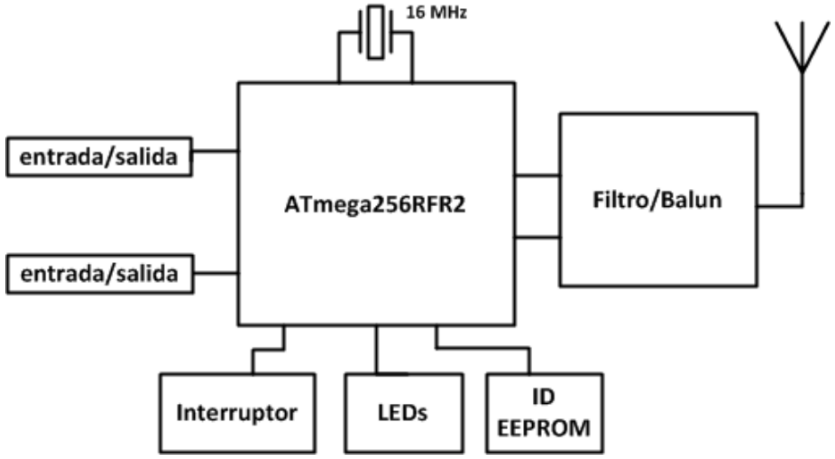


Figura 1.17. Diagrama de bloques del transceptor RCB256RFR2 [16].

1.3.4.1 Microcontrolador ATmega256RFR2

El módulo transceptor funciona con un microcontrolador Atmega256RFR2 el cual permite el desarrollo de aplicaciones IEEE 802.15.4. El microcontrolador es de tipo RISC [2] lo cual permite reducir el tiempo de ejecución mediante el uso de instrucciones cortas y simples.

[2] RISC (Computador de instrucciones reducidas -Reduced Instruction Set Computer): Un microcontrolador con este diseño tendrá una cantidad reducida de instrucciones simples, las instrucciones son ejecutadas generalmente durante un ciclo.

Cuenta con una memoria de programación flash no volátil de 256 KB y una memoria SRAM[3] de 32 KB para operar al nodo como un dispositivo FFD [4] [16].

El microcontrolador es capaz de admitir un conjunto de dispositivos periféricos, lo que permite al dispositivo actuar como un coordinador de red de área personal (PAN-Personal Area Network). Por esta razón, el nodo puede operar como un dispositivo de funcionalidad completa o un dispositivo de funcionalidad reducida. También, utiliza un oscilador interno RC9 sintonizado a 16 MHz y pre escalado a 8 MHz, por lo que está diseñado para funcionar a 16 MHz con voltajes de suministro que van desde 1,8 V a 3,6 V [4] [16].

1.3.4.2 Características generales del Transceptor RCB256RFR2 [16]

- Comunicación half dúplex.
- El hardware cumple con especificaciones del estándar IEEE 802.15.4 – 2006.
  - Evaluación de canal libre (CCA).
  - Detección de energía con (ED).
  - Detección de inicio de trama (SFD-PHY).
  - Verificación de redundancia cíclica (CRC 16).
  - Acuses de recibo (ACK)
  - Medición de intensidad de la señal recibida (RSSI11).
  - Cálculo de la calidad de enlace (LQI).
  - Control de transmisión, recepción, y retransmisión de tramas.
  - Método de acceso al medio CSMA-CA.
- Maneja un amplificador de alta potencia para trasmitir datos.
- Velocidades de datos de: 250 Kbps, 500 Kbps, 1 Mbps y 2 Mbps.
- Sensibilidad para la recepción de -100 dBm.
- Potencia de transmisión programable (-17 dBm a 3.5 dBm).
- Sintetizador PLL con 5 MHz de espacio entre canales para la banda de 2,4 GHz.
- Búfer de 128 bytes para almacenar tramas recibidas o que están por trasmitirse.

[3] SRAM (Memoria estática de acceso aleatorio-Static Random Access Memory,): Este tipo de memoria debe almacenar todas las instrucciones a ejecutar en el microcontrolador, además debe ser capaz de mantener la información mientras este conectada a una fuente de energía.

- Interfaz sencilla para manejo de registros, almacenamiento de trama y control de interrupciones específicas
- Funciona a temperaturas de entre -40°C y 125°C.
- Bajo consumo de potencia (1,8 V a 3,6 V) a (10,1 mA a 18,6 mA).
- Maneja aplicaciones como:
    - ZigBee, IEEE 802.15.4
    - ISM, RF4CE, SP100, WirelessHART y 6LoWPAN.

1.3.5 MODELO DE COMUNICACIÓN PUBLICACIÓN/SUBSCRIPCIÓN

El modelo de comunicación publicación/subscripción (Pub/Sub) está formado por dos grupos de componentes, que interactúan entre sí por medio de un intermediario (ver Figura 1.18). Un grupo está formado por entidades interesadas en consumir información, por la cual las entidades han mostrado interés con anterioridad. Por otra parte, el grupo siguiente está formado por entidades encargadas de producir y publicar la información que se consumirá posteriormente [17].

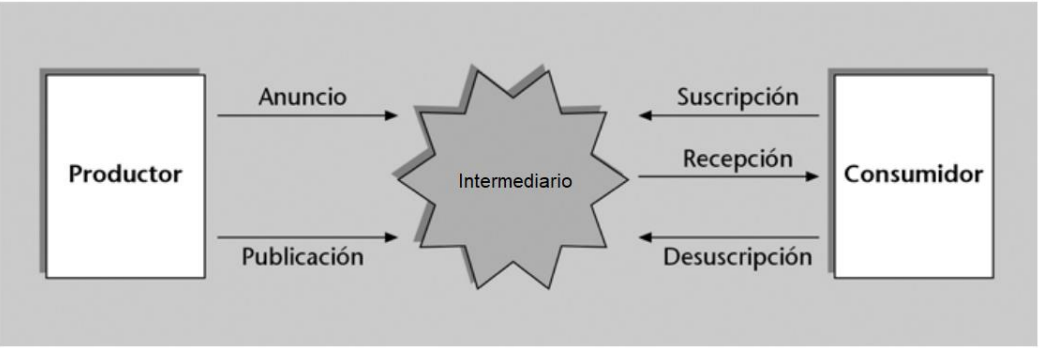


Figura 1.18. Modelos de comunicación publicación subscripción [17].

Debido a que los consumidores y productores no interactúan entre sí, se considera que el envío de mensajes es asíncrono. Por lo cual una entidad productora debe anunciar la disponibilidad de cierta información en un canal. La entidad consumidora debe realizar una subscripción al canal, de esta manera el intermediario notifica cuando el productor a publicado nueva información [18].

Los componentes del modelo Pub/Sub son descritos a continuación:

- **Productor de información:** Se refiere al usuario que posee la información a publicar. El productor transmite su información al intermediario sin tener en cuenta el número de usuarios interesados en dicha información [15].
- **Consumidor de la información:** Se refiere al usuario interesado en recibir información publicada. El consumidor tiene que subscribirse a temas de los cuales desea recibir información. En el momento que el productor publica información, al consumidor se le notificara la disponibilidad de nueva información [18].
- **Intermediario (broker):** Se encuentra ubicado entre el productor y el consumidor. Es el encargado de recibir la información de los productores y las subscripciones de los consumidores, también, es el encargado de enviar notificaciones a los consumidores cuando algún productor publica nueva información [18].
- **Canal:** Hace referencia a los conectores lógicos entre el productor y el consumidor (Figura 1.19). El canal gestiona la forma en la que se distribuyen los contenidos, si la información se entrega en el momento de la publicación, o por el contrario se entregan cuando esta es solicitada, independientemente del instante en que se genera la información [18].

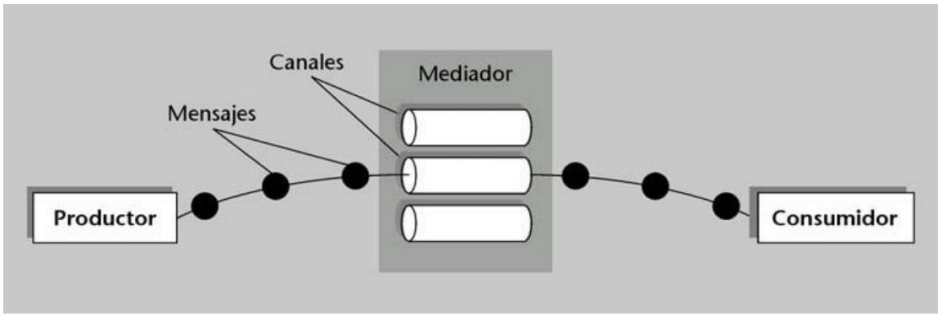


Figura 1.19 Representación de canales entre publicador y subscriptor [17].

1.3.5.1 Formas de subscripción dentro del modelo publicación/subscripción

En función de los datos existen tres formas de subscripción dentro del modelo de publicación/subscripción:

- Subscripción basada en temas: En una subscripción basada en temas, las subscripciones y las publicaciones son realizadas dentro de un conjunto de temas ya especificados. En esta forma de subscripción, durante la fase de diseño de una aplicación, será necesario crear una lista de temas de subscripción [17].
- Subscripción basada en Tipos: En una subscripción basada en tipos, los subscriptores indican que se encuentran interesados en un tipo de datos específicos [17]. Por

ejemplo, si un dispositivo publicador está encargado de censar condiciones climáticas, algún subscriptor puede estar interesado solo en datos de temperatura.

- Subscripción basada en Contenido: En una subscripción basada en contenido, el dispositivo subscriptor describe el contenido de la información que requiere recibir, es decir la entrega de mensajes está limitada a un tipo de contenido específico [17]. Por ejemplo, si un dispositivo publicador está encargado de censar velocidad, el dispositivo subscriptor solo estará interesado en recibir mensajes que sobrepasen cierto umbral ya establecido.

1.3.5.2 Protocolos bajo el modelo de Publicación/Subscripción

Existen varios protocolos de comunicación que utilizan el modelo publicación/subscripción, a continuación, se describen los más conocidos:

- MQTT (Message Queuing Telemetry Transport): Este protocolo está diseñado para manejar aplicaciones que requieran enviar poca cantidad de datos por un limitado ancho de banda de red. Existe una variación a este protocolo denominado MQTT-SN, está dirigido a dispositivos que no requieren TCP/IP [14].

- CoAP (Constrained Application Protocol): Desarrollado por el IETF, el protocolo CoAP está basado en UDP y está diseñado para usarlo en dispositivos con recursos limitados, como los nodos de una red WSN [14].

- DDS (Data Distribution Service): La especificación DDS está basado en TCP, está diseñado para sistemas distribuidos que trabajan con aplicaciones de tiempo real. Sus características principales son la confiabilidad y alto rendimiento [14].

1.3.6 PROTOCOLO MQTT

MQTT o Message Queuing Telemetry Transport es un protocolo derivado de WebSphere Message Queue (Cola de mensajes) y fue desarrollado por Andy Stanford-Clark y Arlen Nipper de IBM para manejar los desafíos únicos de vincular oleoductos y gasoductos remotos a través de satélites [19].

MQTT es un protocolo que utiliza el modelo de comunicación publicación/subscripción y es bastante simple, así como liviano. El protocolo fue diseñado para utilizar un mínimo del ancho de banda y de los recursos de los dispositivos. Además, intenta brindar confiabilidad y cierto nivel de garantía de entrega de mensajes. Estas propiedades hacen que el

protocolo sea adecuado para utilizarlo en comunicaciones M2M(Machine-to-machine[4]) [20].

MQTT puede ejecutarse sobre TCP/IP, o sobre otros protocolos de red que proporcionan conexiones bidireccionales [20], busca cumplir con los siguientes objetivos:

- Transporte de mensajes independiente de la carga útil.
- Tres niveles de calidad de servicio para la entrega de mensajes:
  - Como máximo una vez, donde los mensajes se entregan de acuerdo con los mejores esfuerzos del entorno operativo. Puede ocurrir la pérdida de mensajes [17].
  - Al menos una vez, donde se asegura que los mensajes serán se entregados, pero pueden ocurrir duplicados [17].
  - Exactamente una vez, donde se asegura que los mensajes lleguen exactamente una vez. En este nivel no se aceptan mensajes duplicados o perdidos [17].
- Una sobrecarga de transporte pequeña para reducir el tráfico de red.
- Un mecanismo que indica la existencia de desconexiones anómalas.

1.3.6.1 Arquitectura MQTT

MQTT utiliza el modelo de comunicación publicación/subscripción. Este modelo es utilizado porque tanto los clientes que generan publicaciones, así como los clientes que reciben dichas publicaciones, no están vinculados directamente. Los clientes no conocen ningún tipo de identificador, como una dirección IP o un puerto [19].

Un cliente que envía un mensaje se conoce como publicador y un cliente que recibe un mensaje se conoce como subscriptor. El dispositivo responsable de conectar clientes y filtrar datos se denomina broker (Figura 1.20), el cual funciona como intermediario entre publicador y subscriptor [19]. MQTT es un protocolo que separa con éxito a publicadores de subscritores, no es necesario identificar a alguno de ellos, el broker es la entidad que los regulará. Los modelos pub/sub, son invariantes en el tiempo [19]. Esto significa que un subscriptor puede leer y responder a un mensaje enviado por un cliente en cualquier momento. Por ejemplo, un subscriptor podría estar en un estado de baja potencia y responder a un mensaje horas más tarde.

[4] Machine-to-machine: M2M o Machine-to-machine es considerada una solución empresarial, la cual facilita la recolección de datos. de empresas, las cuales están conectadas a sus máquinas por medio de la nube. [20]

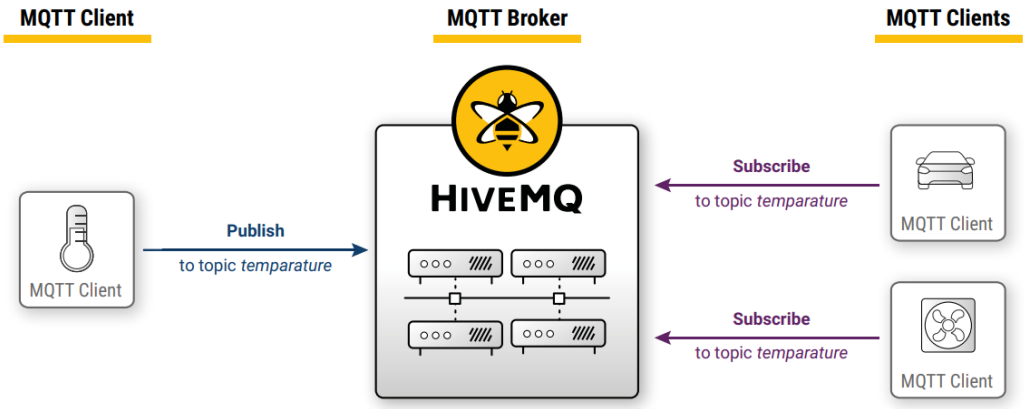


Figura 1.20 Arquitectura MQTT [19].

El broker está encargado de proporcionar un filtrado basado en temas, donde los clientes se *SUBSCRIBE*n a temas específicos, por lo que no obtienen más información de la que necesitan. Cada mensaje publicado debe ser analizado en el broker, si el mensaje contiene un tema al cual un cliente se ha subscrito el mensaje será retrasmitido, de lo contrario este es ignorado [19].

1.3.6.1.1 Cliente MQTT

Un cliente MQTT es cualquier dispositivo, desde un microcontrolador hasta un servidor completo, que ejecuta una biblioteca MQTT y se conecta a un agente MQTT a través de una red. Por ejemplo, un dispositivo muy pequeño con recursos limitados que se conecta de forma inalámbrica y tiene una biblioteca mínima es considerado un cliente MQTT. Realizar la implementación de un cliente MQTT es relativamente simple y directa, por esta razón este protocolo es apropiado para dispositivos pequeños [19]. Básicamente, un dispositivo que utilice MQTT sobre una pila TCP/IP se puede considerar cliente MQTT.

Las bibliotecas MQTT están disponibles para una gran variedad de lenguajes de programación, como son, C, C++, entre otros.

1.3.6.1.2 Broker MQTT

Un broker es el dispositivo primordial utilizado por protocolos pub/sub como MQTT. El broker MQTT es el encargado de enlazar a los clientes que publican con los clientes que consumen los mensajes publicados. El broker está a cargo de recibir todos los mensajes, filtrarlos, determinar quién se ha subscrito a cada mensaje y luego enviar el mensaje a quienes se han subscrito. Además, el broker se encarga de la autenticación y autorización del cliente, almacena información de los clientes que tienen sesiones persistentes, como las subscripciones y los mensajes perdidos. El broker es altamente escalable, puede

admitir millones de clientes MQTT conectados simultáneamente, según la implementación [19].

1.3.6.2 MQTT y el modelo de comunicación TCP/IP

El protocolo MQTT está basado en TCP, por consiguiente, el cliente como el intermediario deben tener una pila TCP (Figura 1.21). Debido a que MQTT necesita del protocolo TCP, existe cierta garantía de los paquetes de información que serán transferidos de manera confiable, no obstante, las conexiones aún pueden perderse [21].

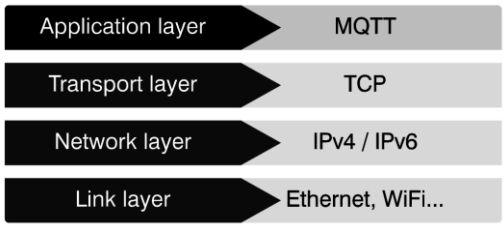


Figura 1.21. MQTT dentro de TCP/IP [21].

1.3.6.3 Formato de mensaje MQTT

Un mensaje MQTT está conformado por un encabezado fijo de 2 bytes, un encabezado variable (opcional) y una carga útil (opcional), que se codifica mediante el ordenamiento de bytes y bits. El encabezado fijo de 2 bytes siempre estará presente en todos los paquetes, mientras que el encabezado variable y la carga útil no siempre estarán presentes en el mensaje [22]. La estructura del mensaje se muestra en la Figura 1.22.

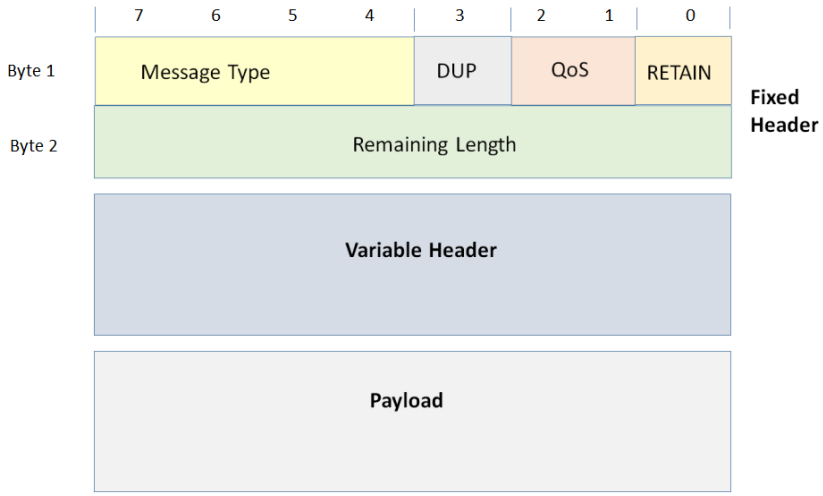


Figura 1.22. Formato de mensaje MQTT [22].

El encabezado fijo es obligatorio y sus campos se describen a continuación:

- Message Type: Indica el tipo de mensaje enviado o recibido del publicador, subscriptor o el broker [22] [23].
- Flags: Está formado por los últimos cuatro bits del primer byte, contienen indicadores específicos para cada tipo de mensaje. Son solo relevantes para el mensaje PUBLICAR y se dividen en los siguientes subcampos [22] [23]:
  - **DUP.** Utiliza un bit para indicar que el mensaje se envía o no por primera vez.
  - **QoS.** Necesita de 2 bits para codificar tres diferentes niveles de calidad de servicio.
  - **RETAIN**. Utiliza 1 bit para indicar al broker que almacene el mensaje y la QoS para futuros subscriptores.
- Remaining length: Este campo Indica la longitud del encabezado variable y de la carga útil. Tiene una longitud variable de 1 a 4 bytes.
- Variable heather: Este campo no está presente en todos los paquetes MQTT. La información que contiene el encabezado variable depende del tipo de mensaje, generalmente proporciona información adicional o indicadores del mismo. Un identificador de mensajes es común en la mayoría de los tipos de tipo de mensajes [22] [23].
- Payload: Representa la carga útil, depende de la aplicación en la que opere el protocolo. Por lo general, contiene datos para transmitir o recuperar de un tema registrado en el broker. Por ejemplo, un mensaje *PUBLISH* contiene datos asociados a un determinado tema del almacenado en el intermediario [22] [23].

1.3.6.4 Conexión MQTT

Después de que un cliente establece una conexión a nivel de red con un servidor, el cliente puede iniciar una conexión MQTT, para esto, el cliente envía un mensaje *CONNECT* al broker. En respuesta al mensaje de conexión se envía un mensaje *CONNACK* con el cual se acepta o rechaza la conexión. La conexión es rechazada si el mensaje *CONNECT* tiene un formato incorrecto o si pasa demasiado tiempo entre la apertura de un socket de red y el envío del mensaje de conexión [19].

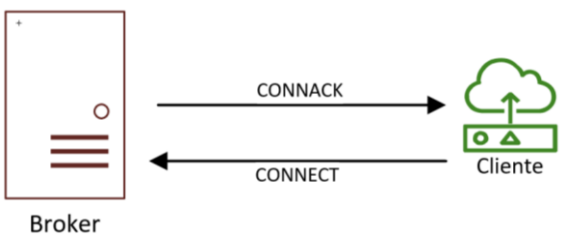


Figura 1.23. Conexión MQTT entre cliente y broker.

1.3.6.5 Publicación MQTT

Tanto el broker como el cliente tienen la capacidad de publicar mediante mensajes *PUBLISH*. El cliente puede publicar mensajes después de realizar una conexión exitosa, mientras que el broker o intermediario tiene que filtrar por tema la carga útil recibida antes de realizar una publicación. Cada mensaje *PUBLISH* debe contener un tema para que el intermediario pueda reenviarlo a cada cliente que este subscrito al tema indicado [19].

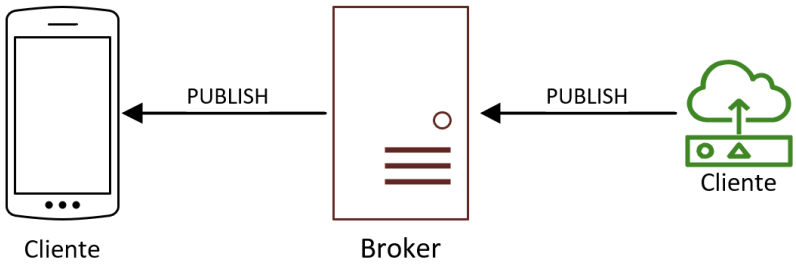


Figura 1.24. Publicación MQTT de cliente a broker y de broker a cliente.

Antes de procesar un mensaje *PUBLISH* el receptor debe reconocer el nivel de calidad de servicio, de esta forma sabrá si es necesario responder a cliente con mensajes *PUBACK*, *PUBREC*, o *PUBCOMP*, caso contrario, el emisor solo se preocupará por entregar el mensaje *PUBLISH* [19].

A un cliente publicador no le interesa conocer si los subscritores reciben su carga útil o si están interesados en sus mensajes *PUBLISH*, ya que el broker es el que se encargara de estas responsabilidades.

1.3.6.6 Subscripción MQTT

Antes de poder recibir publicaciones un cliente tiene que subscribirse, a algún tema almacenado en el broker, enviando un mensaje *SUBSCRIBE*. Este mensaje de subscripción solo contiene un ID de paquete y una lista de subscripciones. Cada subscripción contiene de un tema y un nivel de QoS [19].

Para confirmar las subscripciones, el broker envía un mensaje de reconocimiento *SUBACK* al cliente. Este mensaje contiene una lista de códigos de retorno que aceptan o rechazan las subscripciones a los temas contenidos en el broker [19].

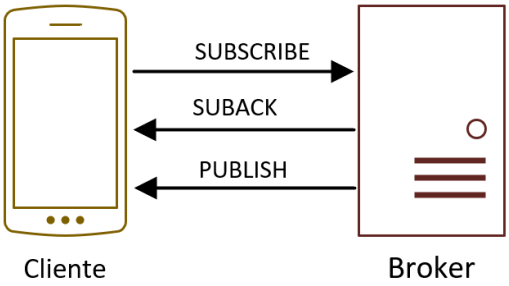


Figura 1.25. Subscripción MQTT entre cliente y broker.

1.3.6.7 Dar de baja subscripciones MQTT

Para dar de baja de una o varias subscripciones existentes el cliente envía el mensaje *UNSUBSCRIBE*. El mensaje *UNSUBSCRIBE* contiene la lista de temas de los que el cliente ya no requiere recibir información. El broker elimina las subscripciones indicadas y responde con un mensaje *UNSUBACK* [19].

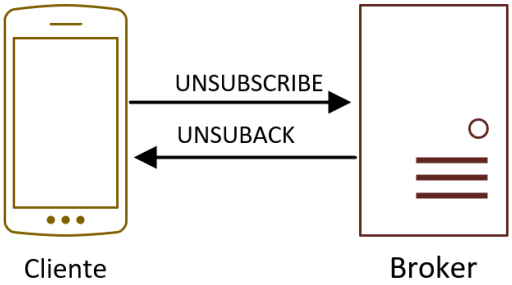


Figura 1.26. Cliente dándose de baja de una o más subscripciones.

1.3.6.8 Temas MQTT

Un tema es una cadena de texto UTF-8 utilizada por el broker para filtrar mensajes recibidos de clientes publicadores. Mediante la filtración se decide a que clientes subscriptores se debe reenviar los mensajes que le llegan al broker. El tema formara una jerarquía de varios niveles que están separados por el caracter slash [16]. Cada nivel también está formado por una cadena de texto, como se muestra en el siguiente ejemplo:

Figura 1.27. Partes de un Tema MQTT.

1.3.6.9 Caracteres comodines

Los comodines son caracteres especiales que permiten a los clientes subscribirse a más de un tema, pueden ser dé nivel único y multinivel. Estos caracteres no se pueden utilizar para publicar mensajes [24].

- Comodín de nivel único: Está representado por el símbolo más (+) y reemplaza a un único nivel de un tema. Por ejemplo, si un broker proporciona acceso a los siguientes temas [25]:
  - Myhogar/Piso_1/Dormitorio/Temperatura
  - Myhogar/Piso_2/Dormitorio/Temperatura
  - Myhogar/Piso_1/Cocina/Temperatura
  - Myhogar/Piso_2/Baño/Temperatura

Y el cliente requiere información solo de los dormitorios en cada piso puede utilizar:

Myhogar/**+**/Dormitorio/Temperatura

- Comodín multinivel: Está representado por el símbolo numeral (#) y se utiliza para reemplazar múltiples niveles de un tema. Por ejemplo, si un broker proporciona acceso a los siguientes temas [25]:
  - Myhogar/Piso_1/Dormitorio/Temperatura
  - Myhogar/Piso_2/Dormitorio/Temperatura
  - Myhogar/Piso_1/Cocina/Temperatura
  - Myhogar/Piso_2/Baño/Temperatura

Y el cliente requiere información de todas las habitaciones disponibles debería utilizar:

Myhogar/#

1.3.6.10 Datos will (temas y mensajes de última voluntad)

Cuando se presenta una desconexión abrupta, ya sea por problemas en el hardware, pérdida de conexión de la red subyacente, errores en el intercambio de mensajes, etc. El cliente no tendrá la posibilidad de avisar al broker, o a los clientes que consuman información proveniente de este cliente, cuando este tipo de desconexión ocurra [26].

Para solucionar estos problemas se utiliza un tema y mensaje de última voluntad. Esta información también denominada *datos will* es enviada, por los clientes que lo deseen, en el momento que se ejecuta la conexión cliente-broker [27]. Los temas y mensajes son almacenados para ser enviados cuando se presente una desconexión no prevista.

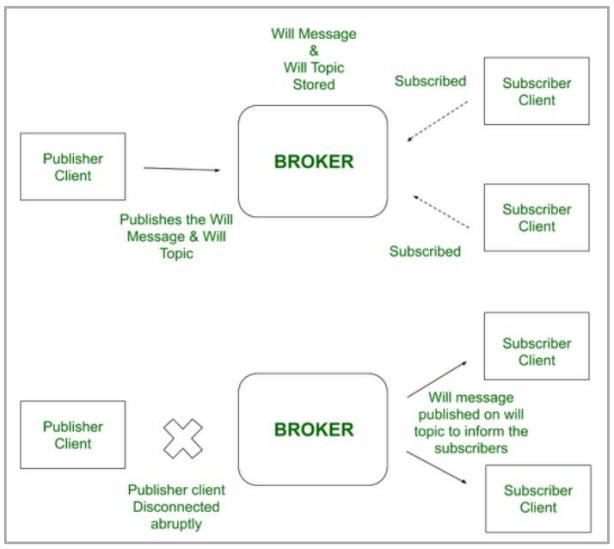


Figura 1.28. Publicación de mensaje de última voluntad [27].

1.3.6.11 Calidad de servicio en MQTT

El remitente define el tipo de Calidad de servicio (QoS) en MQTT. Existen tres niveles de calidad de servicio descritas a continuación [22]:

- QoS 0 (transmisión no asegurada): Este es el nivel más bajo de QoS y es un proceso de entrega de mejor esfuerzo. El destinatario no reconoce la recepción de un mensaje, tampoco el remitente vuelve a intentar transmitir.
- Qos 1 (transmisión asegurada): Este nivel garantiza que el mensaje se entregue al menos una vez al receptor. Se puede retransmitir varias veces y el receptor responderá con un mensaje *PUBACK*.
- QoS 2 (servicio asegurado en aplicaciones): Este es el nivel más alto de QoS, tanto el remitente como el destinatario se aseguran e informan cuando un mensaje se ha enviado correctamente. Si se envía un mensaje con QoS-2, el receptor responderá con un mensaje *PUBREC* al remitente. Esto indica que el mensaje ha sido reconocido y el remitente reaccionará con un mensaje *PUBREL*. El mensaje *PUBREL* permite al receptor descartar cualquier retransmisión de mensajes. El receptor responde con un mensaje *PUBCOMP* después de recibir el mensaje *PUBREL*.

1.3.7 PROTOCOLO MQTT-SN

El protocolo MQTT-SN es considerado como una adaptación del protocolo MQTT para ser utilizado sobre dispositivos con recursos limitados, por lo que es ideal para redes inalámbricas de sensores. Básicamente, cualquier red que proporcione un servicio de transferencia de datos bidireccional es compatible con MQTT-SN. Por esta razón, TCP/IP no es indispensable para su operación [28]. De manera general se puede definir a MQTT-SN como un protocolo de publicación/subscripción optimizada para WSNs formadas con dispositivos que poseen escasa capacidad de procesamiento, poca memoria, y necesiten baterías [20].

1.3.7.1 Arquitectura MQTT-SN

La arquitectura utilizada por MQTT-SN se puede ver en la Figura 1.29. Hay que tener en cuenta que MQTT-SN tienen que trabajar en conjunto con MQTT, por esta razón se definen tres componentes que se describen continuación:

1.3.7.1.1 Clientes MQTT-SN

Los clientes MQTT-SN son los encargados de publicar y subscribirse al broker MQTT, sin embargo, no pueden conectarse a él directamente [28].

1.3.7.1.2 Forwarder MQTT-SN

Los clientes MQTT-SN también pueden acceder a un gateway a través de un forwarder. El forwarder simplemente encapsula los mensajes MQTT-SN en una trama de reenvío para enviarlos a un gateway. Cuando el forwarder recibe una mensaje encapsulado la desencapsula para después enviarla al cliente [28].

1.3.7.1.3 Gateway MQTT-SN

Funciona como intermediario entre cliente y broker, es responsable de convertir los mensajes de comunicación MQTT-SN a MQTT. Un Gateway puede estar integrado al broker o ser independiente [29].El gateway independiente debe tener la capacidad de convertir un mensaje MQTT-SN a un mensaje MQTT y viceversa ya que para comunicarse con los clientes utiliza mensajes MQTT-SN mientras que para comunicarse con el broker utiliza mensajes MQTT [28].

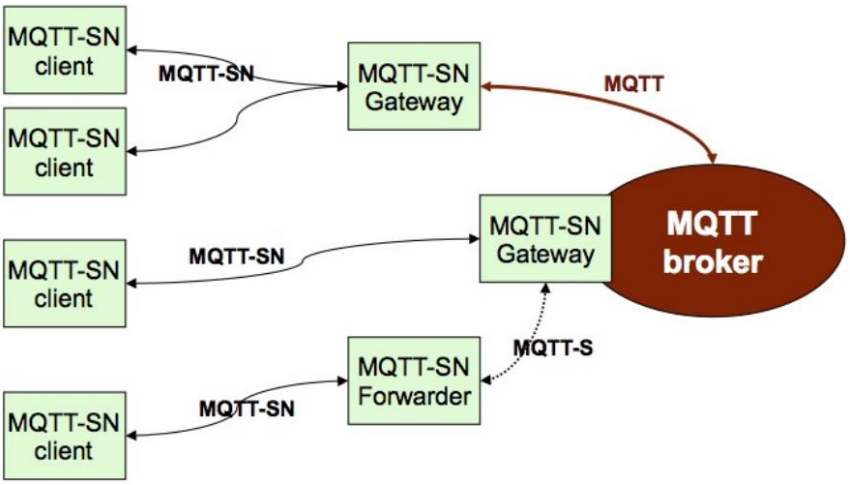


Figura 1.29. Arquitectura MQTT-SN [28].

1.3.7.1.4 Gateway transparente

Un gateway transparente establecerá y mantendrá una conexión MQTT con el broker por cada cliente MQTT-SN (Figura 1.30). Por esta razón, se considera las conexiones entre clientes y el broker son casi transparentes. Debido a que todos los intercambios de mensajes son de extremo a extremo, entre el cliente MQTT-SN y el servidor MQTT, el gateway transparente realizará una traducción de protocolos por cada flujo de mensajes [28].

1.3.7.1.5 Gateway agregado

Un gateway agregado, a diferencia de un transparente, maneja solo una conexión MQTT para administrar a todos los clientes MQTT-SN conectados. Los flujos de mensajes entre los clientes y el gateway son combinados para después enviarlos al broker, el gateway decidirá qué mensaje será atendido por la conexión MQTT como se muestra en la Figura 1.30. Un gateway de agregación puede ser utilizado cuando un broker MQTT, no admite una gran cantidad de conexiones simultaneas [28].

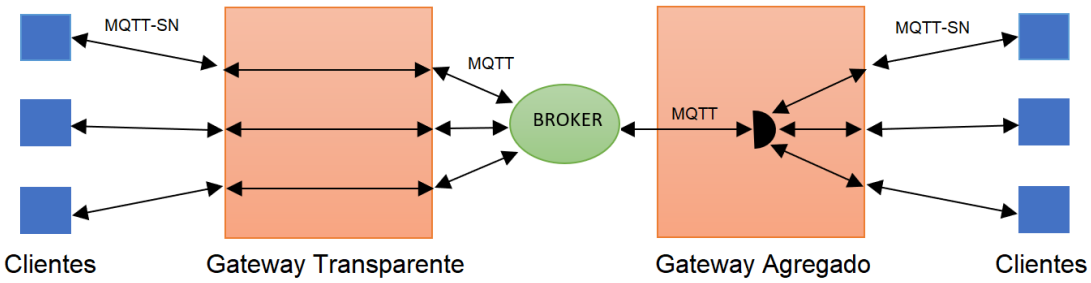


Figura 1.30. Gateway MQTT-SN transparente y agregado.

La implementación de un gateway transparente es más simple en comparación con la de un gateway agregado. Sin embargo, hay que tener en cuenta la cantidad de conexiones simultaneas que es capaz de sostener el broker MQTT, así como la sobrecarga de información que supone la conexión de una gran cantidad de clientes MQTT-SN [20].

1.3.7.2 Formato general del mensaje MQTT-SN

Un mensaje MQTT-SN se compone de un encabezado fijo y una parte variable, tal como se muestra en la Figura 1.31. El encabezado fijo es obligatorio y sus campos son iguales para todos los mensajes, por el contrario, la parte variable y sus campos depende del tipo de mensaje MQTT-SN [28].

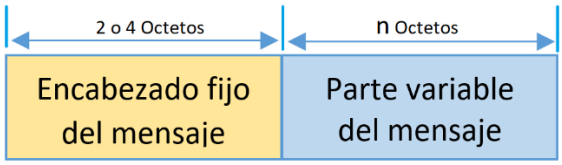


Figura 1.31. Formato general del mensaje MQTT-SN.

1.3.7.2.1 Encabezado del mensaje MQTT-SN

El encabezado del mensaje cuenta con dos campos los cuales serán descritos a continuación:

- **Campo Length:** Indica el total de octetos del mensaje, incluye el tamaño el mismo. Su longitud puede ser de 1 o 3 octetos. Se utiliza el formato de 1 octeto para mensajes con una longitud menor o igual a 255 octetos [28]**.**

  El formato de 3 octetos lo utilizaran mensajes con una longitud mayor a 256 octetos. En este caso el primer octeto del campo longitud se codifica con el valor 0x01, los dos octetos restantes indicaran el número de octetos total del mensaje.

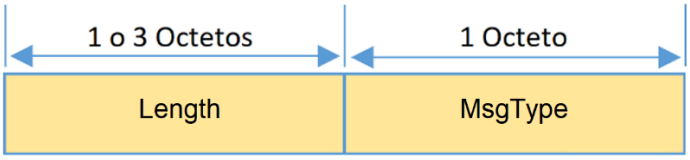


Figura 1.32 Encabezado del mensaje MQTT-SN.

- **Campo MsgType:** Este campo sirve para identificar de manera única a cualquier mensaje MQTT-SN [28]**.** Tiene una longitud de 1 octeto y sus valores se los puede apreciar en la Tabla 1.2.

Tabla 1.2. Valores del Campo tipo y parte variable de los mensajes MQTT-SN [28].

NOMBRE DEL MENSAJE	VALOR CAMPO TIPO		PARTE VARIABLE DEL MENSAJE			
	HEX	DEC				
ADVERTISE	0x00	0	GwId	Duration		
SEARCHGW	0x01	1	Radius			
GWINFO	0x02	2	GwId	GwAdd		
Reservado	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			
Reservado	0x11	17				
SUBSCRIBE	0x12	18	Flags	MsgId	TopicName or TopicId	
SUBACK	0x13	19	Flags	TopicId	MsgId	Return Code
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)			
Reservado	0x19	25				
WILLTOPICUPD	0x1A	26	Flags	Willtopic		
WILLTOPICRESP	0x1B	27	Return Code			
WILLMSGUPD	0x1C	28	Willmsg			
WILLMSGRESP	0x1D	29	ReturnCode			
Reservado	0x1E-0xFD	30-253				
Mensaje Encapsulado	0xFE	254	Ctrl	Wireless Node Id	MQTT-SN message	
Reservado	0xFF	255				

1.3.7.2.2 Parte variable del mensaje

Existen 14 campos que pueden constituir la parte variable de un mensaje MQTT-SN, no obstante, depende del tipo de mensaje el uso y distribución de cada campo (Tabla 1.2) [26]. A continuación, se describe a cada uno de los campos:

- **ClientId (1-23 Caracteres):** Este campo es una cadena de caracteres que permite al servidor identificar de manera única a cualquier cliente.
- **Data (n bytes):** Este campo contiene la carga útil que transportan los mensajes *PUBLISH*. Su longitud depende del tipo de red sobre la que esté operando MQTT-SN.
- **Duration (2 bytes):** Define la duración de un período de tiempo en segundos.
- **Flags (1 byte):** Contiene seis banderas descritas a continuación. (Figura 1.33)

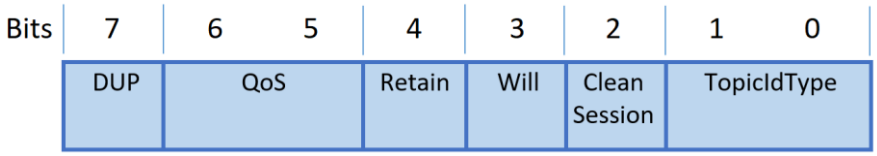


Figura 1.33. Campo flags MQTT-SN.

  - **DUP:** Utilizado en los mensajes *PUBLISH* para indicar si el mensaje se transmite por primera vez (0) o si es retransmitido (1).
  - **QoS:** Indica el nivel de calidad de servicio QoS 0 (0b00), QoS 1 (0b01), QoS 2 (0b10) y QoS -1 (0b11); este último es propio de MQTT-SN y solo es utilizado en publicaciones que no requieren procedimientos previos.
  - **Retain:** Solo es utilizado en mensajes *PUBLISH* para que el cliente indique al servidor que debe reemplazar cualquier mensaje retenido existente y almacenar el mensaje recibido.
  - **Will:** Es utilizado en mensajes *CONNECT* enviados por el cliente, indicando que se va a enviar un tema Will y mensaje Will.
  - **CleanSeccion:** Si está establecida en 0 indica al broker que debe reanudar la comunicación con el cliente siempre y cuando tenga una sesión asociada a este. Además, el Cliente y el Servidor tienen que almacenar la sesión después de desconectarse. Si CleanSession se establece en 1, el cliente y el servidor deben descartar cualquier sesión anterior e iniciar una nueva, esta nueva sesión se eliminará cuando la conexión de red termine. CleanSeccion es utilizado solo en mensajes *CONNECT*.

- **TopicIdType:** En un mensaje MQTT-SN, esta bandera indica el valor del campo TopicId o TopicName. Se puede tener un identificador de tema normal (0b00), un identificador de tema predefinido (0b01) o un nombre de tema corto (0b10).

- **GwAdd (n bytes):** Contiene la dirección de un gateway que está presente en la red.
- **GwId (1 byte):** Permite identificar de forma exclusiva un gateway.
- **MsgId (2 bytes):** Utilizado para asociar un mensaje con su respectivo acuse de recibo, cuando existe un intercambio de mensajes que requieren confirmación.
- **Protocol Id (1 byte):** Utilizado para representar el nombre y la versión del protocolo. Solo está presente en mensajes *CONNECT*.
- **Radius (1 byte):** Indica el valor del radio de transmisión de mensaje broadcast.
- **ReturnCode (1 byte):** Es utilizado para aceptar o rechazar un mensaje que espera un acuse de recibo, sus valores y significado de muestra en la Tabla 1.3.

Tabla 1.3 Valores del campo ReturnCode [26].

Campo ReturnCode	
Valor	**Significado**
0x00	Aceptado
0x01	Rechazado: Existe cogestión
0x02	Rechazado: TopicId invalido
0x03	Rechazado: No soportado
0x04 - 0xFF	Valores reservados

- **TopicId (2 bytes):** Contiene un valor que permite identificar un tema en específico. Los valores 0x0000 y 0xFFFF no pueden utilizarse.
- **TopicName (n bytes):** Contiene una cadena de caracteres que representan un nombre de tema.
- ***WILLMSG* (n bytes):** Contiene una cadena de caracteres que representan el mensaje de última voluntad.
- ***WILLTOPIC* (n bytes):** Contiene una cadena de caracteres que representan el tema al que pertenece el mensaje de última voluntad.

1.3.7.3 ID de tema predefinido

Es un identificador de tema el cual ha sido asignado previamente a un nombre de tema, el ID es conocido tanto por el cliente como por el gateway. Esto se indica en el campo flags

del mensaje. Cuando se utilizan identificadores de temas predefinidos, ambas partes pueden enviar mensajes de publicación de inmediato; no se requiere el procedimiento para registrar temas convencionales. Hay que tener en cuenta que, si el receptor recibe un mensaje *PUBACK* que indica un rechazo por identificador de tema inválido, no puede resolver este problema al registrarse nuevamente.

1.3.7.4 Nombre de tema corto

Un nombre de tema está representado por solo dos caracteres, por lo cual el campo tendrá una longitud de dos octetos. No se requiere iniciar procedimiento alguno para su registro. Además, vale la pena señalar que usar caracteres comodines en subscripciones a nombres de temas cortos no es una buena práctica, debido a que, no es posible manejar una jerarquía de niveles de temas con solo dos caracteres.

1.3.8 LENGUAJE DE MODELADO UNIFICADO UML

El lenguaje de modelado unificado permite la representación visual de estados, objetos, y procesos de un sistema. UML ayuda a muchos desarrolladores con el desarrollo y representación de sistemas, de tal forma que sea entendible por terceras personas. Además, el lenguaje permite garantizar una arquitectura de información estructurada de un proyecto. El desarrollo de software orientado a objetos y la visualización de procesos empresariales son los usos más importantes que se le da a UML [30].

Los diagramas UML utiliza los siguientes componentes para representar un sistema:

- Objetos individuales (elementos básicos).
- Clases (combina elementos con las mismas propiedades).
- Relaciones entre objetos (jerarquía y comportamiento/comunicación entre objetos).
- Actividad (combinación compleja de acciones/módulos de comportamiento).
- Interacciones entre objetos e interfaces.

UML no es considerada una metodología, no obstante, proporciona diversos tipos de diagramas lo cuales son utilizados dentro de una metodología determinada, facilitan la comprensión de un proyecto en desarrollo [25].

Los diagramas UML más populares son: diagrama de casos de uso, diagrama de clases, diagrama de estados, diagrama de secuencia, diagrama de componentes diagrama de actividades, y diagrama de implementación.

1.3.9 DIAGRAMA DE SECUENCIA

El diagrama de secuencia es un tipo de diagrama del lenguaje de modelado unificado, utilizado para modelar procesos de programación, así como procesos de negocio con el objetivo de presentar contenidos complejos mediante elementos gráficos. Por lo cual, UML establece una notación estandarizada y recurre a elementos visuales para representar componentes y su comportamiento [30].

El diagrama de secuencia, en desarrollo de procesos de programación, es utilizado específicamente para mostrar interacciones entre componentes (intercambio de mensajes entre objetos). Las interacciones serán mostradas en el orden secuencial en que ocurren [31].

1.3.9.1 Elementos de un diagrama de secuencia

- **Línea de Vida:** representa a un participante de un diagrama de secuencia, comúnmente posee un rectángulo en la parte superior que contiene el nombre del componente (objeto, entidad, proceso, etc.). Si la línea de vida contiene el nombre self representa al clasificador que posee el diagrama de secuencia [30].

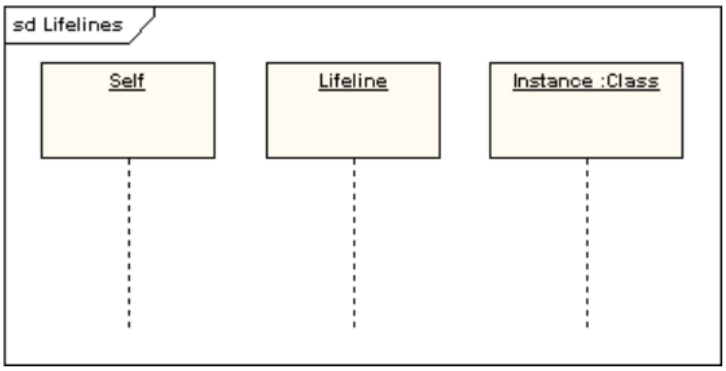


Figura 1.34 Línea de vida de un diagrama de secuencia [30].

Mensajes: En un diagrama de secuencia los mensajes son elementos básicos, los cuales permite interactuar a las líneas de vida entre sí. Mediante diferentes formas de flechas se representa relaciones direccionales o flujos de información. Existen diferentes formas de representar una secuencia de mensajes [30], depende del tipo de mensaje la forma que tomara la flecha tal y como se puede ver en la Figura 1.35.

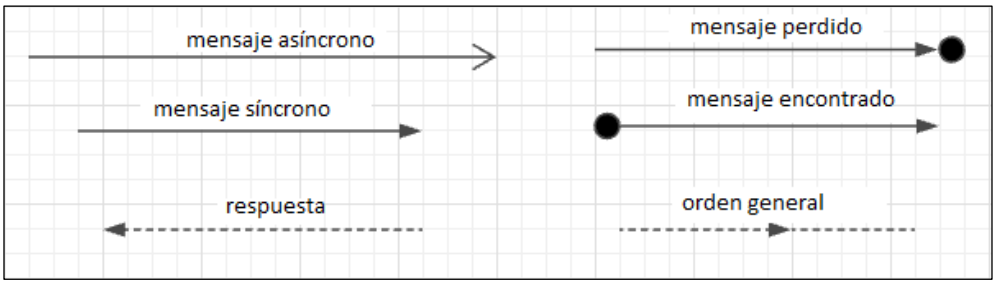


Figura 1.35. Tipo de mensaje y su representación en diagramas de flujo [30].

En una línea de vida que envía un mensaje a otra, la flecha que representa el mensaje, iniciara en el emisor y la punta de la misma terminara en el receptor. El nombre de mensaje está ubicado encima de la flecha. Por lo general, el primer mensaje de un diagrama de secuencia empieza en la parte superior de lado izquierdo del diagrama, facilitando la lectura del mismo. Los siguientes mensajes deben ser agregados debajo del mensaje anterior.

- **Foco de control:** Permite la ejecución o activación de una secuencia de mensajes, y representa todo el tiempo en el que se lleva a cabo un trabajo. Tambien, representa momentos en los que se deba cumplir una condición para transmitir un mensaje. Se puede decir que este componente se utiliza para control de flujo de mensajes [32] [31]. El Foco de control se representa mediante un rectángulo que recorre la línea de vida.

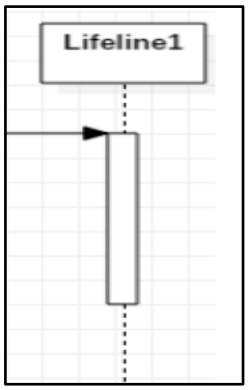


Figura 1.36 Foco de control sobre línea de vida [30].

- **Marco:** Este elemento es utilizado como límite gráfico de un diagrama de secuencia. Se representa mediante un rectángulo, y cuenta con una etiqueta en la esquina superior izquierda que contiene una denominación del diagrama que delimita. El elemento marco es opcional [31].

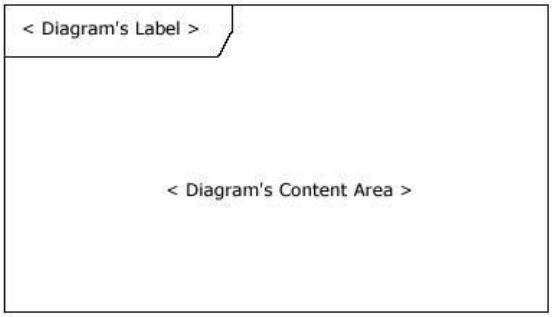


Figura 1.37 Marco contenedor de un diagrama de sequencia [31].

1.3.10 MÁQUINA DE ESTADO FINITO

Una Máquina de Estados Finitos o Autómata Finito es un modelo abstracto para la manipulación de símbolos, permite conocer si una cadena de símbolos pertenece a un lenguaje o a su vez genera otro conjunto de símbolos [33]. Una MEF está compuesta por un conjunto de estados que incluye un estado inicial y final, además, depende de la cadena ingresada y los cambios de estado que sufre la MEF.

Una MEF se compone de cinco partes y se pueden definir como una quíntupla:

$$A = \{Q, q0, F, \Sigma, \delta\}$$

donde:

Q: Conjunto finito de estados.

q0: Estado inicial donde q0 ∈ Q.

F: Conjunto de estados finales

Σ: Alfabeto finito de entrada.

δ: Función de Transición $Q \times \Sigma \rightarrow Q$

Si la MEF se halla en el estado qi (donde qi ∈ Q) e ingresa el símbolo a (donde a ∈ Σ), causa que la MEF cambie del estado qi al estado qk. La función δ, llamada función de transición, describe este cambio de la forma δ(qi, a) → qk de esta forma se obtienen un nuevo estado [33].

Otra forma de representar un MEF es mediante un diagrama de estados, estos diagramas muestran de forma gráfica todos los elementos de la máquina de estados tal como se muestra en la Figura 1.38.

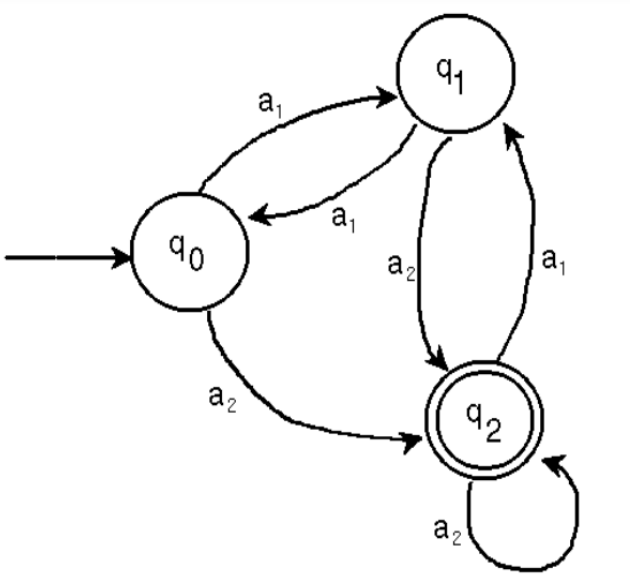


Figura 1.38 Ejemplo de diagrama de estados [33].

Para describir una función de transición de estados δ también se puede utilizar tablas, las columnas se etiquetan con los símbolos de entrada mientras que las filas son etiquetadas con los estados y en las intersecciones se colocan los nuevos estados δ(qi, a), tal como se muestra en la Figura 1.39.

	a_1	a_2
q_0	q_1	q_2
q_1	q_0	q_2
q_2	q_1	q_2

Figura 1.39 Función de transición de estados δ [33].

1.3.10.1 Máquinas de Estados Finitos Transductores

Las máquinas de estados transductores son una derivación de la MEF, con la notable diferencia de que entregan como resultado un conjunto de símbolos que pertenecen a un lenguaje, las aceptadoras nos indican si un conjunto de símbolos pertenece o no al lenguaje. El conjunto de estados finales es cambiado por una función de salida, que toma como parámetro el estado actual o una transición de la MEF y arroja un elemento del conjunto de símbolos de salida [33]**.** Por lo tanto, se tiene dos tipos de máquinas de estados transductores:

1.3.10.1.1 Máquina de Moore

La máquina de Moore requiere de un estado inicial q0, la función de salida λ entrega un símbolo s en cuanto la máquina llegue a un estado, la función de transición δ lee un

elemento de la cadena de entrada Σ e indica el nuevo estado que adoptaremos [33].La máquina de Moore se puede representar mediante una tupla o un diagrama de estados como se puede ver en la Figura 1.40.

Una máquina de Moore se define como una 6-tupla $\{Q, \Sigma, S, \delta, \lambda, q0\}$ donde:

Q: Conjunto finito de estados.
Σ: Alfabeto de entrada.
S: El alfabeto de salida.
δ: Función de transición Q × Σ → Q.
λ: Función de Q a S, si q arroja una s donde s ∈ S y q ∈ Q.
q0: Estado inicial.

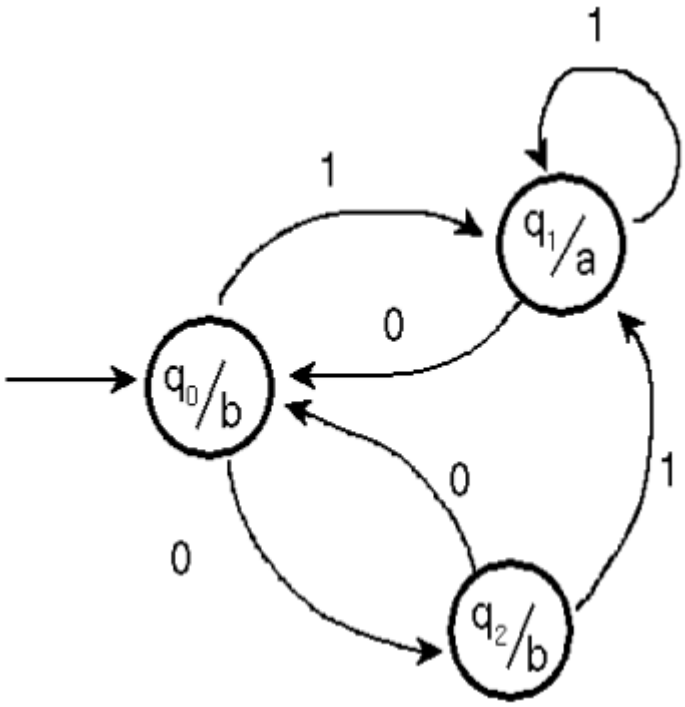


Figura 1.40 Ejemplo de diagrama de máquina de Moore [33].

1.3.10.1.2 Máquina de Mealy

Esta máquina de estados es similar a la de Moore con la notable diferencia de que la función de salida λ entregara un resultado en cada transición de estados y no cuando se llega a un estado [33]. La máquina de Mealy se puede representar mediante una tupla o un diagrama de estados como se puede ver en la Figura 1.41.

Una máquina de Mealy se define como una 6-tupla $\{Q, \Sigma, S, \delta, \lambda, q0\}$, donde:

Q: Conjunto finito de estados.

Σ: Alfabeto de entrada.

S: Alfabeto de salida.

δ: Función de transición $Q \times \Sigma \rightarrow Q$

λ: Función de salida $Q \times \Sigma \rightarrow S$, $\lambda(qi, a) \rightarrow s$ donde s ∈ S, q ∈ Q y a ∈ Σ.

q0: Estado inicial.

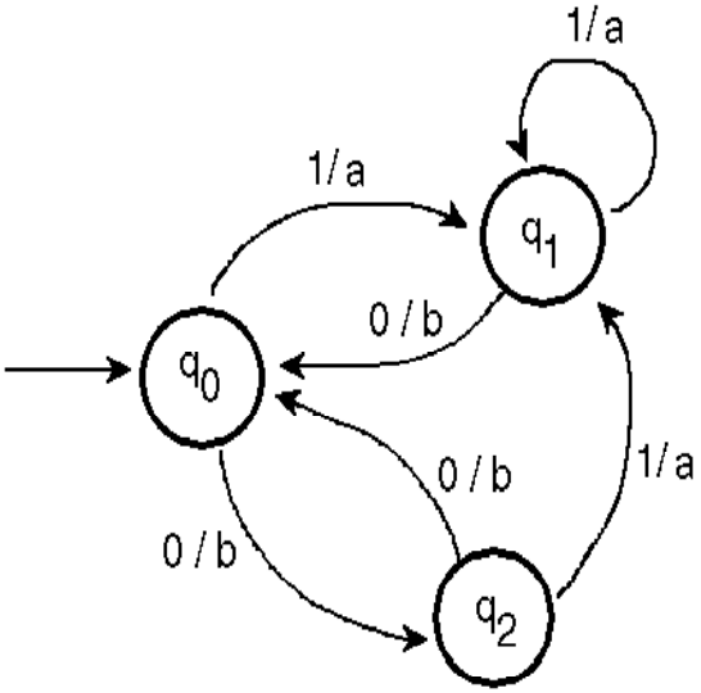


Figura 1.41 Ejemplo de diagrama de máquina de Mealy [33].

2 METODOLOGÍA

En este capítulo se especifica la metodología empleada para desarrollar la máquina de estados finitos del protocolo MQTT-SN para su operación sobre IEEE 802.15.4 en topologías lineales. Se tendrá en cuenta el funcionamiento del nodo RCB256RF, el estándar IEEE 802.15.4, y su contabilidad con el protocolo MQTT-SN, para el desarrollo de un diagrama de MEF que describa el funcionamiento del protocolo sobre el nodo sensor RCB256RF. Con el objetivo de comprobar el funcionamiento de la MEF, se representa la máquina de estados mediante el lenguaje de especificación y descripción (SDL). El lenguaje SDL permite codificar los nodos de tal manera que se pueda apreciar los cambios de estados, soportados por el nodo cliente y nodo gateway.

Para cumplir con los objetivos propuestos en este trabajo de titulación se realizan los siguientes pasos:

- Identificación de inconvenientes cuando el protocolo MQTT-SN opera sobre IEEE 802.15.4
- Diseño de diagramas de secuencia del protocolo MQTT-SN.
- Determinación de los mensajes MQTT-SN a intercambiar entre el cliente y el gateway.
- Determinación de otros eventos a utilizarse en la máquina de estados finitos.
- Desarrollo de la máquina de estados finitos.
- Representación de la máquina de estados mediante el lenguaje SDL.
- Codificación de estados del cliente.
- Codificación de estados del Gateway.
- Implementación de estados en simulador JFLAP.

2.1 IDENTIFICACIÓN DE INCONVENIENTES CUANDO EL PROTOCOLO MQTT-SN OPERA SOBRE IEEE 802.15.4

MQTT-SN se desarrolló originalmente para ejecutarse sobre ZigBee, el cual ha seleccionado el estándar IEEE 802.15.4 como protocolo para las capas PHY y MAC, proporcionando así interoperabilidad entre productos de diferentes proveedores [26]. Por lo cual se puede afirmar que, el protocolo MQTT-SN puede trabajar sobre las capas proporcionadas por el estándar IEEE 802.15.4, en una red de sensores inalámbricos.

MQTT-SN está diseñado para ser independiente de las capas inferiores sobre las que esté operando. Una red que tenga la capacidad de proporcionar un servicio de transferencia de datos bidireccional entre cualquier nodo y uno en particular (gateway) debe ser compatible

con MQTT-SN [28]. Por esta razón, la encapsulación de los mensajes MQTT-SN sobre tramas 802.15.4 no debe afectar al funcionamiento del protocolo.

2.2 DISEÑO DE DIAGRAMAS DE SECUENCIA DEL PROTOCOLO MQTT-SN

En este literal se describirá los procedimientos indicados en la especificación MQTT-SN, y sus correspondientes diagramas de secuencia. Todo esto con el fin de identificar las diferentes secuencias de mensajes intercambiadas entre dispositivos cliente y Gateway, durante la ejecución de cada procedimiento.

El protocolo MQTT-SN maneja 12 procedimientos para su funcionamiento [28]. A continuación, se presenta la descripción de cada procedimiento junto a sus respectivos diagramas de secuencia.

2.2.1 PROCEDIMIENTO ANUNCIO Y DESCUBRIMIENTO DE GATEWAY

Para anunciar su presencia un Gateway primero debe establecer conexión con un broker, para después enviar mensajes *ADVERTISE* a todos los dispositivos que forman parte de la red MQTT-SN. Si el gateway es parte del broker anunciará su presencia al instante. Los mensajes *ADVERTISE* son enviados periódicamente. Cada período de tiempo tendrá una duración *TADV*, indicada en el campo Duration del mensaje.

Si nuevos clientes se integran a la red pueden esperar mensajes *ADVERTISE* o enviar mensajes *SEARCHGW*. Cada cliente esperará un tiempo aleatorio, de entre 0 a *TEARCHGW*, antes de transmitir un mensaje *SEARCHGW,* en respuesta el gateway enviará un mensaje *GWINFO* para indicar que el dispositivo está activo. Los mensajes *SEARCHGW* pueden ser retransmitidos cuando no existe respuesta ante su envío. Cada intervalo de tiempo entre dos mensajes *SEARCHGW* consecutivos tienen que aumentar exponencialmente. En el diagrama de la Figura 2.1 se puede observar a dos clientes que descubren un gateway activo, el cliente 1 solo espera por el mensaje de advertencia del gateway, mientras que el cliente 2 solicita un *GWINFO*. Después de estas interacciones los dos clientes esperan por los siguientes mensajes *ADVERTISE.*

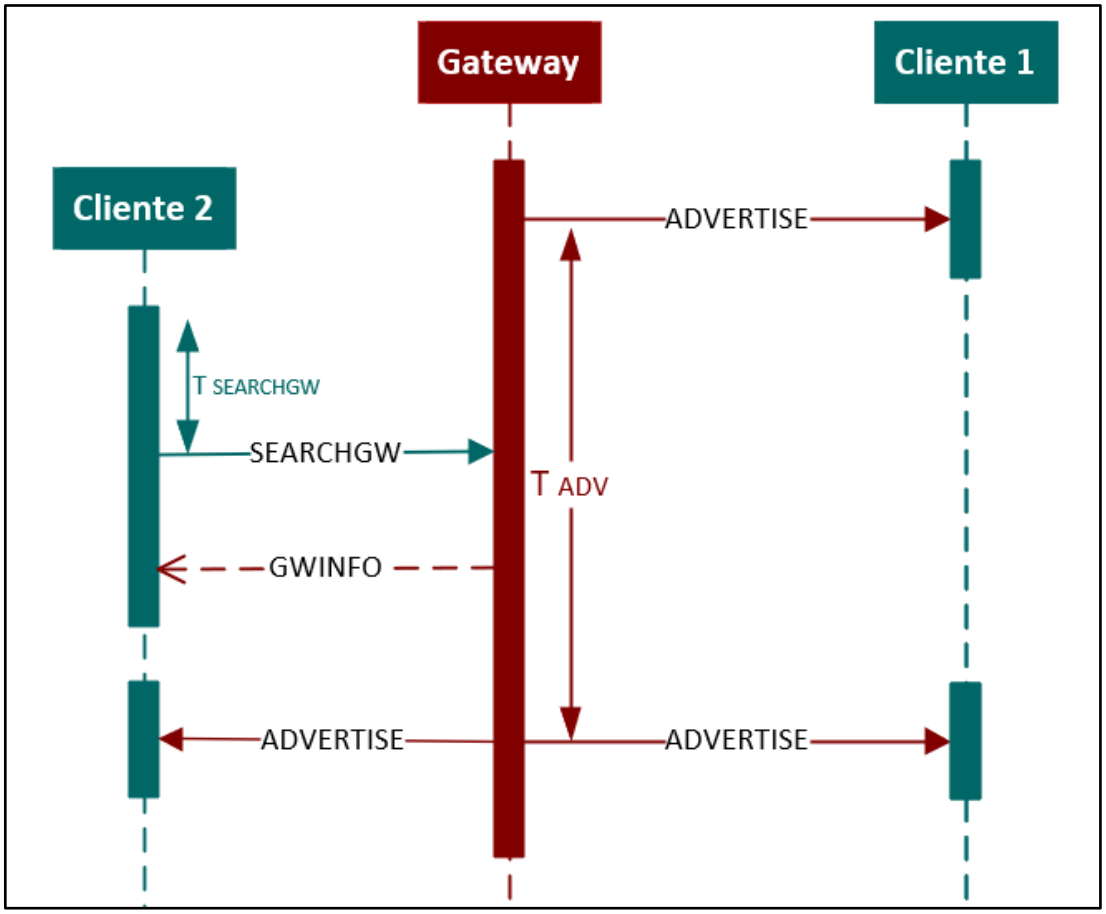


Figura 2.1 Gateway anunciando su presencia a dos clientes

Por otro lado, todos los clientes contienen una lista que almacena identificadores de gateways activos. La lista se mantiene actualizada con la información de mensajes *ADVERTISE* y *GWINFO* enviados por el gateway.

En una interacción entre clientes, un cliente tiene la posibilidad de responder a un mensaje *SEARCHGW* con su respectivo *GWINFO*, si su lista de gateways contiene información de al menos un gateway activo. El cliente selecciona un dispositivo activo de la lista para el envío de su dirección e identificación dentro de un *GWINFO*. La transmisión de mensajes *GWINFO*, por parte del cliente, debe ser retrasada un tiempo *TGWINFO* porque el gateway tiene prioridad en responder a un *SEARCHGW*. En la Figura 2.2 se muestra el diagrama de secuencia, donde el cliente 2 desea conocer la existencia del gateway activo. Tanto el cliente 1 como el gateway reciben el mensaje *SEARCHGW*, sin embargo, este cliente cancela el envío de su respuesta ya que el gateway tiene prioridad en responder.

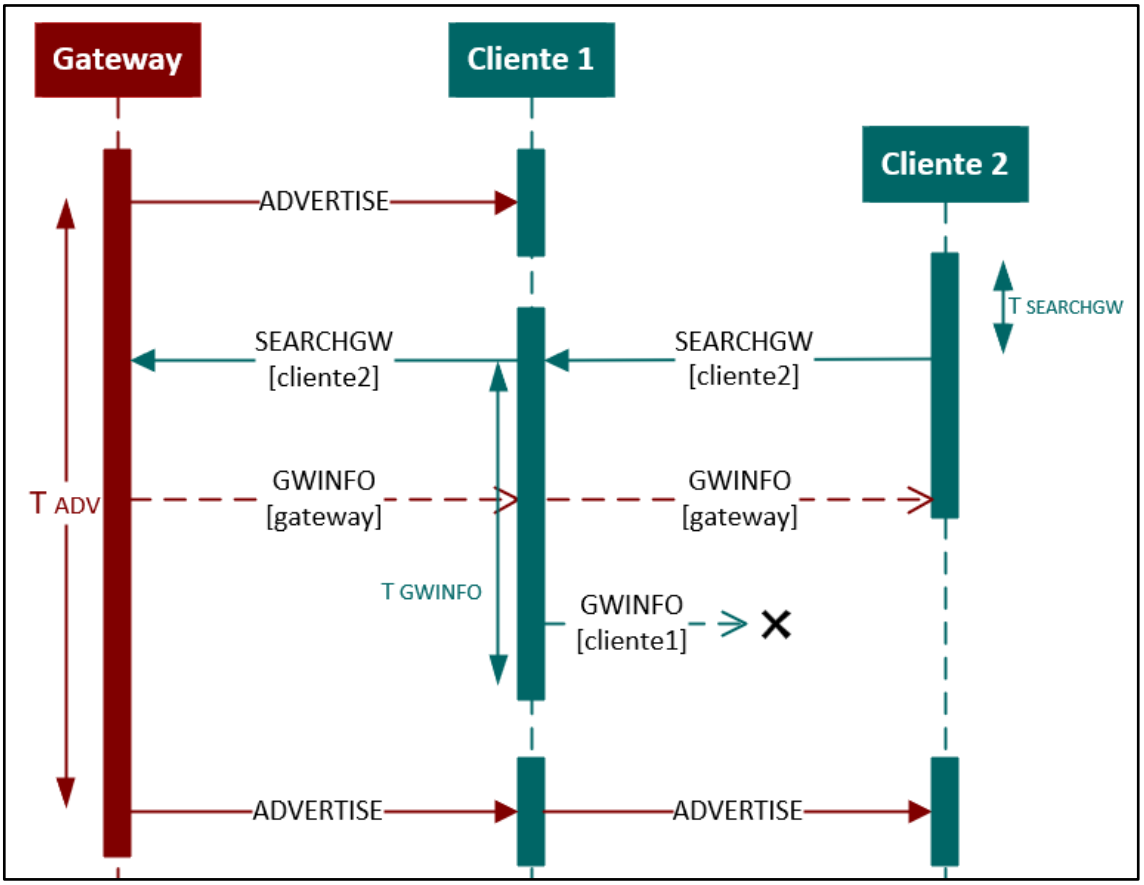


Figura 2.2 Cliente solicitando información, acerca de un gateway activo, a otro cliente.

Es posible que la red funcione con más de un gateway, pero un cliente es capaz de conectarse solo a un gateway, si la conexión falla el cliente buscará otro gateway.

Una red MQTT-SN es capaz de admitir varios gateways inactivos, los cuales no envían ningún tipo de mensaje. Un gateway inactivo solo permite la recepción de mensajes *ADVERTISE* de otros gateways activos, sin embargo, iniciará el envío de sus propios mensajes *ADVERTISE* en cuanto deja de recibirlos un par de veces.

Como se ve en el diagrama de secuencia de la Figura 2.3, el cliente y el gateway 2 (inactivo) reciben el mensaje de anuncio del Gateway 1 (activo), sin embargo, los mensajes de este último se pierden NADV veces, esto hace que el gateway 2 se active y empiece a enviar sus propios mensajes *ADVERTISE*.

Debido a que pueden coexistir varios gateways activos en la red, los clientes deben recibir sus mensajes *ADVERTISE* repetidamente cada cierto tiempo TADV. Si los clientes dejan de recibir mensajes cierto número de veces (NADV veces) de cierto gateway, eliminará su información de todas las listas.

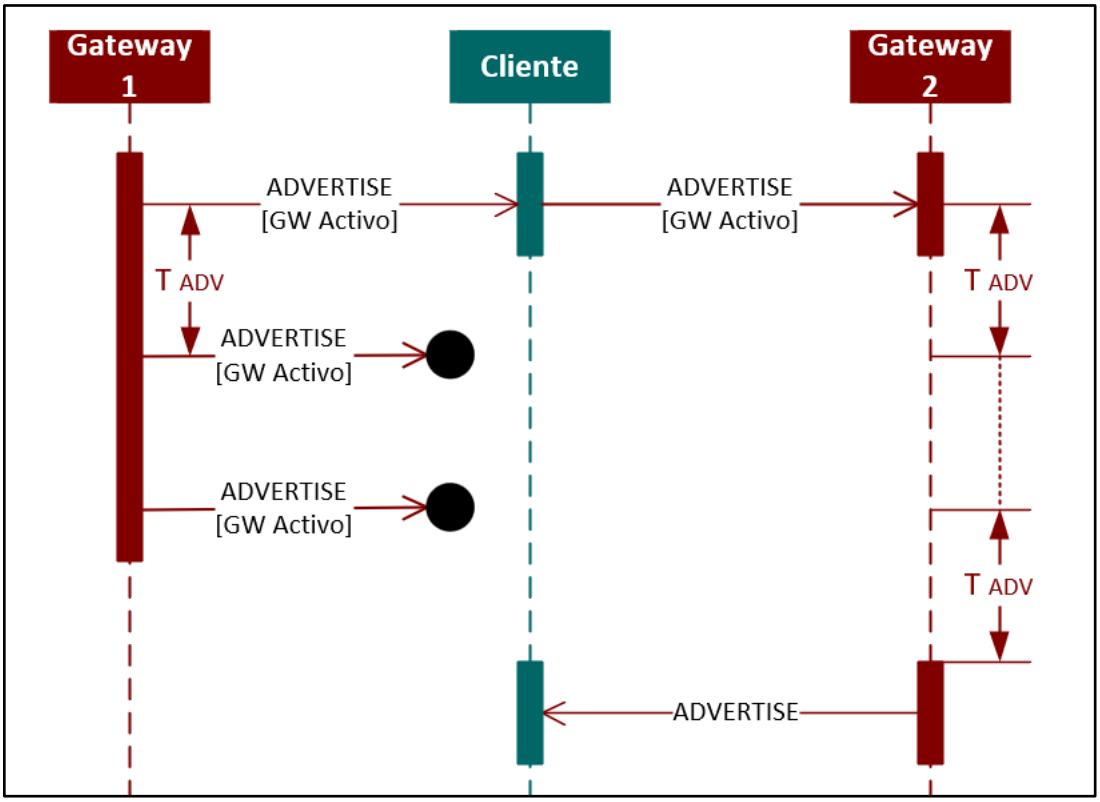


Figura 2.3 Funcionamiento de un Gateway activo e inactivo.

2.2.2 CONFIGURACIÓN PARA CONEXIÓN DEL CLIENTE

Para que un cliente pueda intercambiar información con un gateway, previamente tienen que establecer conexión. Durante este procedimiento un cliente debe enviar un mensaje *CONNECT* hacia el gateway al que desee conectarse. El mensaje le indica al gateway, que el cliente requiere o no enviar datos will mediante la bandera will.

Si el cliente requiere enviar un tema y un mensaje de última voluntad, establecerá en 1 la bandera will. Después de que el gateway reciba el mensaje *CONNECT* solicita el tema y mensaje will mediante los mensajes *WILLTOPICREQ* y *WILLMSGREQ*. El cliente enviará los datos will respondiendo con los mensajes *WILLTOPIC* y *WILLMSG* respectivamente. Finalmente, el gateway enviará un mensaje *CONNACK* aceptando o rechazando la conexión.

Si el cliente establece en 0 la bandera will del mensaje *CONNECT*, el gateway responde de inmediato con un mensaje *CONNACK*. En el diagrama de la Figura 2.4, se muestra dos clientes estableciendo conexión con un gateway con un valor diferente de bandera will.

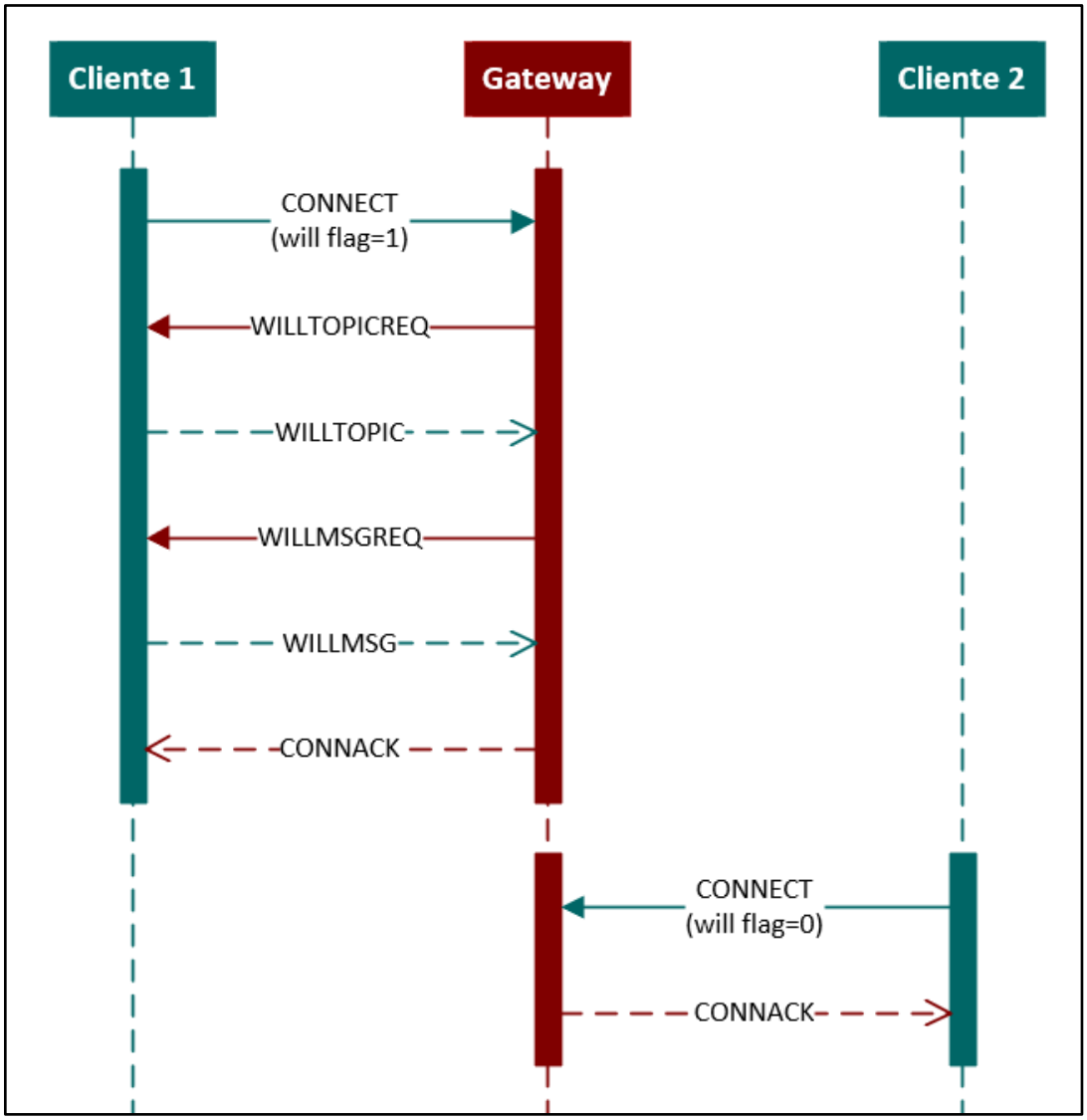


Figura 2.4 Gateway recibiendo conexión de dos clientes.

2.2.3 PROCEDIMIENTO PARA ACTUALIZAR DATOS WILL

Después de establecer una conexión MQTT-SN, los mensajes de última voluntad almacenados pueden ser actualizados en cualquier momento por el cliente. El tema will puede actualizarse enviando un mensaje WILTOPICUPD mientras que el mensaje will puede actualizarse mediante un mensaje *WILLMSG*UPD, ninguno de estos dos mensajes depende del otro. En respuesta a los mensajes de actualización el gateway envía *WILLTOPIC*RESP o *WILLMSG*RESP como se aprecia en el diagrama de la Figura 2.5. Además, si es necesario eliminar el tema y mensaje will de un cliente, este último puede enviar un mensaje *WILLTOPIC*UPD vacío.

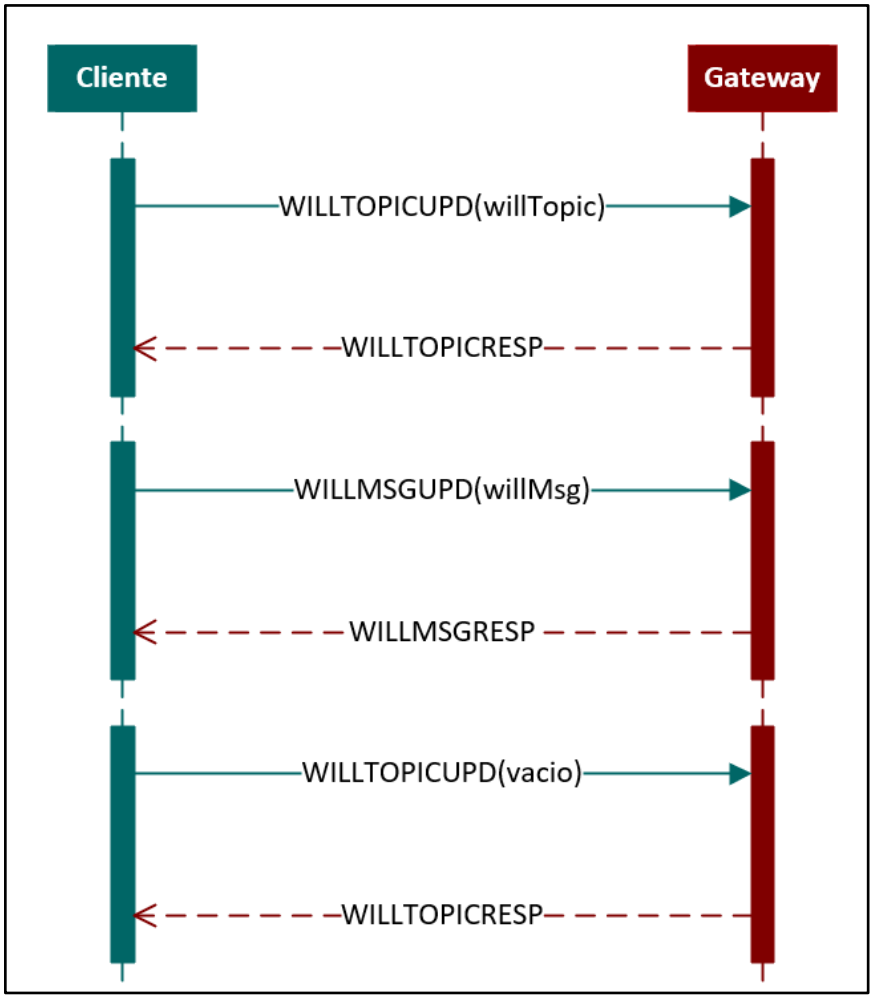


Figura 2.5 Actualización de datos will.

2.2.4 PROCEDIMIENTO PARA REGISTRAR NOMBRES DE TEMAS

Con el fin de reducir el tamaño de los mensajes de publicación, MQTT-SN agrega un procedimiento para registrar nombres de tema. El procedimiento permite que el cliente y el gateway informen a su par acerca de un identificador de tema. El identificador de tema debería ser más corto que un nombre de tema y estará ubicado en el campo TOPICID.

Si el cliente requiere registrar un nombre de tema, primero envía un mensaje *REGISTER*. El gateway recibe el mensaje *REGISTER*, analiza el nombre de tema, le asigna un identificador y envía el identificador de tema mediante un mensaje *REGACK*. Si el registro es rechazado también se envía un *REGACK*, con el campo returnCode indicando el motivo del rechazo, como se muestra el diagrama de la Figura 2.6.

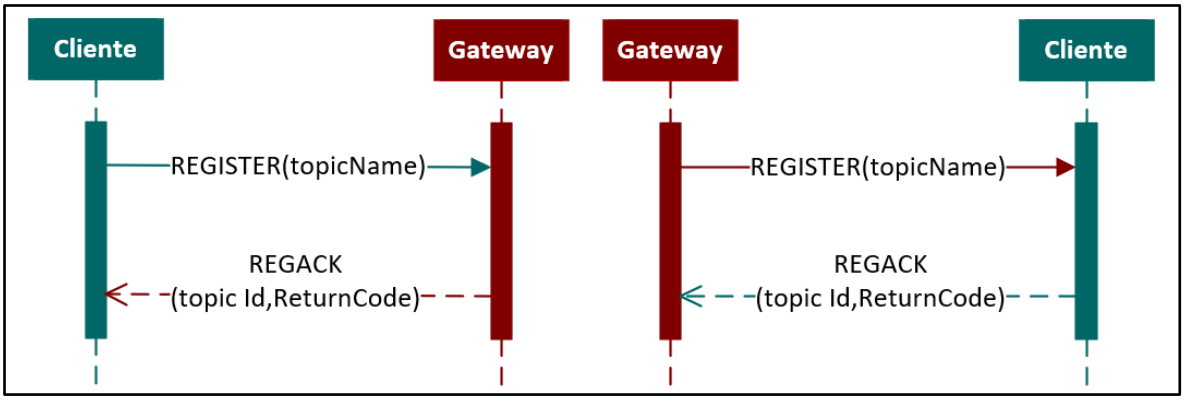


Figura 2.6 Procedimiento para registrar un nombre de un tema, ejecutado por un cliente y por un gateway.

En caso de que el cliente pierda su conexión y después se conecte sin haber configurado la bandera CleanSession, puede que necesite información de los nombres e identificadores de tema a los que se registró o subscribió previamente. El gateway avisa al cliente acerca del nombre de tema y su correspondiente identificador dentro de un mensaje *REGISTER*. El cliente utiliza el identificador de tema recibido para enviar sus mensajes de publicación. El Gateway también utiliza el procedimiento de registro cuando el cliente se ha subscrito a nombres de temas que incluyen caracteres comodines. Esto debido a que el mensaje que acepta una subscripción no puede llevar más de un identificador de tema.

En caso de existir un rechazo por congestión se reiniciará el procedimiento después de esperar un tiempo TWAIT, ya sea por el cliente o el gateway.

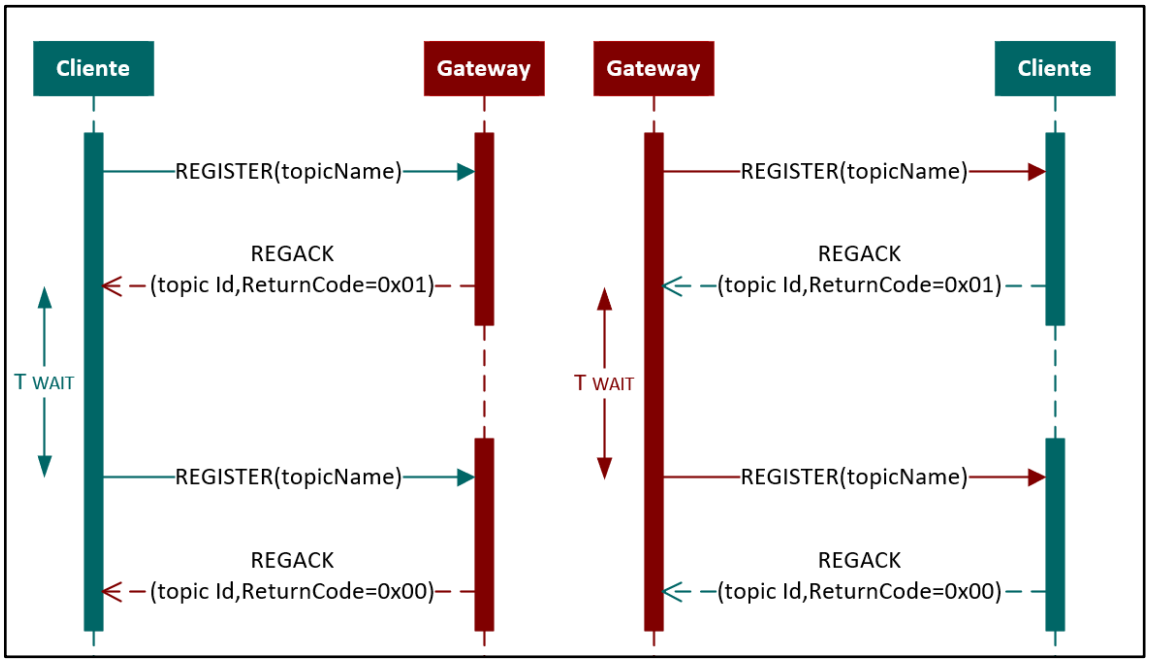


Figura 2.7 Rechazo por congestión durante un procedimiento de registro**.**

2.2.5 PROCEDIMIENTO DE PUBLICACIÓN(CLIENTE)

Un cliente puede publicar información relacionada con un nombre de tema, si previamente a completado el procedimiento de registro. La carga útil, junto a su identificador de tema, es enviada dentro de un mensaje *PUBLISH*.

Este procedimiento cubre tres de los cuatros niveles de calidad de servicio (QoS 0, QoS 1 y QoS 2) soportados por MQTT-SN, por lo tanto, un gateway responderá de diferente forma ante la recepción de una publicación. A continuación, se describe el comportamiento del gateway ante los tres niveles de QoS mencionados:

- Si el gateway recibe una publicación con QoS 0 no envía ningún tipo de mensaje.
- Si el gateway recibe una publicación con QoS 1 responde con un mensaje PUBAC.
- Si el gateway recibe una publicación con QoS 2 responde con un mensaje *PUBREC*, el cliente recibe este mensaje y responde con un mensaje *PUBREL*. Finalmente, el gateway envía el mensaje *PUBCOMP* en respuesta al último mensaje del cliente.

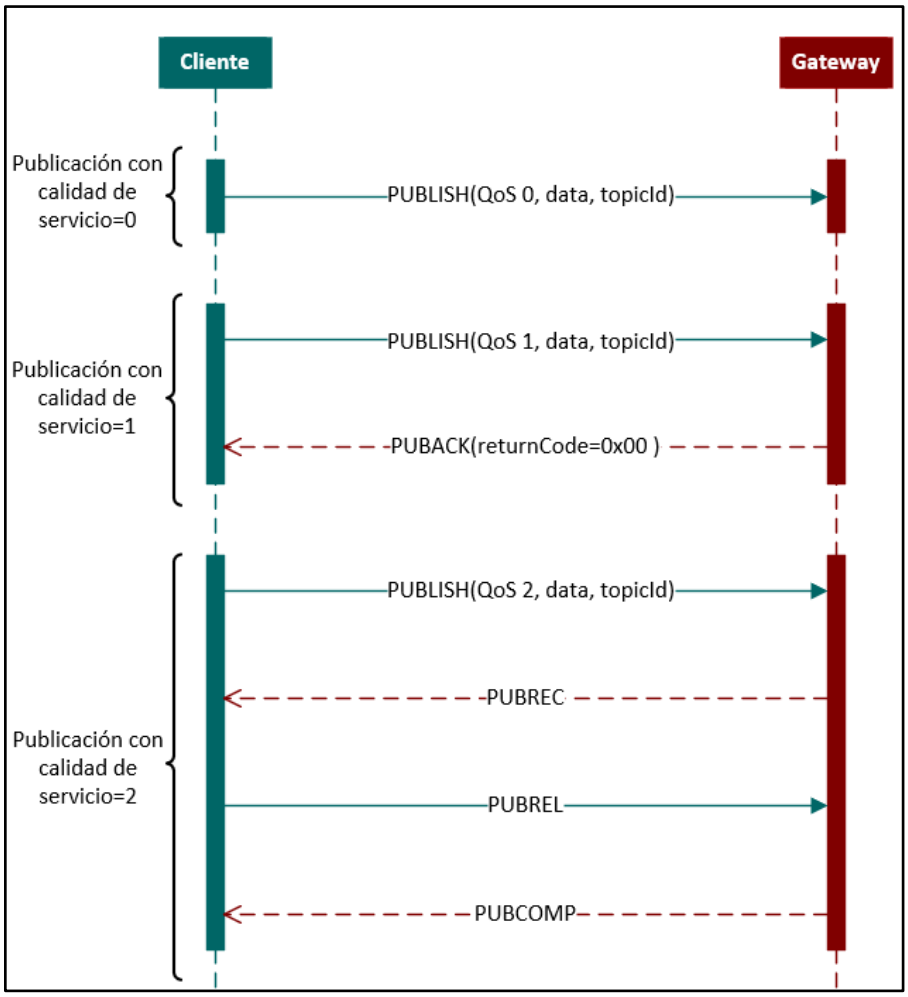


Figura 2.8 Cliente enviando mensajes *PUBLISH* (QoS 0, QoS 1 y QoS 2) a un gateway.

También hay que mencionar que, si un cliente realiza una publicación con calidad de servicio 1 o 2, tendrá que esperar a que el intercambio de mensajes finalice antes de iniciar una nueva publicación.

El diagrama de secuencia de la Figura 2.8 muestra a un cliente enviando mensajes de publicación con los diferentes niveles de calidad de servicio mencionados en este procedimiento.

Finalmente, se debe indicar que un gateway puede responder con un mensaje *PUBACK* sin importar el nivel de QoS, cuando la publicación es rechazada. El campo returnCode indica el motivo del rechazo. En caso de existir un rechazo por un identificador de tema inválido (returnCode=0x02), el cliente tiene que registrar de nuevo el nombre de tema.

En publicaciones con QoS 1 y QoS 2, si existe un rechazo por congestión (returnCode =0x01), el cliente intentará publicar después de esperar un tiempo *TWAIT*, tal como se muestra en los diagramas de la Figura 2.9.

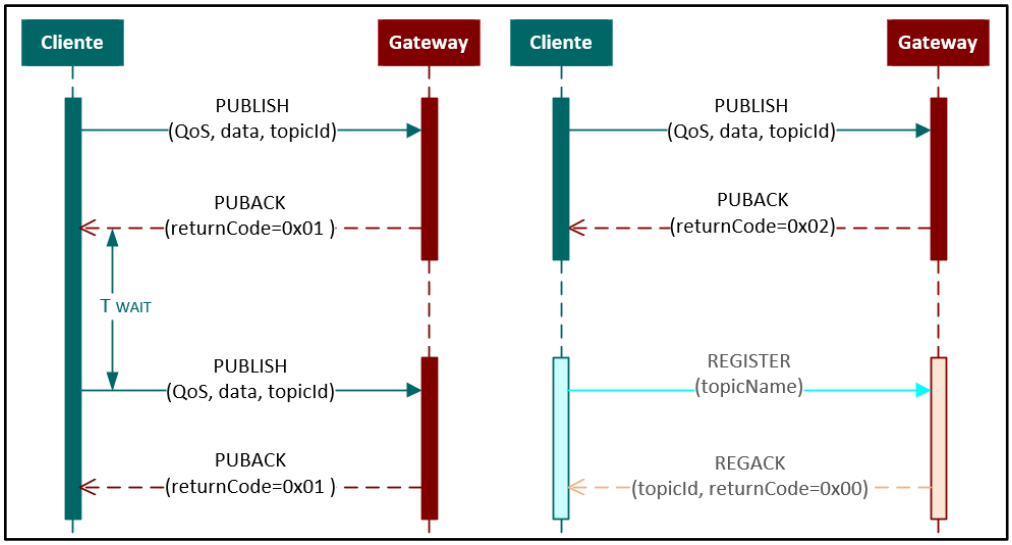


Figura 2.9 Intercambio de mensajes entre cliente y gateway cuando existen errores en las publicaciones enviadas por el cliente.

2.2.6 PROCEDIMIENTO DE PUBLICACIÓN (QOS-1)

El procedimiento de publicación con calidad de servicio -1 es útil para clientes MQTT-SN simples los cuales solo manejan este procedimiento. No será necesario realizar procesos de conexión, registro, subscripción o descubrimiento; el cliente debe conocer la dirección del gateway previamente. El cliente envía mensajes *PUBLISH* al gateway sin importar el estado en que se encuentre, tampoco verifica si el mensaje llega o no Figura 2.10.

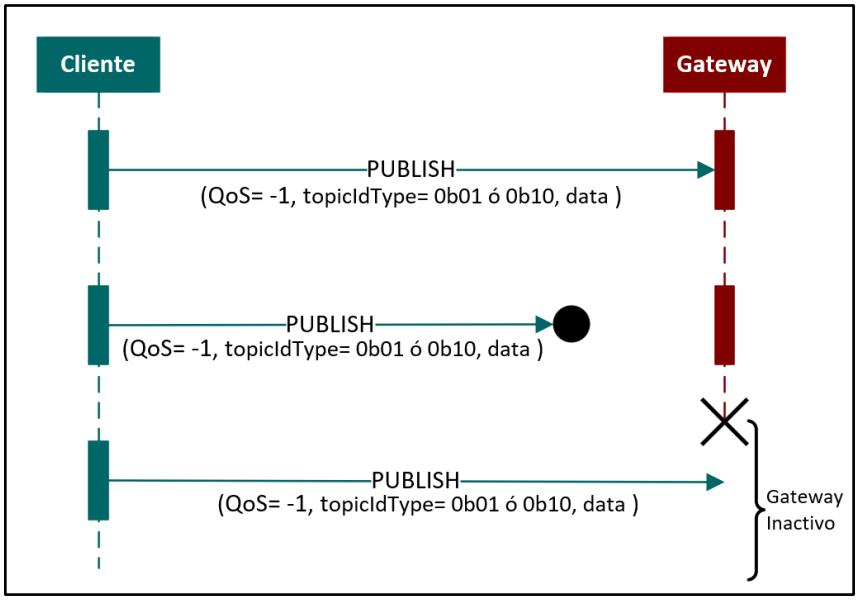


Figura 2.10 Cliente enviando mensajes *PUBLISH* (QoS -1) al gateway.

Los mensajes *PUBLISH* solo puede tener los siguientes valores:

- QoS flag: "0b11" para nivel de calidad de servicio -1.
- TopicIdType flag: "0b01" para un identificador de tema predefinido o "0b10" para un nombre de tema corto.
- Campo TopicId: Valor de un identificador de tema predefinido o nombre de tema corto.
- Campo de datos: Valor de la carga útil del mensaje.

2.2.7 PROCEDIMIENTO PARA SUBSCRIBIRSE O DARSE DE BAJA DE UN TEMA

Para subscribirse a un nombre de tema, un cliente envía un mensaje *SUBSCRIBE* a la puerta de enlace indicando el tema en el que está interesado. Si el gateway puede aceptar la subscripción, asigna un ID de tema al nombre del tema recibido y lo devuelve dentro de un mensaje *SUBACK* al cliente,tal como se muestra en el diagrama de la Figura 2.11.

Si el cliente se *SUBSCRIBE* a un nombre de tema que contiene un caracter comodín (+ ó #), el mensaje *SUBACK* que regresa contendrá el valor de identificación del tema 0x0000. Cuando un cliente se *SUBSCRIBE* a varios temas utilizando caracteres comodines, se informará acerca del identificador de todos los temas involucrados, antes de enviar el primer mensaje de publicación de un tema en específico. Para informar al cliente sobre el valor de identificador de tema, el gateway utiliza el procedimiento de registro.

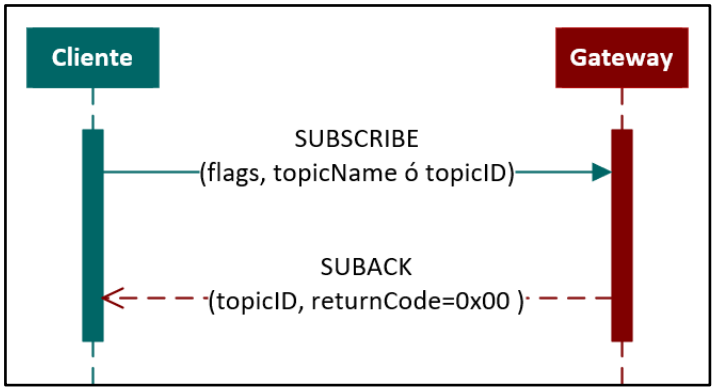


Figura 2.11 Subscripción del cliente.

De manera similar al procedimiento de publicación, los identificadores de temas pueden estar predefinidos para ciertos nombres de temas, también se puede tener nombres de temas cortos. En los dos casos, el cliente debe subscribirse estos identificadores o nombres de temas si requiere recibir publicaciones relacionadas a estos dos últimos.

En caso de que no se pueda aceptar la subscripción, también se devuelve un mensaje *SUBACK* al cliente con la causa del rechazo codificada en el campo returnCode. Si la causa del rechazo es "rechazado: congestión", el cliente debe esperar el tiempo TWAIT antes de reenviar el mensaje *SUBSCRIBE* al Gateway (Figura 2.12).

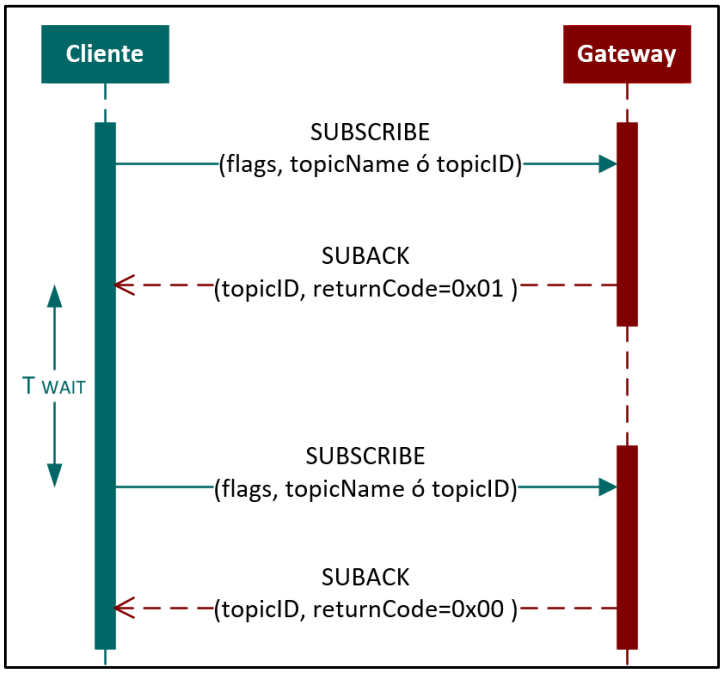


Figura 2.12 Interacción entre cliente y gateway cuando no es aceptada la subscripción por congestión.

Para darse de baja, un cliente envía un mensaje *UNSUBSCRIBE* al gateway, este último debe responder con un mensaje *UNSUBACK*. El cliente indica el nombre o identificador de tema del que desea darse de baja, tal como se muestra en el diagrama de la Figura 2.13.

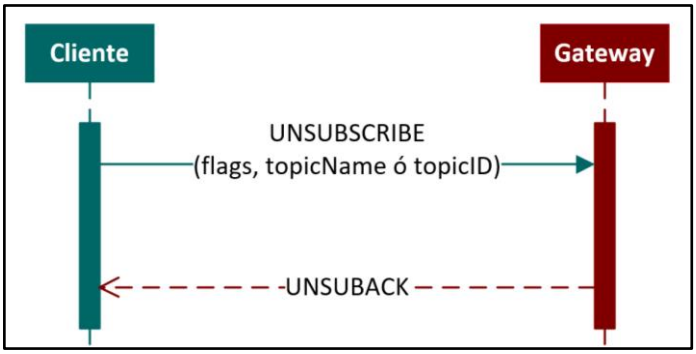


Figura 2.13 Diagrama de secuencia subscripción del cliente.

2.2.8 PROCEDIMIENTO DE PUBLICACIÓN(GATEWAY)

Para que un gateway pueda realizar una publicación de un tema en específico, primero debe tener un cliente subscrito a ese tema. Las publicaciones utilizan el identificador de tema y la calidad de servicio establecidos durante la subscripción. Al igual que en una publicación del cliente, un gateway envía un mensaje *PUBLISH* que dependiendo del nivel de calidad de servicio se puede obtener (QoS 1 y QoS 2) o no (QoS 0) una respuesta del cliente. Los diagramas de secuencia de la Figura 2.14, muestran varias publicaciones realizadas por el gateway.

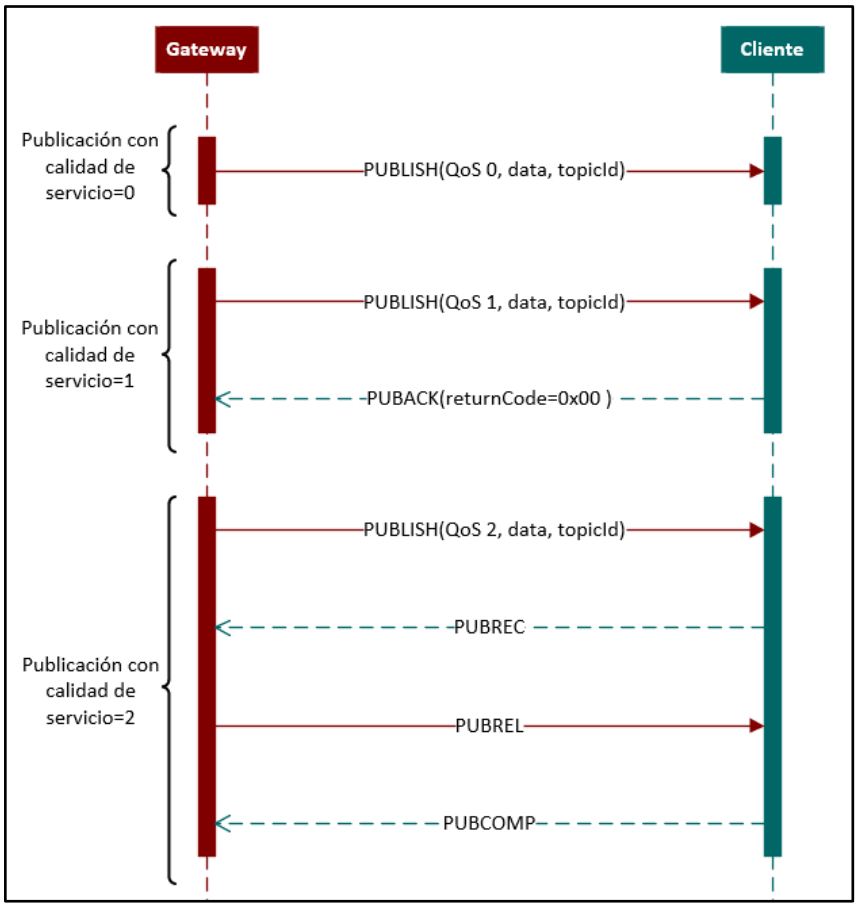


Figura 2.14 Gateway enviando mensajes *PUBLISH* (QoS 0, QoS 1 y QoS 2) a un cliente.

Las publicaciones pueden ser rechazadas mediante mensajes *PUBACK* similar a lo indicado el literal 2.2.5. En caso de tener identificadores de tema inválidos, el gateway intentará corregirlos, eliminarlos, o abortará el procedimiento de publicación sin ninguna advertencia, tal como se muestra en el diagrama de la Figura 2.15.

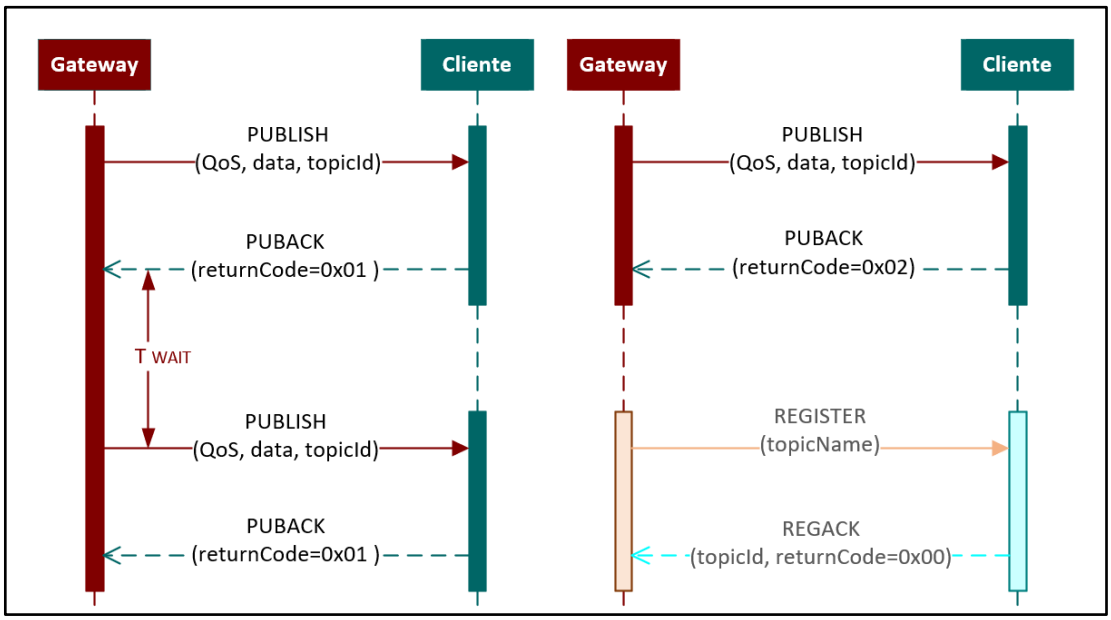


Figura 2.15 Intercambio de mensajes entre cliente y gateway cuando existen errores en las publicaciones enviadas por el gateway.

2.2.9 PROCEDIMIENTO KEEP ALIVE Y PING

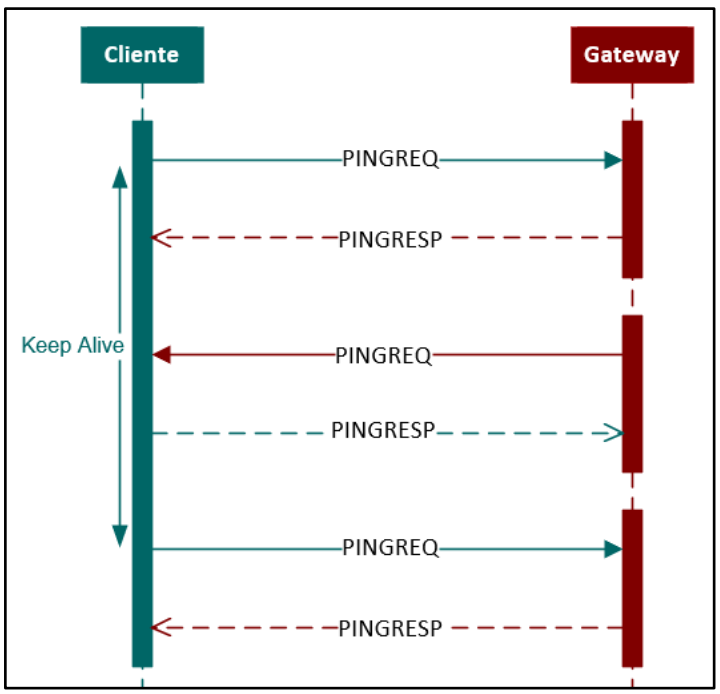


Figura 2.16 Procedimiento Keep Alive y PING.

En una red MQTT-SN los clientes manejan este procedimiento para verificar el correcto funcionamiento del gateway. El cliente debe enviar un mensaje *PINGREQ* dentro de cada período de tiempo Keep Alive, indicado por el cliente durante la conexión, que el gateway reconoce el mensaje recibido con un *PINGRESP*. De manera similar, un cliente responderá con un mensaje *PINGRESP* si recibe un mensaje *PINGREQ* del gateway al que está conectado. El envío y recepción de los mensajes mencionados está indicado en el diagrama de secuencia mostrado en la Figura 2.16.

2.2.10 PROCEDIMIENTO PARA DESCONEXIÓN DEL CLIENTE

Sin un cliente desea desconectarse de la red MQTT-SN, debe enviar un mensaje *DISCONNECT*, a continuación, el gateway envía otro mensaje *DISCONNECT* para confirmar la desconexión (Figura 2.17). En este tipo de desconexión no es necesario iniciar un procedimiento de registro, si el cliente requiere conectarse nuevamente a la red. Cuando un cliente se desconecta, los datos will y las subscripciones de este persisten si no se ha indicado lo contrario en la bandera cleanSession durante una nueva conexión.

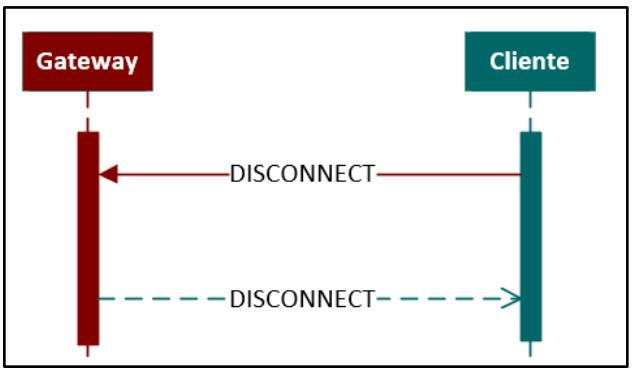


Figura 2.17 Procedimiento para desconexión del cliente.

El gateway envía un mensaje *DISCONNECT* cuando un cliente no puede ser reconocido, el cliente que reciba este mensaje tiene que establecer conexión nuevamente.

2.2.11 PROCEDIMIENTO DE RETRANSMISIÓN DEL CLIENTE

Este procedimiento es utilizado por clientes cuando envían mensajes unicast (los cuales requieren un mensaje de respuesta) al gateway. Cuando el mensaje es enviado y no recibe respuesta se inicia un temporizador de reintentos (Tretry) y un contador de reintentos (Nretry). Si durante el tiempo que dura el temporizador no se recibe respuesta, el cliente retransmite el mensaje unicast, el temporizador es reiniciado y el contador aumenta en uno. Si la respuesta esperada se recibe se suspende el temporizador y el contador, como se indica en el diagrama de secuencia mostrado en la Figura 2.18.

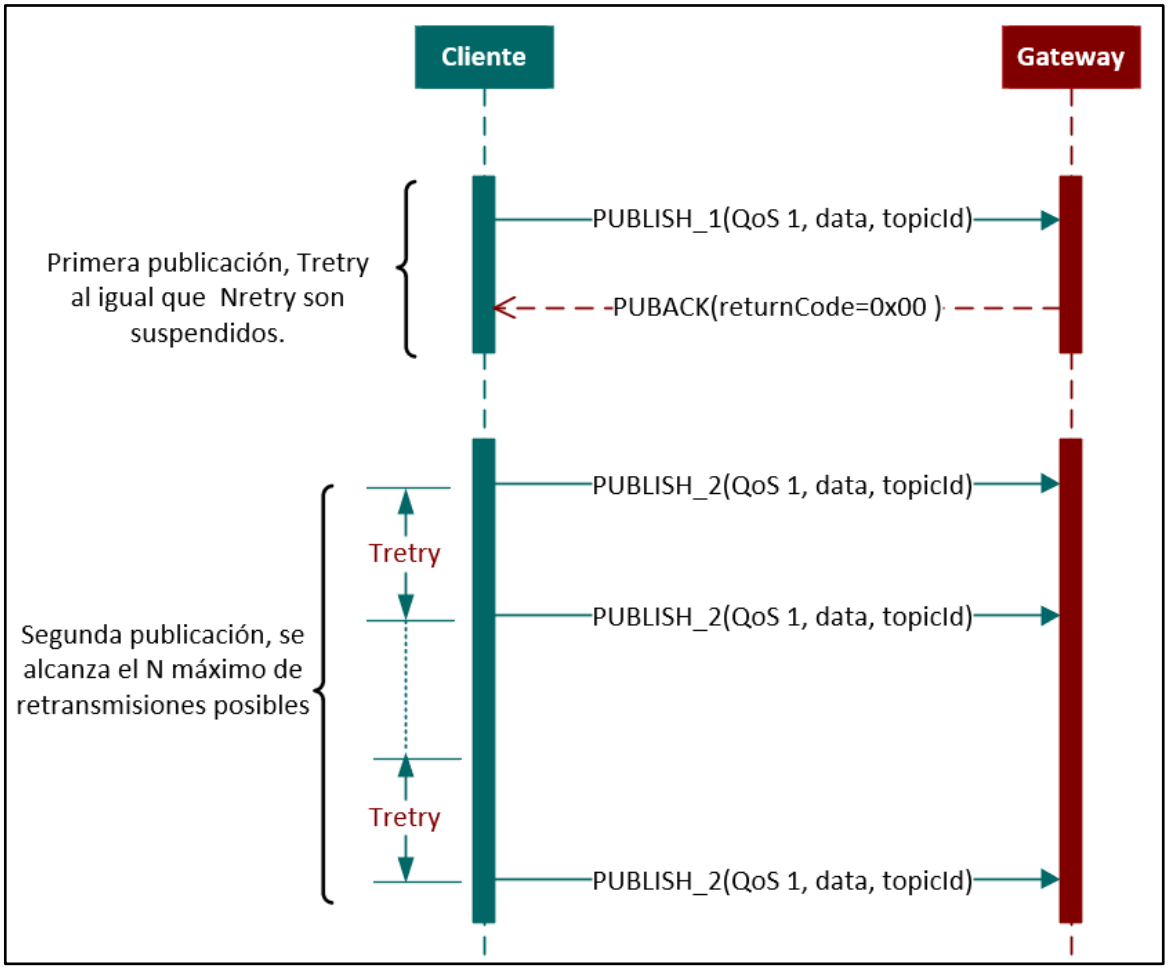


Figura 2.18 Cliente retransmitiendo un mensaje *PUBLISH* (QoS 1).

Existe un máximo de retransmisiones que es posible realizar, después de superarlas el cliente aborta el procedimiento, se asume que la conexión MQTT-SN se ha perdido. A continuación, se debe iniciar una nueva conexión ya sea con el mismo o con otro gateway.

2.2.12 SOPORTE PARA AHORRO DE ENERGÍA

Este procedimiento es utilizado para dar soporte a clientes que requieren ahorrar energía por lo cual deben permanecer dormidos. Los clientes solo se despiertan en el momento que tengan información para enviar o recibir. El gateway almacena en un buffer los mensajes que el cliente no puede recibir, hasta que este último despierte.

Desde la perspectiva de un gateway, un cliente puede estar en uno de los siguientes estados: activo, dormido, despierto, perdido y desconectado [28]. Todos los estados excepto el desconectado son supervisados constantemente.

Un cliente pasa a estado activo cuando se ha establecido correctamente una conexión. Este estado es supervisado periódicamente por gateway mediante el prendimiento Keep

Alive y PING. Si el temporizador Keep Alive finaliza sin que se reciba mensaje alguno el gateway considera que el cliente está perdido.

Cuando un cliente requiere pasar a estado dormido tiene que enviar un mensaje *DISCONNECT*, el cual contiene tiempo de duración del temporizador Tsleep que dicho cliente posee. El mensaje es reconocido con otro mensaje *DISCONNECT* para a continuación iniciar el temporizador. El gateway monitorea el estado del cliente mediante su propio temporizador Tsleep cuya duración se indica en el mensaje *DISCONNECT* enviado por el cliente. Si el cliente no envía ningún mensaje antes de que el temporizador Tsleep del Gateway finalice, se lo considera como cliente en estado perdido.

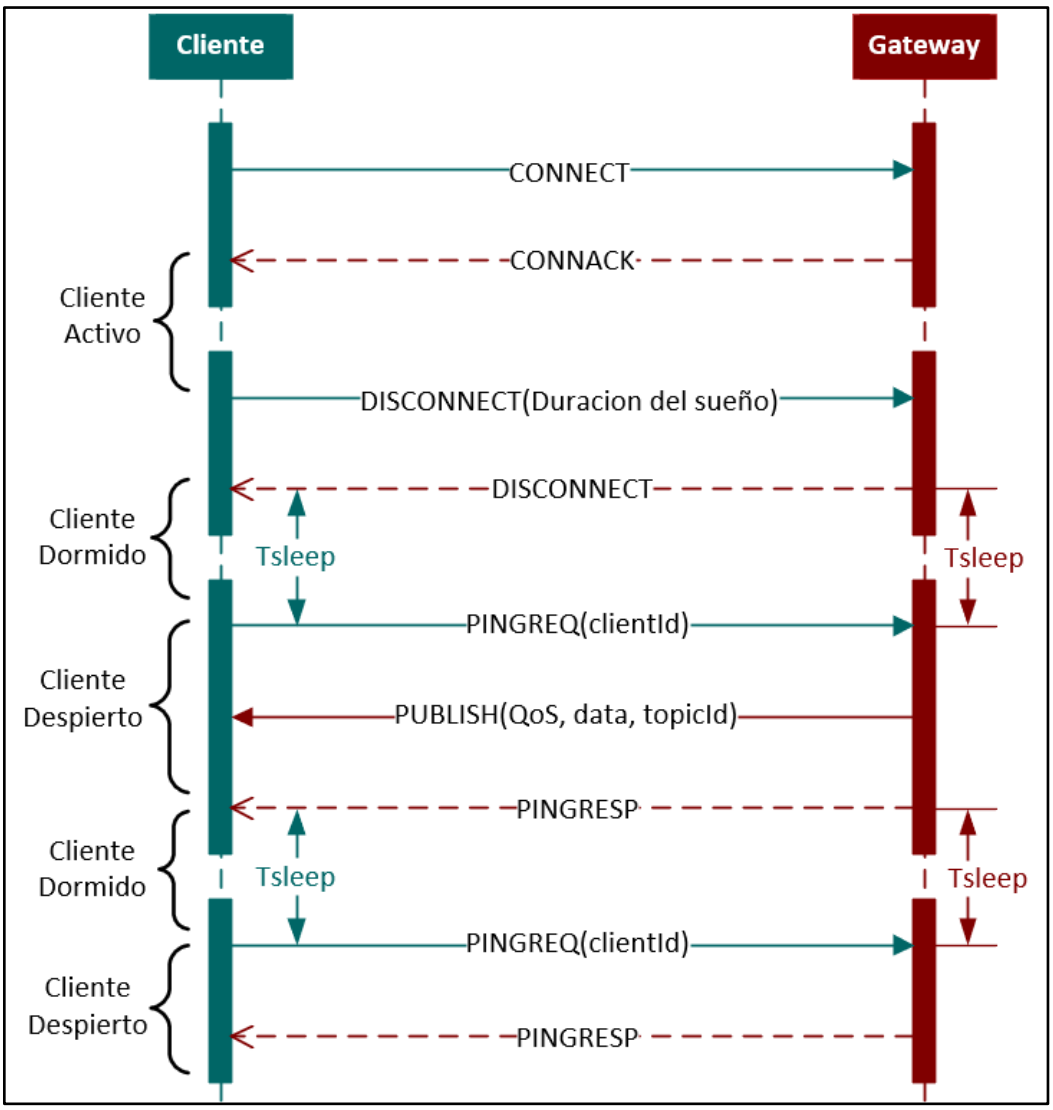


Figura 2.19 Cliente que requiere ahorrar energía.

Si el gateway recibe un *PINGREQ* antes de que su temporizador Tsleep termine, entonces identifica al cliente que lo envío y considera que ha despertado. Por parte del cliente, pasa a estado despierto cuando envía el mensaje *PINGREQ* con su respectivo identificador, una

vez que termina su Tsleep. Durante el estado que el cliente permanezca despierto, ocurre una transferencia de mensajes pendientes, la cual el gateway finaliza con el envío de un mensaje *PINGRESP*. De no existir mensajes pendientes inmediatamente el gateway envía el mensaje *PINGRESP*. Tras la recepción del mensaje *PINGRESP* el cliente reinicia su temporizador Tsleep y regresa al estado dormido. Simultáneamente el gateway considera que el cliente se encuentra dormido tras el envío del *PINGRESP* y también reinicia su temporizador. Como se muestra en el diagrama de secuencia de la Figura 2.19.

El estado desconectado no es supervisado por el gateway, un cliente llegara a este estado tras enviar un mensaje *DISCONNECT* y recibir su respectiva confirmación, tal como se indica en el procedimiento de desconexión explicado en el literal 2.2.10.

Para controlar la transición de los mensajes *PINGREQ* y *PINGRESP* el cliente hará uso del procedimiento de retransmisión explicado en el literal 2.2.11.

2.3 MENSAJES MQTT-SN A INTERCAMBIAR ENTRE EL CLIENTE Y EL GATEWAY

Después de realizar los diagramas de secuencia se determina que para desarrollar las MEF del protocolo MQTT-SN habrá que utilizar todos los mensajes de la especificación MQTT-SN, excepto la trama de encapsulación MQTT-SN. Los mensajes deben ser enviados y recibidos de acuerdo al procedimiento que se esté ejecutado en el dispositivo que los utilice. A continuación, se indica una breve descripción de cada uno de los mensajes.

2.3.1 MENSAJES MQTT-SN

- ***ADVERTISE*:** Mensaje enviado por el gateway, para anunciar su presencia dentro de una red. Es enviado periódicamente según lo indique el temporizador TADV.
- ***SEARCHGW*:** Mensaje enviado por un cliente cuando este último no quiere esperar a los mensajes *ADVERTISE*.
- ***GWINFO*:** Este mensaje es enviado por un gateway o un cliente en respuesta a un mensaje *SEARCHGW*.
- ***CONNECT*:** Este mensaje lo envía un cliente cuando desear establecer conexión.
- ***CONNACK*:** Este mensaje es enviado por el gateway para confirmar una conexión solicitada por un nodo cliente.
- ***WILLTOPICREQ*:** Mensaje enviado por el gateway cuando este recibe un mensaje *CONNECT,* con la bandera will configurada con un valor de uno.
- ***WILLTOPIC*:** Mensaje enviado por un nodo cliente en respuesta al mensaje *WILLTOPICREQ*, contiene el tema de última voluntad.

- ***WILLMSGREQ*:** Mensaje enviado por el gateway después de recibir el mensaje *WILTOPIC*.
- ***WILLMSG*:** Mensaje enviado por un nodo cliente en respuesta al mensaje *WILLMSGREQ*, contiene el mensaje de última voluntad.
- ***REGISTER*:** Este mensaje es enviado por un cliente para solicitar un identificador de tema a un gateway. El gateway también puede enviar el mensaje cuando requiere informar al cliente sobre el valor del identificador de tema que se ha asignado previamente.
- ***REGACK*:** mensaje enviado por un cliente o un gateway en respuesta a un mensaje *REGISTER.*
- ***PUBLISH*:** mensaje enviado por un cliente o un gateway cuando deseen publicar datos de un determinado tema.
- ***PUBACK***: mensaje enviado por un cliente o un gateway en respuesta a un mensaje *PUBLISH con QoS 1,* si existen problemas de congestión también se envía este mensaje cuando se recibe un mensaje *PUBLISH* con QoS 1 y QoS 2. Asimismo, es enviado cuando no se reconoce un identificador de tema independientemente del nivel de QoS.
- ***PUBREC*:** mensaje es enviado por un gateway en respuesta a un mensaje *PUBLISH* con QoS 2. Cuando este mensaje es enviado se cancela el procedimiento de retransmisión.
- ***PUBREL*:** mensaje es enviado por un cliente en respuesta a un mensaje *PUBREC*.
- ***PUBCOMP*:** mensaje enviado por un gateway en respuesta a un mensaje *PUBREL*.
- ***SUBSCRIBE*:** este mensaje es utilizado para subscribirse a un tema al que el cliente se encuentre interesado.
- ***SUBACK*:** mensaje enviado por un gateway en respuesta a un mensaje *SUBSCRIBE.*
- ***UNSUBSCRIBE*:** este mensaje es enviado por el cliente para darse de baja de algún tema al que se encontraba subscrito.
- ***UNSUBACK*:** mensaje enviado por un gateway en respuesta a un mensaje *UNSUBSCRIBE.*
- ***PINGREQ*:** este mensaje es utilizado para comprobar si un cliente se encuentra activo dentro de la red inalámbrica. Además, el cliente puede utilizar este mensaje para indicarle al gateway que ha pasado de estado dormido a despierto.
- ***PINGRESP***: este mensaje es enviado en respuesta a un mensaje *PINGREQ* para indicar que el cliente se encuentra activo. Además, un mensaje *PINGRESP* es enviado por un gateway para que el cliente regrese al estado dormido.

- ***DISCONNECT*:** este mensaje es utilizado por un cliente cuando quiere desconectarse o cuando quiere pasar a estado dormido. El gateway debe confirmar la desconexión o el cambio de estado con otro mensaje *DISCONNECT.*

Los mensajes son considerados como señales de entrada y salida para las máquinas de estados finitos desarrolladas. Tanto el nodo cliente como el nodo gateway tienen la capacidad de generar sus respectivos mensajes MQTT-SN. Además, como se puede apreciar en los diagramas de secuencia un dispositivo genera un mensaje cuando recibe un mensaje enviado por otro dispositivo.

2.4 CONSIDERACIONES PREVIAS AL DESARROLLO DE LA MÁQUINA DE ESTADOS

Para el desarrollo de máquina de estados finitos del protocolo MQTT-SN, se considera que solo dos nodos tendrán la capacidad de manejar los estados de cada procedimiento MQTT-SN. Los nodos mencionados deben estar en el borde de la red, de esta manera se forma una red con topología lineal como se muestra en la Figura 2.20. Los nodos intermedios solo se encargan de verificar si el tipo de mensaje MQTT-SN es el correcto, para a continuación enviarlo al siguiente nodo hasta que llegue a su destino.

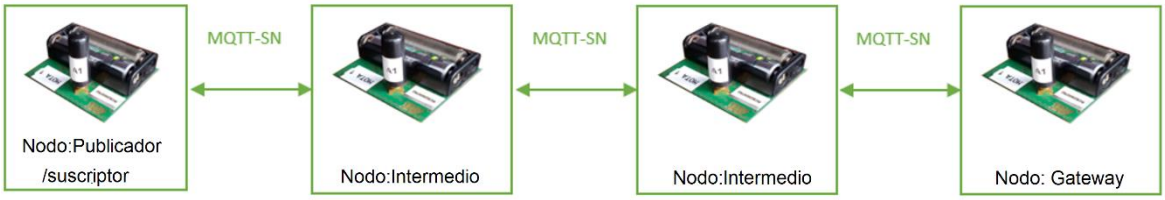


Figura 2.20 Red IEEE 802.15.4.

Ya que los nodos RCB256RFR2 tienen sus propias características de funcionamiento, se debe determinar si alguna de ellas afecta al desarrollo de la MEF. En particular se estudia el funcionamiento de los nodos en topologías lineales, el servicio de recepción y transferencia de datos y el lenguaje de programación utilizado por el Nodo. Además, se tiene en cuenta el funcionamiento del broker.

2.4.1 FUNCIONAMIENTO DE LOS NODOS SOBRE TOPOLOGÍAS LINEALES

El protocolo MQTT-SN al estar diseñado para redes inalámbricas de sensores permite que opere directamente sobre la capa de enlace [1]. Además, sobre una topología lineal no se requiere de una capa de red ya que solo existe una única ruta por donde se envía la información.

En la topología lineal propuesta solo los nodos de borde tienen la capacidad de ejecutar los procedimientos MQTT-SN. El nodo RCB256RFR2 permite acceder directamente al campo carga útil de una trama IEEE 802.15.4 [1]. Por lo cual, el nodo gateway y el nodo publicador/subscriptor(cliente) tienen la capacidad de generar mensajes MQTT-SN y enviarlos a través de los nodos intermedios.

Ya que los nodos trabajan sobre una topología lineal se omiten funciones en el procedimiento de anuncio y descubrimiento.

No se tienen en cuenta la capacidad de responder a mensajes *SEARCHGW* por parte de nodos que no sean el gateway, debido a que en la topología de red propuesta solo existe un nodo publicador/subscriptor. Tampoco se toman en cuenta los temporizadores con los cuales se da prioridad de respuesta al gateway y, no se omite todo el procedimiento ya que el envío constante de mensajes *ADVERTISE* permite conocer el estado del gateway al cliente.

Finalmente, se debe mencionar que al trabajar sobre la topología lineal propuesta hay que asumir que no debe existir problemas por transmisión o recepción de mensajes simultáneos.

2.4.2 SERVICIO DE TRANSFERENCIA Y RECEPCIÓN DE DATOS DEL NODO RCB256RFR2

Como se especifica en las características del nodo RCB256RFR2, éste no puede recibir y enviar información al mismo tiempo, por lo que se considera que la recepción de un mensaje debe ser el detonante para un cambio de estado. En respuesta a la recepción de un mensaje el nodo puede enviar otro mensaje según el procedimiento y estado en el que se encuentre.

Cuando el nodo se encuentra en un determinado estado puede recibir diferentes tipos de mensajes, pero solo un mensaje a la vez. Después de la recepción y transmisión, de ser el caso, el nodo debe pasar a otro estado o regresar al mismo estado. A continuación, se describe cómo funciona la transmisión de recepción de tramas IEEE-802.15.4 mediante un diagrama de estados.

2.4.2.1 Estados de operación del transceptor RCB256RFR2

Los modos o estados de funcionamiento principales del transceptor están diseñados para trabajar con aplicaciones del estándar IEEE 802.15.4 en el rango de 2,4 GHz. Cuando se enciende un dispositivo, pasa por una secuencia de inicio del sistema antes de entrar en el estado de espera. Si se procesa o recibe una instrucción o una señal, puede entrar en uno

de los siguientes estados: estado de transmisión, estado de recepción, estado de reposo, o reinicio [13].

El transceptor RCB256RFR2 utiliza un servicio de transferencia y recepción de información half dúplex. Por esta razón, antes de poder recibir una trama se debe esperar a que una transmisión que el nodo tenga pendiente se complete. Cuando el transceptor está a punto de transmitir o recibir, pasan al estado de transmisión o el estado de recepción, así como el estado de transmisión ocupada o el estado de recepción ocupada, como se muestra en la Figura 2.21. El transceptor permanece en cualquiera de estos estados hasta que haya completado la transmisión o recepción [16].

La operación de los estados sleep, reinicio, así como los demás estados se discutirá en los siguientes párrafos.

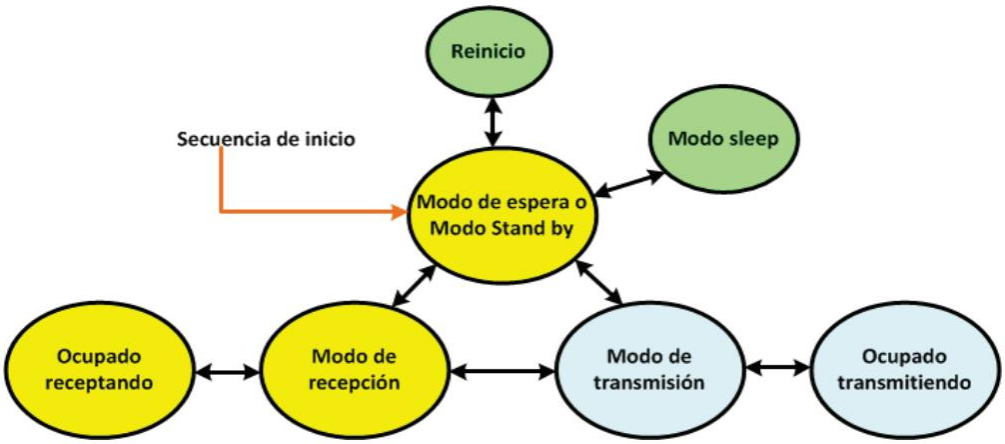


Figura 2.21 Diagrama de estados del transceptor RCB256RFR2 [16].

2.4.2.1.1 Estado espera o stand by

Una vez encendido el transceptor, el oscilador de cristal comienza a trabajar en la sincronización del transceptor, mientras que el microcontrolador espera una nueva instrucción que le indique si debe activar el transmisor o el receptor. Durante este momento, es posible acceder a la memoria, el búfer y los registros de control del transceptor, sin embargo, esto hace que el transceptor se deshabilite, por lo que no recibe ni transmite señales [16].

2.4.2.1.2 Estado transmisión y ocupado transmitiendo

Cuando se requiere transmitir datos, el transceptor pasara del estado de espera al estado de transmisión, lo que activa el sintetizador PLL, además de sintonizar la frecuencia y el canal para la transmisión. A continuación, el transceptor ingresa al estado ocupado transmitiendo para enviar los datos y luego regresa al estado de transmisión [16].

2.4.2.1.3 Estado recepción y estado ocupando recibiendo

El transceptor pasa a estado de recepción cuando empieza a recibir una trama, para activar el receptor y el sintetizador PLL. Cuando se identifica un encabezado de sincronización válido (SHR), el receptor ingresa al estado de recepción ocupado y permanece allí hasta que se completa la recepción de la trama, lo que garantiza que la trama se reciba correctamente [16].

Durante la recepción, los bytes de la trama entrante se almacenan en el búfer hasta que se recibe el último byte, momento en el que una interrupción indica que la recepción está completa y el transceptor vuelve al estado de recepción.

2.4.2.1.4 Estado sleep

Cuando el dispositivo no está en uso, cambia a este estado para ahorrar energía al desactivar el transceptor y la circuitería. El contenido de los registros se mantiene y el oscilador del microcontrolador permanece activo, lo que permite que el temporizador siga funcionando. Toda la información contenida en el búfer es eliminada [16].

2.4.2.1.5 Reinicio

El estado de reinicio restaura los valores predeterminados de los registros del microcontrolador y obliga al transceptor a pasar al estado de espera. Este estado solo puede ser activado desde el exterior [16].

2.4.3 USO DEL LENGUAJE DE PROGRAMACIÓN DEL NODO RCB256RFR2

El nodo utiliza el lenguaje de programación C, por lo que el dispositivo no puede ejecutar procesos multihilo[5]. Debido a esto el nodo solo puede ejecutar un proceso a la vez, siendo otra razón por la cual el nodo no puede recibir y enviar varios mensajes simultáneamente.

Ya que el lenguaje C no permite programación orientada a objetos, la generación de mensajes MQTT-SN se la realiza mediante agrupación de vectores. Al ser mensajes simples no ocupan gran cantidad de recursos de los dispositivos, por esta razón se considera que no debe existir problemas en las entradas y salidas de la MEF.

El nodo permanece en un determinado estado hasta que reciba un mensaje o señal que permita realizar un cambio de estado. El leguaje C permite codificar esta funcionalidad sin ningún problema dentro del nodo, por ende, tampoco debe existir problema en el desarrollo de la MEF.

[5] Proceso multihilo: Hace referencia a la capacidad de ejecutar varios procesos de manera concurrente [34].

2.4.4 INTERACCIÓN ENTRE BROKER MQTT Y GATEWAY MQTT-SN

En este proyecto no se desarrolla ninguna MEF correspondiente al broker MQTT. Sin embargo, se debe tomar en cuenta el funcionamiento de este dispositivo para la creación de las MEF, sobre todo las que tienen que ver con el gateway.

En una red MQTT-SN, el gateway debe anunciar su presencia periódicamente, si el gateway es independiente, primero debe establecer conexión con el broker, mientras que, si el gateway está integrado al broker, anuncia su presencia cuando inicia su funcionamiento. En los dos casos se establece una señal que indique que el broker está listo para recibir mensajes para el desarrollo de la máquina de estados.

Por otro lado, al no utilizar un broker en la red MQTT-SN planteada, se considera que el gateway responde a los mensajes enviados por el cliente de manera similar a como lo haría si se encuentra conectado a un broker o a su vez es parte de un broker. No se toma en cuenta el tiempo que debería tardar en responder el broker.

2.5 DESARROLLO DE LA MÁQUINA DE ESTADOS FINITOS DEL PROTOCOLO MQTT-SN PARA SU OPERACIÓN SOBRE EL NODO RCB256RFR2

Debido a que en la mayoría de los procedimientos cada dispositivo involucrado responden a la recepción de un mensaje con otro mensaje, se opta por utilizar la máquina de Mealy para representar los estados por los que pasan los nodos cliente y gateway. La máquina de Mealy permite aceptar el ingreso de diferentes secuencias de mensajes MQTT-SN y a su vez envía una secuencia de mensajes MQTT-SN de salida. Solo utiliza un estado de inicio y ningún estado final, por lo cual se puede simular en la aplicación JFLAP.

Como el protocolo MQTT-SN consta de 12 procedimientos revisados en el literal 2.2, para su operación se opta por desarrollar una máquina de Mealy (para el cliente y el gateway) de cada procedimiento de forma independiente, ya que será más sencillo el entendimiento de cada MEF obtenida. Además, las secuencias de mensajes permitidas a utilizarse son más cortas y dependen de cada procedimiento. De igual manera, es más sencillo comprobar el funcionamiento de un procedimiento a la vez que el de todos en una solo prueba.

A pesar de que los procedimientos MQTT-SN se los considera como independientes, hay que tener en cuenta que el nodo cliente y gateway deben terminar de ejecutar un procedimiento para iniciar otro.

2.5.1 OTROS EVENTOS QUE INFLUIRÁN EN EL DESARROLLO DE LA MEF

Aparte de la recepción de mensajes MQTT-SN, un cambio de estado también es posible cuando ha terminado el tiempo de un temporizador establecido previamente. También, se puede producir un cambio de estado por una señal interna generada dentro del propio nodo. Por ejemplo, cuando el cliente tiene lista una nueva publicación se debe indicar que el mensaje *PUBLISH* está listo para publicarse. Para recrear estas señales internas se hará uso del pulsador ubicado en la placa del nodo RCB256RFR2.

Por otro lado, los mensajes que tienen un campo returnCode o el campo flags (QoS y will) también puede producir un cambio de estado.

2.5.2 PROCEDIMIENTO ANUNCIO Y DESCUBRIMIENTO DE GATEWAY

2.5.2.1 Gateway

El diagrama de máquina de estados para este procedimiento indica que el dispositivo puede permanecer en tres estados: inactivo, espera anuncio y respaldo (Figura 2.22).

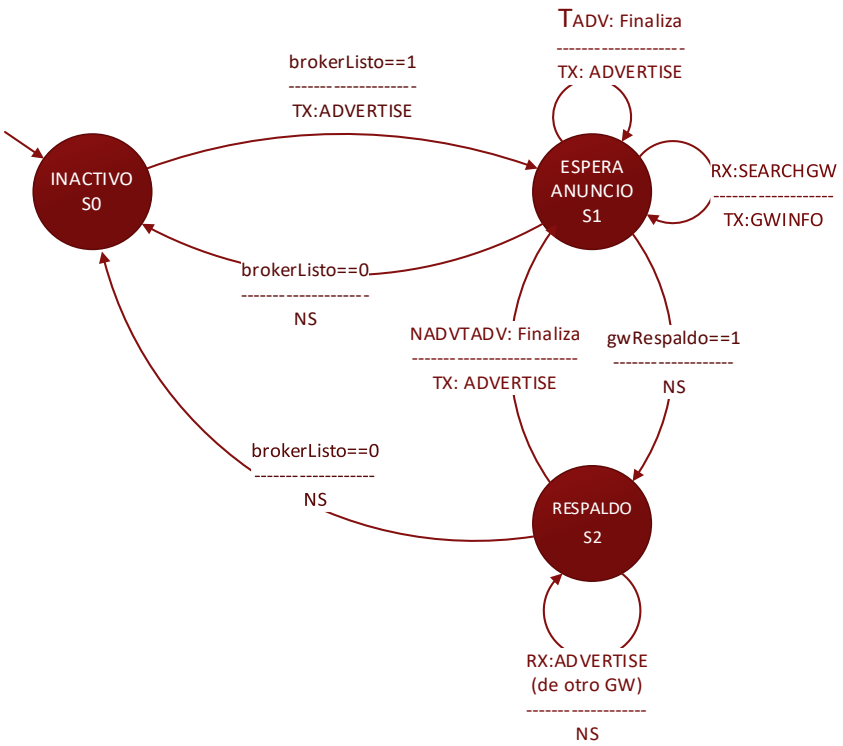


Figura 2.22 Diagrama de estados del gateway (Procedimiento anuncio y descubrimiento).

En el momento en que el broker MQTT esté listo para recibir y transmitir mensajes, el nodo gateway también estará listo para iniciar su funcionamiento, si este último es parte del broker; caso contrario se debe establecer conexión entre ambos dispositivos para que el gateway inicie su operación. Por lo anterior, el nodo permanece en estado inactivo hasta

que el broker, mediante la señal de entrada *brokerListo=1*, le indique al gateway que puede iniciar su funcionamiento. Recibida la señal de entrada el dispositivo responde con el mensaje *ADVERTISE.*

Cuando el dispositivo llega al estado espera anuncio, el gateway envía periódicamente mensajes *ADVERTISE* cuando el tiempo del temporizador *TADV* finaliza, además, durante este estado el gateway puede recibir mensajes *SEARCHGW* para responder de inmediato con un *GWINFO*. De esta manera el nodo puede anunciar su presencia en una red MQTT-SN.

También, se tiene un estado denominado respaldo, al cual el nodo llegará si recibe la señal *gwRespaldo=1* cuando está en estado espera anuncio. Durante este estado el nodo espera por mensajes *ADVERTISE* de otro gateway cercano, el cual debe anunciar su presencia periódicamente. El gateway en estado de respaldo regresara al estado de anuncio si el temporizador *NADVTADV* finaliza, tras dejar de recibir mensajes *ADVERTISE* de otro gateway. Además, desde el estado inactivo no puede pasar al estado respaldo directamente, se debe esperar a que el broker indique que está listo para funcionar.

Finalmente, un Gateway pasará al estado inactivo, desde cualquier estado, si el broker por algún motivo deja de funcionar, para lo cual se utiliza una señal *brokerListo=0*.

Forma normal

Se define una 6-tupla {Q, Σ, S, δ, λ, q0} donde:
Q= {INACTIVO (S0), ANUNCIO (S1), RESPALDO (S3)};
Σ= {brokerListo=1, brokerListo=0, TADV, *SEARCHGW*, gwRespaldo, *ADVERTISE*, NADVTADV};
S= {*ADVERTISE*, *GWINFO*};
q0= INACTIVO
δ= Función de transición.

Tabla 2.1 Función de transición del gateway (procedimiento anuncio y descubrimiento).

ESTADO ACTUAL		**ESTADO SIGUIENTE**						
ENTRADA:		broker Lst=1	broker Lst=0	TADV	SEARCHGW	gwRespaldo =1	ADVERTISE	NADVTADV
INACTIVO	**S0**	S1	S0	S0	S0	S0	S0	S0
ESP_ANUNCIO	**S1**	S1	S0	S1	S1	S2	S1	S1
RESPALDO	**S2**	S2	S0	S2	S2	S2	S2	S1

λ: Función de salida:

Tabla 2.2 Función de salida del gateway (procedimiento anuncio y descubrimiento).

ESTADO ACTUAL		SALIDA						
ENTRADA:		broker Lst=1	broker Lst=1	TADV	SEARCHGW	gwRespaldo =1	ADVERTISE	NADV TADV
INACTIVO	**S0**	ADVERTISE	NS	NS	NS	NS	NS	NS
ESP_ANUNCIO	**S1**	NS	NS	ADVERTISE	GWINFO	NS	NS	NS
RESPALDO	**S2**	NS	NS	NS	NS	NS	NS	ADVERTISE

2.5.2.2 Cliente

Si bien un cliente MQTT-SN debe tener la capacidad de responder mensajes de otros clientes, para el desarrollo de esta máquina de Mealy no se tomó en cuenta esta característica ya que no es necesaria si el nodo opera sobre una topología lineal.

Por lo anterior solo serán necesarios dos estados: Descubrimiento y RX_*GWINFO* (Figura 2.23). El estado inicial es el estado de Descubrimiento, durante el cual el nodo cliente espera recibir mensajes *ADVERTISE* del nodo gateway, también espera a que los temporizadores *NADVTADV* o *TSEARCHGW* finalicen.

Cuando termina el tiempo del temporizador *TSEARCHGW* se envía un mensaje *SEARCHGW* y el nodo pasa al estado RX_*GWINFO*, en el estado mencionado el nodo espera el mensaje *GWINFO* para regresar al estado inicial para seguir esperando los mensajes de advertencia del gateway. Por el contrario, de no recibir el mensaje esperado se puede retransmitir el mensaje *SEARCHGW* previamente enviado.

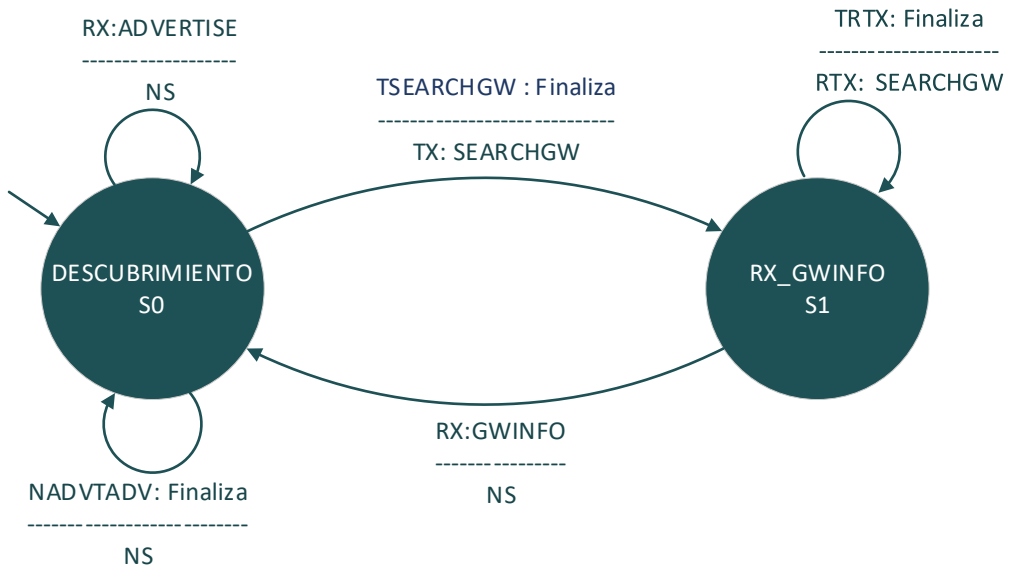


Figura 2.23 Diagrama de estados del cliente (Procedimiento anuncio y descubrimiento).

Forma normal

Se define una 6-tupla {Q, Σ, S, δ, λ, q0} donde:

Q= {DESCUBRIMIENTO (S0), RX_*GWINFO* (S1)};

Σ= {*ADVERTISE*, T*SEARCHGW*, NADVTADV, *GWINFO*, TRTX};

S= {*SEARCHGW*};

q0= DESCUBRIMIENTO.

δ= Función de transición:

Tabla 2.3 Función de transición del cliente (procedimiento anuncio y descubrimiento).

ESTADO ACTUAL		**ESTADO SIGUIENTE**				
ENTRADA:		ADVERTISE	TSEARCHGW	NADVTADV	GWINFO	TRTX
DESCUBRIMIENTO	**S0**	S0	S1	S0	S0	S0
RX_GWINFO	**S1**	S1	S1	S1	S0	S1

λ: Función de salida:

Tabla 2.4 Función de salida del cliente (procedimiento anuncio y descubrimiento).

ESTADO ACTUAL		**SALIDA**				
ENTRADA:		ADVERTISE	TSEARCHGW	NADVTADV	GWINFO	TRTX
DESCUBRIMIENTO	**S0**	NS	SEARCHGW	NS	NS	NS
RX_GWINFO	**S1**	NS	NS	NS	NS	SEARCHGW

2.5.3 CONFIGURACIÓN DE CONEXIÓN DEL CLIENTE

2.5.3.1 Gateway

Este procedimiento inicia en el momento, que el procedimiento anterior a enviado al menos un mensaje de advertencia. En consecuencia, el estado de ESPERA ANUNCIO pasa a denominarse ESPERA CONEXIÓN y es considerado el estado inicial del procedimiento para configuración de conexión del cliente. Además, se maneja los estados RX_TOPIC y RX_MSG que se utilizan según la configuración de la bandera will (Figura 2.24).

En el estado inicial el nodo espera por mensajes *CONNECT*. El nodo permanece en el mismo estado, después de responder con un mensaje *CONNACK,* si el mensaje recibido contiene la bandera will con el valor de cero. Por el contrario, habrá un cambio de estado y se envía un mensaje de petición *WILLTOPICREQ* si se recibe un mensaje con la bandera will igual a uno.

En el estado RX_TOPIC, el nodo espera un mensaje *WILLTOPIC* para enviar un *WILLMSGREQ* y pasar al estado RX_MSG. Por último, en el momento que es recibido un

mensaje *WILLMSG* se transmite un mensaje *CONNACK* y el nodo regresa al estado inicial del procedimiento a esperar por otra conexión.

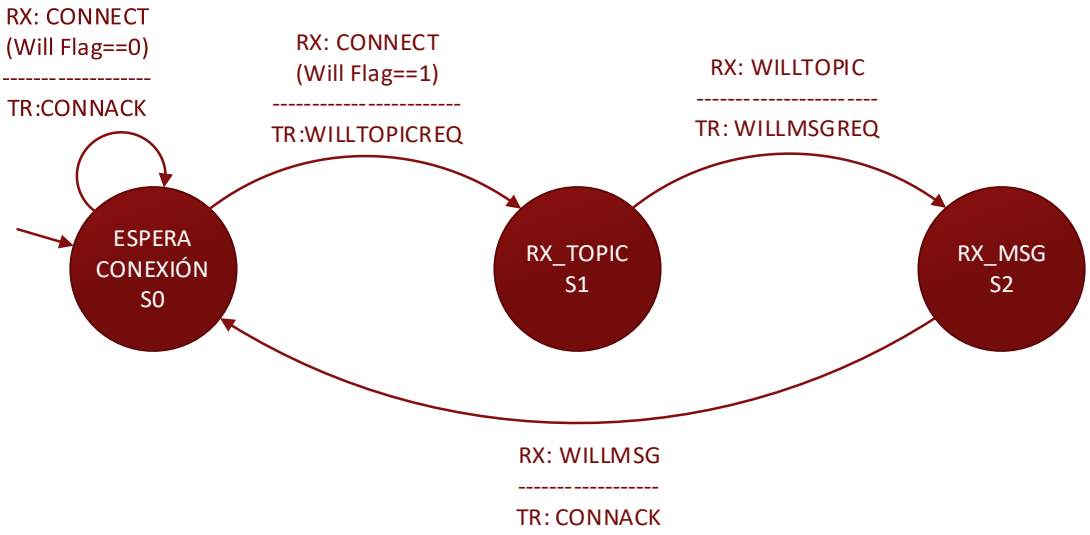


Figura 2.24 Diagrama de estados (Gateway): Configuración de conexión del cliente.

Forma normal

Se define una 6-tupla {Q, Σ, S, δ, λ, q0} donde:

Q= {ESPERA CONEXIÓN (S0), RX_TOPIC (S1), RX_MSG (S2)};

Σ= {WillFlag=0, WillFlag=1, *WILLTOPIC*, *WILLMSG*};

S= {*CONNACK*, *WILLTOPICREQ*, *WILLMSGREQ*};

q0= ESPERA CONEXIÓN

δ= Función de transición:

Tabla 2.5 Función de transición del gateway (Configuración de conexión del cliente).

ESTADO ACTUAL		**ESTADO SIGUIENTE**			
ENTRADA:		WillFlag=0	WillFlag=1	WILLTOPIC	WILLMSG
ESP_CONEXIÓN	**S0**	S0	S1	S0	S0
RX_TOPIC	**S1**	S1	S1	S2	S1
RX_MSG	**S2**	S2	S2	S2	S0

λ: Función de salida:

Tabla 2.6 Función de salida del gateway (Configuración de conexión del cliente).

ESTADO ACTUAL		**SALIDA**			
ENTRADA:		WillFlag=0	WillFlag=1	WILLTOPIC	WILLMSG
ESP_CONEXIÓN	**S0**	CONNACK	WILLTOPICREQ	NS	NS
RX_TOPIC	**S1**	NS	NS	WILLMSGREQ	NS
RX_MSG	**S2**	NS	NS	NS	CONNACK

2.5.3.2 Cliente

Cuando un nodo cliente ha encontrado un gateway al cual se pueda conectar, iniciara el procedimiento para establecer una conexión. Por esto se utiliza un estado inicial denominado establecer conexión, en el cual se verifica el valor de la bandera will antes de ser enviada dentro de un mensaje *CONNECT*.

De acuerdo al diagrama de estados de la Figura 2.25, si la bandera will es igual a uno, el nodo pasa al estado RX_TOPIC_REQ donde espera recibir el mensaje *WILLTOPICREQ*, para después pasar al estado RX_TOPIC_REQ en el que espera recibir el mensaje *WILLMSGREQ* y pasar al estado RX_*CONNACK*. El nodo transmite los mensajes *WILLTOPIC* y *WILLMSG* luego de recibir la petición respectiva en su estado correspondiente.

Si la bandera will es igual a cero, el nodo pasara directamente al estado RX_*CONNACK* después de enviar su mensaje *CONNECT*.

En el estado RX_*CONNACK* el nodo espera un mensaje *CONNACK*, el cual contiene el campo ReturnCode que indica si la conexión es aceptada o rechazada. Cuando la conexión es aceptada o rechazada (ReturnCode: no soportado) el nodo regresa al estado inicial para ejecutar otro procedimiento o reiniciar la conexión

En el caso de existir un rechazo por congestión, el nodo pasara al estado descongestión en el cual permanecerá hasta que el temporizador TWAIT finalice. Después el nodo regresara al estado inicial para intentar conectarse al Gateway nuevamente.

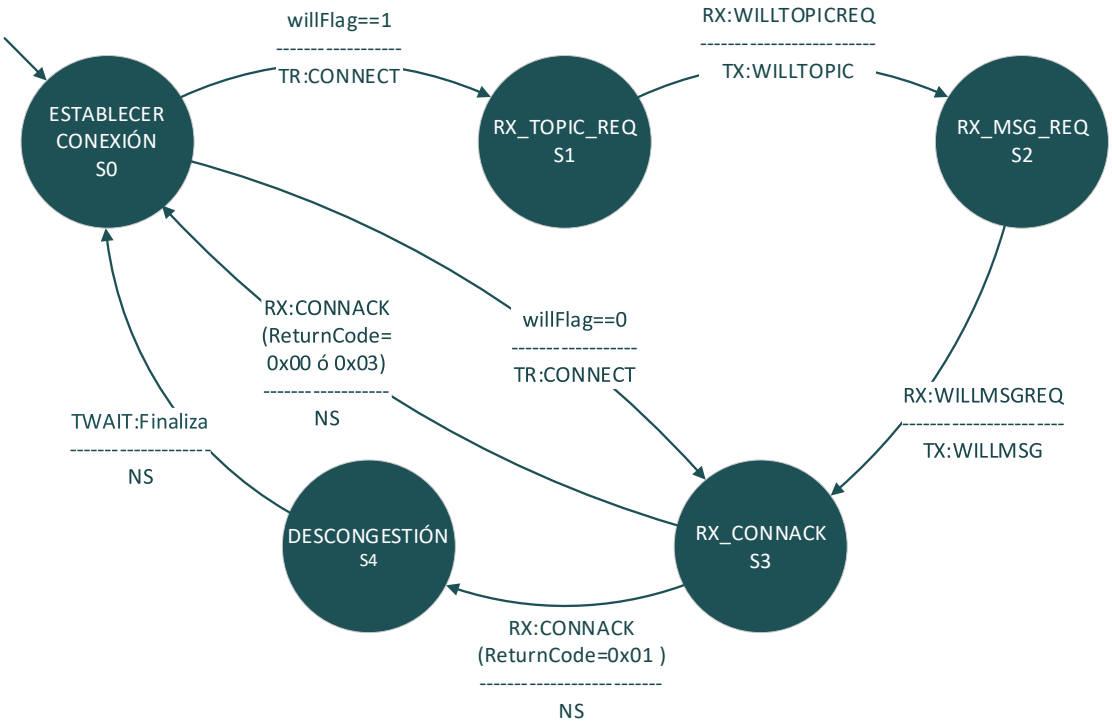


Figura 2.25 Diagrama de estados (Cliente): Configuración de conexión del cliente.

Forma normal

Se define una 6-tupla {Q, Σ, S, δ, λ, q0} donde:

Q= {ESTABLECER CONEXIÓN(S0), RX_TOPIC_REQ (S1), RX_MSG_REQ (S2), RX_*CONNACK* (S3), DESCONGESTIÓN (S4)};

Σ= {WillFlag=0, WillFlag=1, *CONNACK* (0x00 o 0x03), *CONNACK*(0x01), TWAIT, *WILLTOPICREQ*, *WILLMSGREQ*};

S= {*CONNECT*, *WILLTOPIC*, *WILLMSG*};

q0= ESTABLECER CONEXIÓN

δ= Función de transición:

Tabla 2.7 Función de transición del cliente (Configuración de conexión del cliente).

ESTADO ACTUAL		**ESTADO SIGUIENTE**						
ENTRADA:		WillFlag=0	WillFlag=1	WILLTOPICREQ	WILLMSGREQ	CONNACK (0x00 o0x03)	CONNACK (0x01)	T WAIT
EST_CONEXIÓN	**S0**	S3	S1	S0	S0	S0	S0	S0
RX_TOPIC_REQ	**S1**	S1	S1	S2	S1	S1	S1	S1
RX_MSG_REQ	**S2**	S2	S2	S2	S3	S2	S2	S2
RX_CONNACK	**S3**	S3	S3	S1	S3	S1	S4	S3
DESCONGESTIÓN	**S4**	S4	S4	S4	S4	S4	S4	S1

λ: Función de salida:

Tabla 2.8 Función de salida del cliente (Configuración de conexión del cliente).

ESTADO ACTUAL		SALIDA						
ENTRADA:		WillFlag=0	WillFlag=1	WILLTOPICREQ	WILLMSGREQ	CONNACK (0x00 o0x03)	CONNACK (0x01)	T WAIT
EST_CONEXIÓN	**S0**	CONNECT	CONNECT	NS	NS	NS	NS	NS
RX_TOPIC_REQ	**S1**	NS	NS	WILLTOPIC	NS	NS	NS	NS
RX_MSG_REQ	**S2**	NS	NS	NS	WILLMSG	NS	NS	NS
RX_CONNACK	**S3**	NS	NS	NS	NS	NS	NS	NS
DESCONGESTIÓN	**S4**	NS	NS	NS	NS	NS	NS	NS

2.5.4 PROCEDIMIENTO PARA ACTUALIZAR DATOS WILL

2.5.4.1 Gateway

Si un gateway ha recibido un mensaje o tema will, tendrá la posibilidad de recibir peticiones para actualizar los datos will en cualquier momento. El procedimiento para actualizar datos will solo tendrá un estado, el cual puede recibir un *WILLTOPIC*UPD o *WILLMSG*UPD para después responder con un *WILLTOPIC*RESP o *WILLMSG*RESP respectivamente y regresar al mismo estado, tal como se muestra en la Figura 2.26.

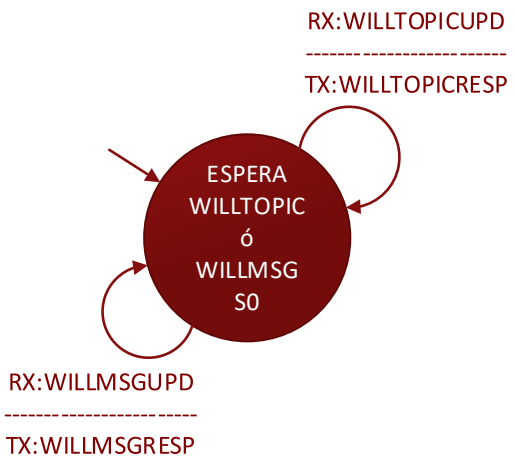


Figura 2.26 Diagrama de estados (Cliente): Procedimiento para actualizar datos Will.

Forma normal

Se define una 6-tupla {Q, Σ, S, δ, λ, q0} donde:

Q= {ESPERA *WILLTOPIC* o *WILLMSG* (S0)};

Σ= {*WILLTOPIC*UPD, *WILLMSG*UPD};

S= {*WILLTOPIC*RESP, *WILLMSG*RESP};

q0= ESPERA *WILLTOPIC* o *WILLMSG*

δ= Función de transición:

Tabla 2.9 Función de transición del gateway (Procedimiento para actualizar datos will).

ESTADO ACTUAL		**ESTADO SIGUIENTE**	
ENTRADA:		WILLTOPICUPD	WILLMSGUPD
ESPERA	**S0**	S0	S0

λ: Función de salida:

Tabla 2.10 Función de salida del gateway (Procedimiento para actualizar datos will).

ESTADO ACTUAL		**SALIDA**	
ENTRADA:		WILLTOPICUPD	WILLMSGUPD
ESPERA	**S0**	WILLTOPICRESP	WILLMSGUPD

2.5.4.2 Cliente

Una vez conectado a un gateway un cliente es considerado como cliente activo y podrá actualizar sus datos will en cualquier momento al ejecutar el procedimiento para actualizar datos will. Si el nodo cliente desea actualizar su tema will necesita recibir una señal *actualizarWILLTOPIC=1,* para transmitir un mensaje *WILLTOPIC*UPD y pasar al estado RX_TOPIC_RESP. De manear similar, si el nodo cliente desea actualizar su mensaje will necesita recibir una señal *actualizarWILLMSG=1,* para transmitir un mensaje *WILLMSG*UPD y pasar al estado RX_MSG_RESP. El cliente regresa al estado inicial en cuanto reciba su respectiva respuesta, *WILLTOPIC*RESP o *WILLMSG*RESP. A continuación, en la Figura 2.27 se muestra el diagrama de estados de este procedimiento.

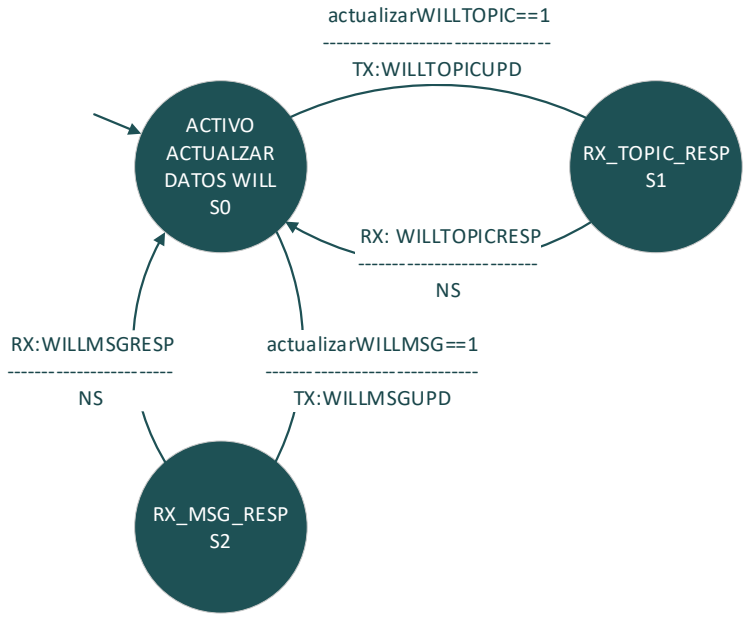


Figura 2.27 Diagrama de estados (Cliente): Procedimiento para actualizar datos will.

Forma normal

Se define una 6-tupla {Q, Σ, S, δ, λ, q0} donde:

Q= {ACTIVO ACTUALZAR DATOS WILL (S0), RX_TOPIC_ RESP (S1), RX_MSG _ RESP (S2)};

Σ= {actualizar*WILLTOPIC*=1, actualizar*WILLMSG*==1, *WILLTOPIC*RESP, *WILLMSG*RESP};

S= {*WILLTOPIC*UPD, *WILLMSG*UPD};

q0= ACTIVO ACTUALZAR DATOS WILL

δ= Función de transición:

Tabla 2.11 Función de transición del cliente (Procedimiento para actualizar datos will).

ESTADO ACTUAL		**ESTADO SIGUIENTE**			
ENTRADA:		Actualizar WILLTOPIC=1	Actualizar WILLMSG=1	WILLTOPICRESP	WILLMSGRESP
ACTIVO	**S0**	S1	S2	S0	S0
RX_TOPIC_RESP	**S1**	S1	S1	S0	S1
RX_MSG_RESP	**S2**	S2	S2	S2	S0

λ: Función de salida:

Tabla 2.12 Función de salida del cliente (Procedimiento para actualizar datos will).

ESTADO ACTUAL		**SALIDA**			
ENTRADA:		Actualizar WILLTOPIC=1	Actualizar WILLMSG=1	WILLTOPICRESP	WILLMSGRESP
ACTIVO	**S0**	WILLTOPICUPD	WILLMSGUPD	NS	NS
RX_TOPIC_RESP	**S1**	NS	NS	NS	NS
RX_MSG_RESP	**S2**	NS	NS	NS	NS

2.5.5 PROCEDIMIENTO PARA REGISTRAR NOMBRES DE TEMAS

2.5.5.1 Gateway

Después de que el gateway recibe una conexión, se mantiene en espera de un registro enviado por el cliente conectado, por lo cual para este procedimiento se establece el estado inicial espera registro. Durante el estado inicial el nodo responde con un mensaje *REACK* ante la recepción de un mensaje *REGISTER*, para después regresar al mismo estado.

Dado que el gateway también tiene la capacidad de iniciar un procedimiento de registro, se puede enviar un mensaje *REGISTER* en cuanto le llegué la señal *listoRegistro=1* desde el estado inicial. Después de enviar el mensaje anterior el nodo pasa al estado

RX_*REGACK*, donde espera un mensaje *REACK* enviado por el cliente. En el momento que es recibido el mensaje *REACK* se revisa su campo ReturnCode. Cuando el registro es aceptado o rechazado (ReturnCode=identificador de tema invalido o no soportado) el nodo regresa al estado inicial para ejecutar otro procedimiento o reiniciar el procedimiento de registro. En la Figura 2.28 se muestra el diagrama de estados de este procedimiento.

En el caso de existir un rechazo por congestión, el nodo gateway debe pasar al estado descongestión en el cual permanecerá hasta que el temporizador TWAIT finalice. Después el nodo regresará al estado inicial para intentar registrar el nombre de tema de nuevo.

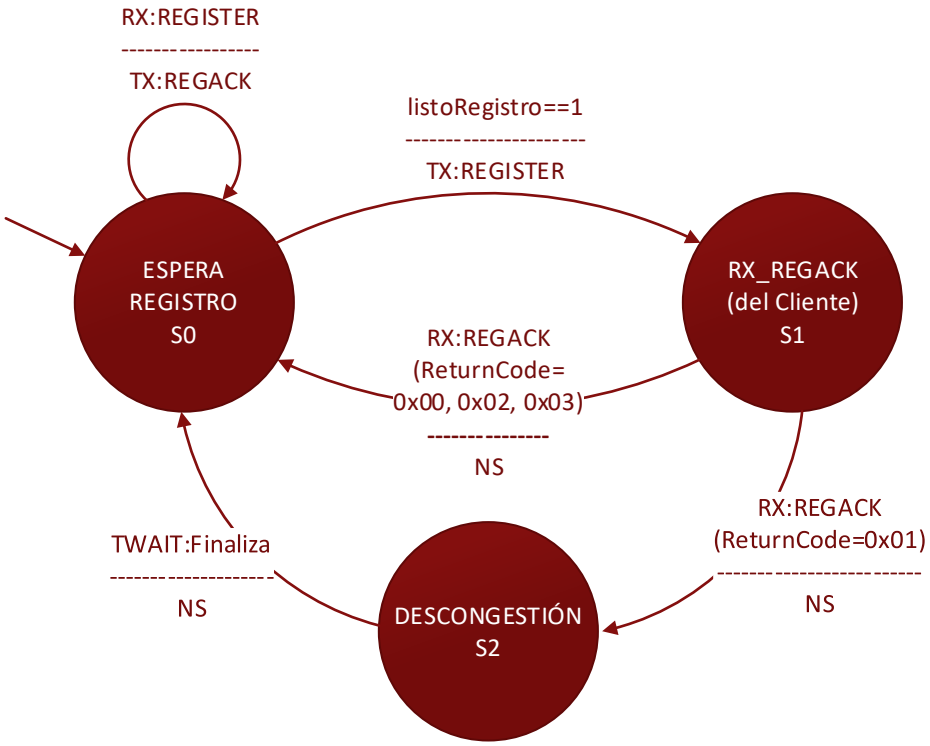


Figura 2.28 Diagrama de estados (Gateway): Procedimiento para registrar nombres de temas.

Forma normal

Se define una 6-tupla {Q, Σ, S, δ, λ, q0} donde:

Q= {ESPERA REGISTRO(S0), RX_*REGACK*(S1), DESCONGESTIÓN(S2)};

Σ= {*REGISTER*, listoRegistro=1, *REGACK* (! =0x01), *REGACK*(0x01), TWAIT};

S= {*REGACK*, *REGISTER*};

q0= ESPERA REGISTRO

δ= Función de transición:

Tabla 2.13 Función de transición del gateway (Procedimiento para registrar nombres de temas).

ESTADO ACTUAL		ESTADO SIGUIENTE				
ENTRADA:		REGISTER	listoRegistro=1	REGACK (! =0x01)	REGACK(0x01)	TWAIT
ESPERA REGISTRO	**S0**	S0	S1	S0	S0	S0
RX_REGACK	**S1**	S1	S1	S0	S2	S1
DESCONGESTIÓN	**S2**	S2	S2	S2	S2	S0

λ: Función de salida:

Tabla 2.14 Función de salida del gateway (Procedimiento para registrar nombres de temas).

ESTADO ACTUAL		SALIDA			
ENTRADA:		REGISTER	listoRegistro=1	REGACK (! =0x01)	REGACK(0x01)
ESPERA REGISTRO	**S0**	REGACK	REGISTER	NS	NS
RX_REGACK	**S1**	NS	NS	NS	NS
DESCONGESTIÓN	**S2**	NS	NS	NS	NS

2.5.5.2 Cliente

Después de que el cliente establezca una conexión exitosa, se lo considera como un cliente activo, pero si requiere realizar alguna publicación primero debe iniciar un procedimiento de registro. Por esta razón, a pesar de que el diagrama de estado para el cliente es similar al diagrama del gateway, al estado inicial se lo denominará como Activo Registro.

Durante el estado inicial el cliente envía un mensaje *REGISTER* cuando le llegué la señal listoRegistro=1. Después de enviar el mensaje anterior el nodo pasa al estado RX*_REGACK*, donde espera un mensaje *REACK* enviado por el gateway. En el momento que es recibido el mensaje *REACK* se revisa su campo ReturnCode. Cuando el registro es aceptado o rechazado (ReturnCode=no soportado) el nodo regresa al estado inicial para ejecutar otro procedimiento o reiniciar el procedimiento de registro.

Además, si el nodo cliente se encuentra en el estado inicial, puede responder con un mensaje *REACK* ante la recepción de un mensaje *REGISTER* enviado por el nodo gateway. En la Figura 2.29 se muestra el diagrama de estados de este procedimiento.

En el caso de existir un rechazo por congestión, el nodo cliente pasa al estado descongestión en el cual permanecerá hasta que el temporizador TWAIT finalice. Después

el nodo tiene que regresar al estado inicial para intentar registrar el nombre de tema de nuevo.

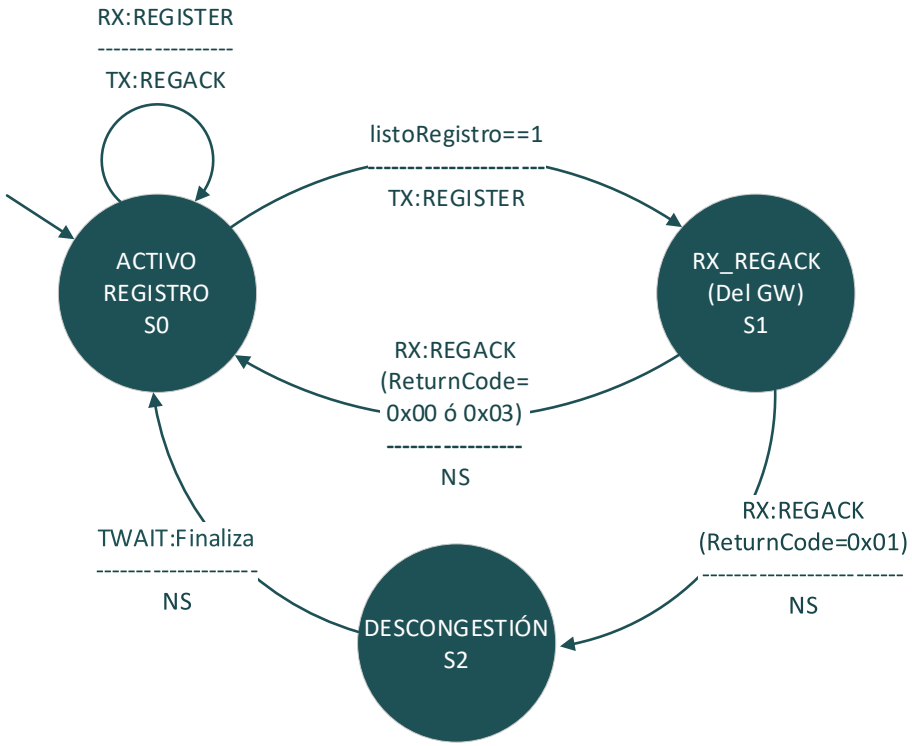


Figura 2.29 Diagrama de estados (Cliente): Procedimiento para registrar nombres de temas.

Forma normal

Se define una 6-tupla {Q, Σ, S, δ, λ, q0} donde:

Q= {ACTIVO REGISTRO(S0), RX_*REGACK*(S1), DESCONGESTIÓN(S2)};

Σ= {*REGISTER*, listoRegistro=1, *REGACK* (0x00,0x03), *REGACK*(0x01), TWAIT};

S= {*REGACK*, *REGISTER*};

q0= ACTIVO REGISTRO

δ= Función de transición:

Tabla 2.15 Función de transición del cliente (Procedimiento para registrar nombres de temas).

ESTADO ACTUAL		**ESTADO SIGUIENTE**				
ENTRADA:		REGISTER	listoRegistro=1	REGACK (0x00,0x03)	REGACK(0x01)	TWAIT
ACTIVO REGISTRO	**S0**	S0	S1	S0	S0	S0
RX_REGACK	**S1**	S1	S1	S0	S2	S1
DESCONGESTIÓN	**S2**	S2	S2	S2	S2	S0

λ: Función de salida:

Tabla 2.16 Función de salida del cliente (Procedimiento para registrar nombres de temas).

ESTADO ACTUAL		**SALIDA**			
ENTRADA:		REGISTER	listoRegistro=1	REGACK (0x00,0x03)	REGACK(0x01)
ACTIVO REGISTRO	**S0**	REGACK	REGISTER	NS	NS
RX_REGACK	**S1**	NS	NS	NS	NS
DESCONGESTIÓN	**S2**	NS	NS	NS	NS

2.5.6 PROCEDIMIENTO DE PUBLICACIÓN(CLIENTE)

2.5.6.1 Gateway

Si el gateway ha finalizado con éxito un procedimiento de registro, iniciado por el cliente, ya puede iniciar un procedimiento de publicación. Por lo cual, se establece para el nodo gateway, un estado inicial denominado ‘espera publicación’. Durante este estado el nodo tiene la capacidad de recibir mensajes *PUBLISH* con las diferentes variantes de calidad de servicio. A continuación, se describe como debe responder el nodo ante cada uno de los niveles de calidad de servicio:

- *PUBLISH* (QoS 0): Cuando se recibe una publicación con esta calidad de servicio, el nodo no responde con ningún mensaje y regresa estado inicial.
- *PUBLISH* (QoS 1): Cuando se recibe una publicación con esta calidad de servicio, el nodo responde con un mensaje *PUBACK* para después regresar al estado inicial.
- *PUBLISH* (QoS 2): Cuando se recibe una publicación con esta calidad de servicio, el nodo responde con un mensaje *PUBREC* para después pasar al estado RX_*PUBREL*. Cuando el nodo se encuentra en el estado mencionado, espera hasta que le llegue el mensaje *PUBREL* para responder con un *PUBCOMP* y regresar al estado inicial.

Además, si el nodo está en el estado inicial y recibe un mensaje *PUBLISH* con un identificador de tema inválido, enviara en respuesta un mensaje *PUBACK* que indica el motivo del rechazo (ReturnCode=0x02), sin importar el nivel de calidad de servicio. Finalmente, el nodo debe regresar al estado inicial para atender nuevas publicaciones o iniciar nuevos procedimientos. En la Figura 2.30 se muestra el diagrama de estados de este procedimiento.

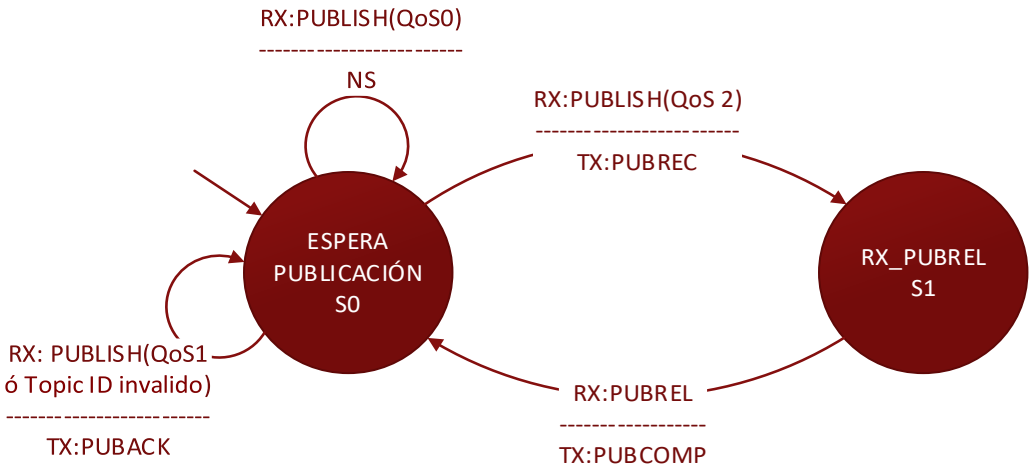


Figura 2.30 Diagrama de estados (Gateway): Procedimiento de publicación del cliente.

Forma normal

Se define una 6-tupla {Q, Σ, S, δ, λ, q0} donde:

Q= {ESPERA PUBLICACIÓN(S0), RX_*PUBREL*(S1)};

Σ= {*PUBLISH*(QoS0), *PUBLISH*(QoS1), *PUBLISH* (ID invalido), *PUBLISH*(QoS2), *PUBREL*};

S= {*PUBACK*, *PUBREC*, *PUBCOMP*};

q0= ESPERA PUBLICACIÓN

δ= Función de transición:

Tabla 2.17 Función de transición del gateway (Procedimiento de publicación del cliente).

ESTADO ACTUAL		**ESTADO SIGUIENTE**				
ENTRADA:		PUBLISH (QoS0)	PUBLISH (QoS1)	PUBLISH (QoS2)	PUBLISH (ID invalido)	PUBREL
ESPERA PUBLICACIÓN	**S0**	S0	S0	S1	S0	S0
RX_PUBREL	**S1**	S1	S1	S1	S1	S0

λ: Función de salida:

Tabla 2.18 Función de salida del gateway (Procedimiento de publicación del cliente).

ESTADO ACTUAL		**SALIDA**				
ENTRADA:		PUBLISH (QoS0)	PUBLISH (QoS1)	PUBLISH (QoS2)	PUBLISH (ID invalido)	PUBREL
ESPERA PUBLICACIÓN	**S0**	NS	PUBACK	PUBREC	PUBACK	NS
RX_PUBREL	**S1**	NS	NS	NS	NS	PUBCOMP

2.5.6.2 Cliente

Si el cliente ha finalizado un procedimiento de registro ya puede iniciar un procedimiento de publicación. Por lo cual se establece, un estado inicial denominado activo publicación. Durante el estado inicial el nodo puede enviar mensajes *PUBLISH*. A continuación, se describe el comportamiento del nodo cliente cuando envía publicaciones con los diferentes niveles de calidad de servicio:

PUBLISH (QoS 0): Cuando el nodo recibe la señal *listaPublicación(QoS0)=1*, transmite un mensaje *PUBLISH* y el nodo permanece en el estado inicial.

PUBLISH (QoS 1): Cuando el nodo recibe la señal *listaPublicación(QoS1)=1*, transmite un mensaje *PUBLISH* y el nodo pasa al estado RX_ACK. Durante este estado el nodo espera un mensaje *PUBACK* para revisar el campo ReturnCode. Si la publicación es aceptada o rechazada, por un motivo diferente a la congestión, el nodo regresa al estado inicial para ejecutar otro procedimiento o reiniciar el procedimiento de publicación.

En el caso de existir un rechazo por congestión, el nodo pasara al estado descongestión en el cual permanecerá hasta que el temporizador TWAIT finalice. Después el nodo regresara al estado inicial para intentar enviar el mensaje de publicación nuevamente.

PUBLISH (QoS 2): Cuando el nodo recibe la señal *listaPublicación(QoS2)=1*, transmite un mensaje *PUBLISH* y el nodo pasa al estado RX_ACK. Durante este estado el nodo espera recibir un mensaje *PUBREC* para transmitir un mensaje *PUBREL* y pasar al estado RX_*PUBCOMP*. Si durante este último estado se recibe un mensaje *PUBCOMP* la publicación se considera como aceptada.

También hay que mencionar que durante el estado inicial también se puede recibir un mensaje *PUBACK*, siempre y cuando contenga el campo ReturnCode=0x02. En la Figura 2.31 se muestra el diagrama de estados de este procedimiento.

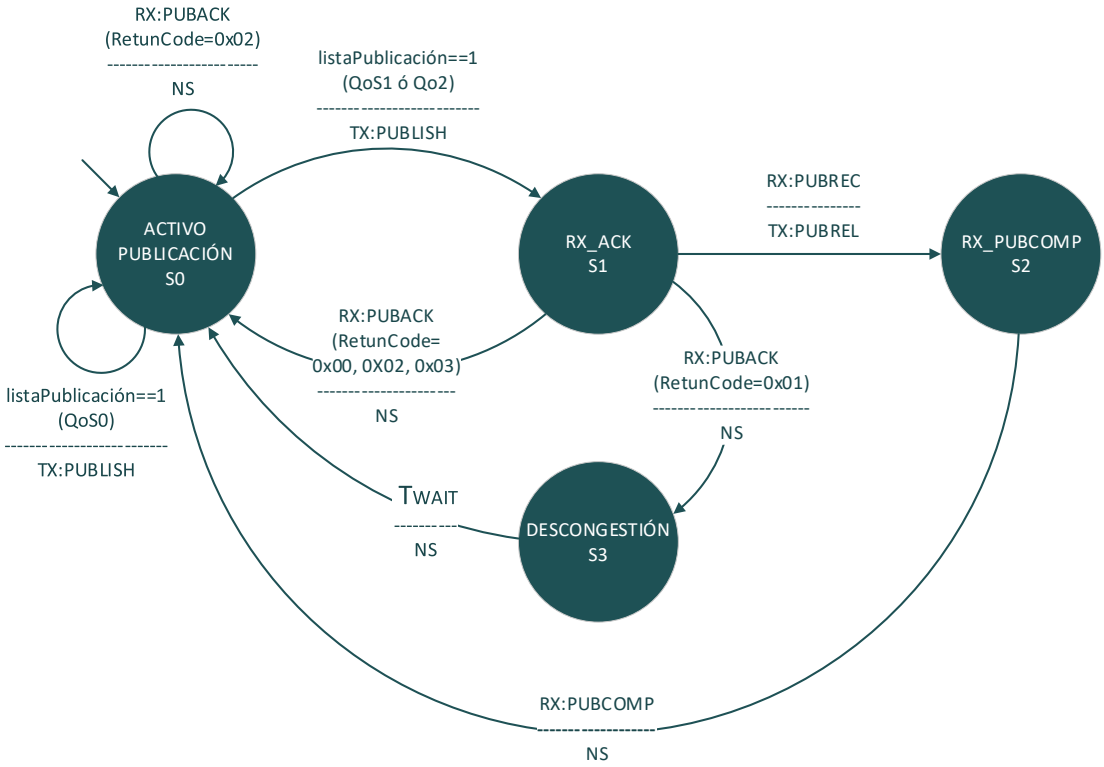


Figura 2.31 Diagrama de estados (Cliente): Procedimiento de publicación del cliente.

Forma normal

Se define una 6-tupla {Q, Σ, S, δ, λ, q0} donde:

Q= {ACTIVO PUBLICACIÓN(S0), RX_ACK(S1), RX_*PUBCOMP*(S2), DESCONGESTIÓN(S3)};

Σ= {listaPublicación(QoS0)=1, listaPublicación(QoS1)=1, listaPublicación(QoS2)=1, *PUBACK*(0x02), *PUBACK*(0x00,0x03), *PUBACK*(0x01), *PUBREC*, *PUBCOMP*, TWAIT};

S= {*PUBLISH*, *PUBREL*};

q0= ACTIVO PUBLICACIÓN

δ= Función de transición:

Tabla 2.19 Función de transición del cliente (Procedimiento de publicación del cliente).

ESTADO ACTUAL		**ESTADO SIGUIENTE**								
ENTRADA:		listaPub (QoS0) =1	listaPub (QoS1) =1	listaPub (QoS2) =1	PUBACK (0x02)	PUBACK (0x00,0x03)	PUBACK (0x01)	PUBREC	PUBCOMP	TWAIT
ACTIVO_PUB	**S0**	S0	S1	S1	S0	S0	S0	S0	S0	S0
RX_ACK	**S1**	S1	S1	S1	S0	S0	S3	S2	S1	S1
RX_PUBCOMP	**S2**	S2	S2	S2	S2	S2	S2	S2	S0	S2
DESCONGEST	**S3**	S3	S3	S3	S3	S3	S3	S3	S3	S0

λ: Función de salida:

Tabla 2.20 Función de salida del cliente (Procedimiento de publicación del cliente).

ESTADO ACTUAL		SALIDA								
ENTRADA:		listaPub (QoS0) =1	listaPub (QoS1) =1	listaPub (QoS2) =1	PUBACK (0x02)	PUBACK (0x00,0x03)	PUBACK (0x01)	PUBREC	PUBCOMP	TWAIT
ACTIVO_PUB	**S0**	PUBLISH	PUBLISH	PUBLISH	NS	NS	NS	NS	NS	NS
RX_ACK	**S1**	NS	NS	NS	NS	NS	NS	PUBREL	NS	NS
RX_PUBCOMP	**S2**	NS	NS	NS	NS	NS	NS	NS	NS	NS
DESCONGEST	**S3**	NS	NS	NS	NS	NS	NS	NS	NS	NS

2.5.7 PROCEDIMIENTO DE PUBLICACIÓN (QOS-1)

2.5.7.1 Gateway

En este procedimiento solo se definen dos estados INACTIVO Y ESPERA PUBLICACIÓN, ya que no se requiere de otros procedimientos para recibir este tipo de publicaciones. Durante el estado Inactivo el nodo espera la señal (*brokerListo=1*) que indica que el broker está listo para funcionar, para pasar al estado espera publicación. En el estado espera publicación se espera recibir mensajes *PUBLISH* con calidad de servicio QoS-1, sin que se envíe ningún tipo de confirmación. En caso de recibir la señal *brokerListo=0* por alguna falla en el bróker, el nodo regresara al estado inactivo. A continuación, la Figura 2.32 muestra el diagrama de estados de este procedimiento.

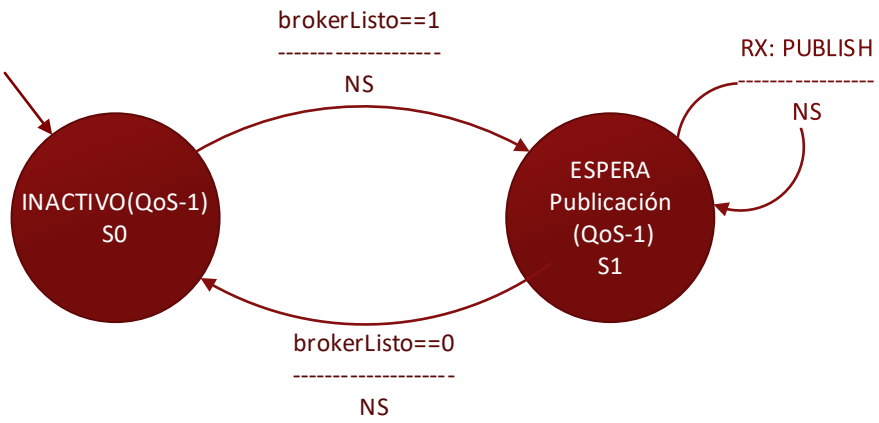


Figura 2.32 Diagrama de estados (Gateway): Procedimiento de publicación con QoS-1.

Forma normal

Se define una 5-tupla {Q, Σ, S, δ, q0} donde:

Q= {INACTIVO(S0), ESPERA Publicación(S1)};

Σ= {brokerListo=1, brokerListo=0, *PUBLISH*};

q0= INACTIVO

δ= Función de transición:

Tabla 2.21 Función de transición del gateway (Procedimiento de publicación con QoS-1).

ESTADO ACTUAL		**ESTADO SIGUIENTE**		
ENTRADA:		brokerListo=1	brokerListo=0	PUBLISH
INACTIVO	**S0**	S1	S0	S1
ESPERA Publicación	**S1**	S1	S0	S1

2.5.7.2 Cliente

De acuerdo al diagrama de estados de la Figura 2.33, un cliente durante este procedimiento solo permanecerá en estado ACTIVO PUBLICACIÓN, donde solo se espera la señal *listaPublicación(QoS-1)=1* para después enviar el mensaje *PUBLISH* correspondiente y luego regresar al mismo estado.

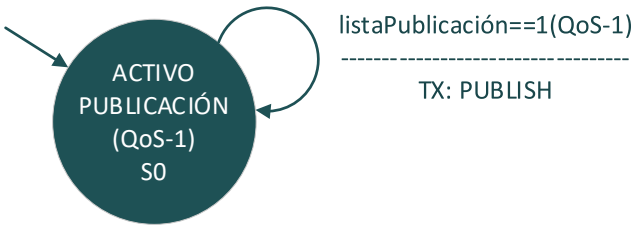


Figura 2.33 Diagrama de estados (Cliente): Procedimiento de publicación con QoS-1.

Forma normal

Se define una 6-tupla {Q, Σ, S, δ, λ, q0} donde:

Q= {INACTIVO(S0)};

Σ= {listaPublicación(QoS-1)=1};

S= {*PUBLISH*};

q0= INACTIVO

δ= Función de transición: S0 [listaPublicación(QoS-1)=1] = S0

λ: Función de salida: S0 [listaPublicación(QoS-1)=1] = *PUBLISH*

2.5.8 PROCEDIMIENTO PARA SUBSCRIBIRSE O DARSE DE BAJA DE UN TEMA

2.5.8.1 Gateway

En este procedimiento el nodo gateway requiere estar en un único estado**,** donde se mantiene esperando mensajes *SUBSCRIBE* para después enviar un mensaje *SUBACK* para indicar si la subscripción es aceptada o rechazada. También se puede recibir el

mensaje *UNSUBSCRIBE*, de un cliente que quiere darse de baja, que se responde con un *UNSUBACK*. Después de responder a los mensajes recibidos el nodo regresa al mismo estado, como se muestra en el diagrama de estados de la Figura 2.34.

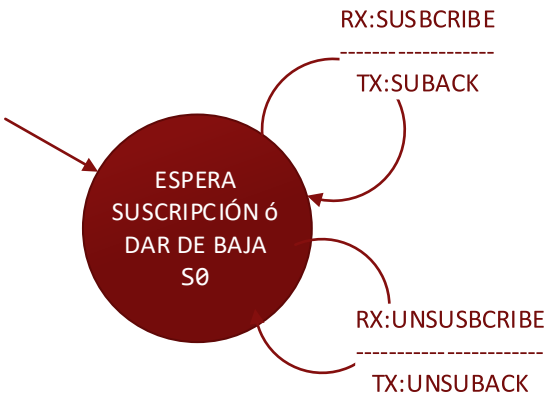


Figura 2.34 Diagrama de estados (Gateway): Procedimiento para subscribirse o darse de baja de un tema.

Forma normal

Se define una 6-tupla {Q, Σ, S, δ, λ, q0} donde:

Q= {ESPERA SUBSCRIPCIÓN o DAR DE BAJA(S0)};

Σ= {SUSBCRIBE, UNSUSBCRIBE};

S= {*SUBACK*, *UNSUBACK*}

q0= ESPERA SUBSCRIPCIÓN o DAR DE BAJA

δ= Función de transición: S0 [SUSBCRIBE] = S0; S0 [UNSUSBCRIBE] = S0

λ: Función de salida: S0 [SUSBCRIBE] = *SUBACK*; S0 [UNSUSBCRIBE] = *UNSUBACK*

2.5.8.2 Cliente

Como se mencionó antes, cuando un cliente ha establecido una conexión con un gateway se considera que esta en estado ACTIVO, por lo cual puede subscribirse a un tema en el que esté interesado, mediante este procedimiento.

Por lo anterior, en este procedimiento se considera que el nodo inicia en el estado denominado ACTIVO SUBSCRIPCIÓN O DAR DE BAJA, durante el cual se espera una señal que permita subscribirse o darse de baja de un tema. Si en el estado inicial se recibe la señal *listaSubscripción=1* se procede a enviar un mensaje *SUBSCRIBE*, para después pasar al estado *RX_SUBACK* donde espera un mensaje *SUBACK* enviado por el gateway. Cuando se recibe el mensaje *SUBACK* se revisa su campo ReturnCode. Si la subscripción es aceptada o rechazada por una razón diferente a la congestión, el nodo regresa al estado inicial para ejecutar otro procedimiento o reiniciar el procedimiento.

En el caso de existir un rechazo por congestión, el nodo cliente pasa al estado DESCONGESTIÓN en el cual permanecerá hasta que el temporizador *TWAIT* finalice. Después el nodo tendrá que regresar al estado inicial para intentar subscribirse otra vez.

Si en el estado inicial se recibe la señal darDeBaja=1 se procede a enviar un mensaje *UNSUBCRIBE*, para después pasar al estado *RX_UNSUBACK* donde espera un mensaje *USUBACK* enviado por el gateway. De esta forma un cliente se da de baja de un tema al que se haya subscrito previamente. En la Figura 2.35 se muestra el diagrama de estados de este procedimiento.

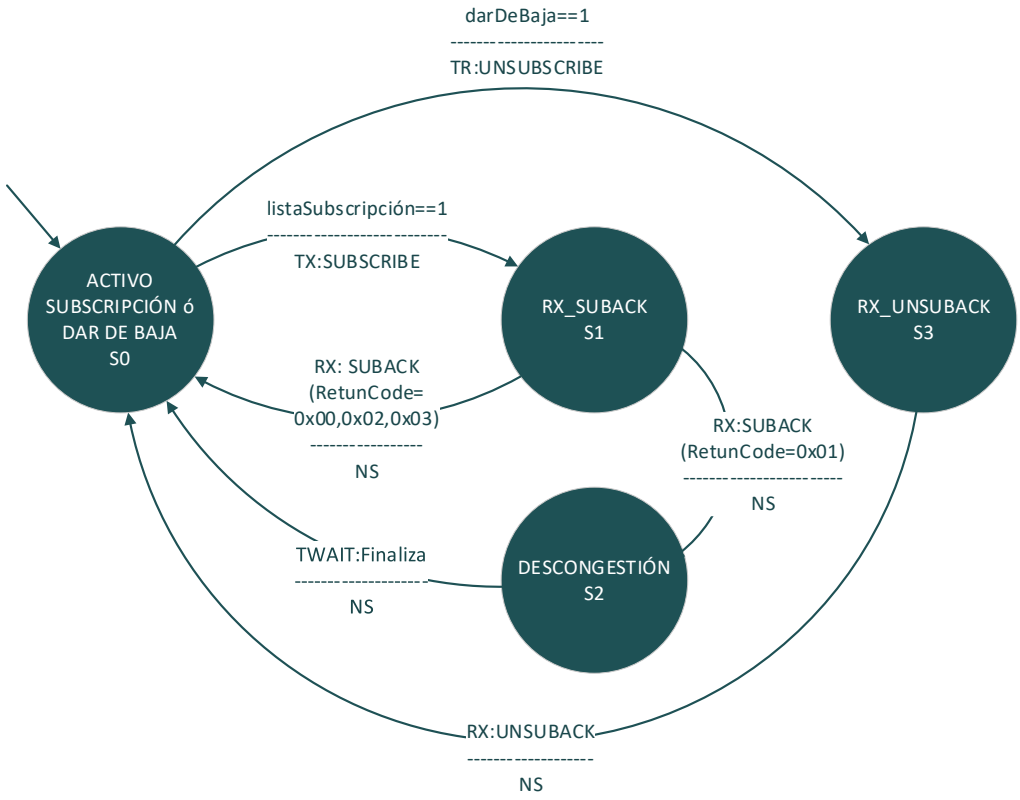


Figura 2.35 Diagrama de estados (Cliente): Procedimiento para subscribirse o darse de baja de un tema.

Forma normal

Se define una 6-tupla {Q, Σ, S, δ, λ, q0} donde:

Q= {ACTIVO SUBSCRIPCIÓN ó DAR DE BAJA(S0),
RX_*SUBACK* (S1), DESCONGESTIÓN(S2), RX_*UNSUBACK*(S3)};

Σ= {listaSubscripción=1, *SUBACK*(!=0x01), *SUBACK*(0x01), TWAIT,
darDeBaja=1, *UNSUBACK*};

S= {*SUBSCRIBE*, *UNSUBSCRIBE*};

q0= ACTIVO SUBSCRIPCIÓN ó DAR DE BAJA

δ= Función de transición:

Tabla 2.22 Función de transición del cliente (Procedimiento para subscribirse ó darse de baja de un tema).

ESTADO ACTUAL		**ESTADO SIGUIENTE**					
ENTRADA:		Lista Subscripción=1	SUBACK (!=0x01)	SUBACK (0x01)	TWAIT	darDe Baja=1	UNSUBACK
ACTIVO SUB o DAR DE BAJA	**S0**	S1	S0	S0	S0	S3	S0
RX_SUBACK	**S1**	S1	S0	S2	S1	S1	S1
DESCONGEST	**S2**	S2	S2	S2	S0	S2	S2
RX_UNSUBACK	**S3**	S3	S3	S3	S3	S3	S0

λ: Función de salida:

Tabla 2.23 Función de salida del cliente (Procedimiento para subscribirse ó darse de baja de un tema).

ESTADO ACTUAL		**SALIDA**					
ENTRADA:		Lista Subscripción=1	SUBACK (!=0x01)	SUBACK (0x01)	TWAIT	darDeBaja=1	UNSUBACK
ACTIVO SUB o DAR DE BAJA	**S0**	SUBSCRIBE	NS	NS	NS	UNSUBSCRIBE	NS
RX_SUBACK	**S1**	NS	NS	NS	NS	NS	NS
DESCONGEST	**S2**	NS	NS	NS	NS	NS	NS
RX_UNSUBACK	**S3**	NS	NS	NS	NS	NS	NS

2.5.9 PROCEDIMIENTO DE PUBLICACIÓN(GATEWAY)

2.5.9.1 Gateway

El diagrama de estados de este procedimiento, para el nodo gateway, es bastante similar al del procedimiento de publicación para el nodo cliente (literal 2.5.6). Sin embargo, el nodo debe verificar primero si tiene publicaciones pendientes antes de alistar dichas publicaciones para ser enviadas. Por esto, el nodo inicia en el estado Espera en donde como su nombre indica espera la señal *publicacionesPendientes>=1* para de esta forma pasar al estado PUBLICACIÓN. A partir del segundo estado el nodo gateway se comporta igual que un cliente en el procedimiento del literal 2.5.6, con la notable diferencia de que cuando se termine de enviar todas las publicaciones pendientes el nodo regresa al estado inicial. Se necesita de la señal *publicacionesPendientes=0* para retornar al estado Espera e iniciar otro procedimiento de ser necesario. A continuación, la Figura 2.36 muestra el diagrama de estados de este procedimiento.

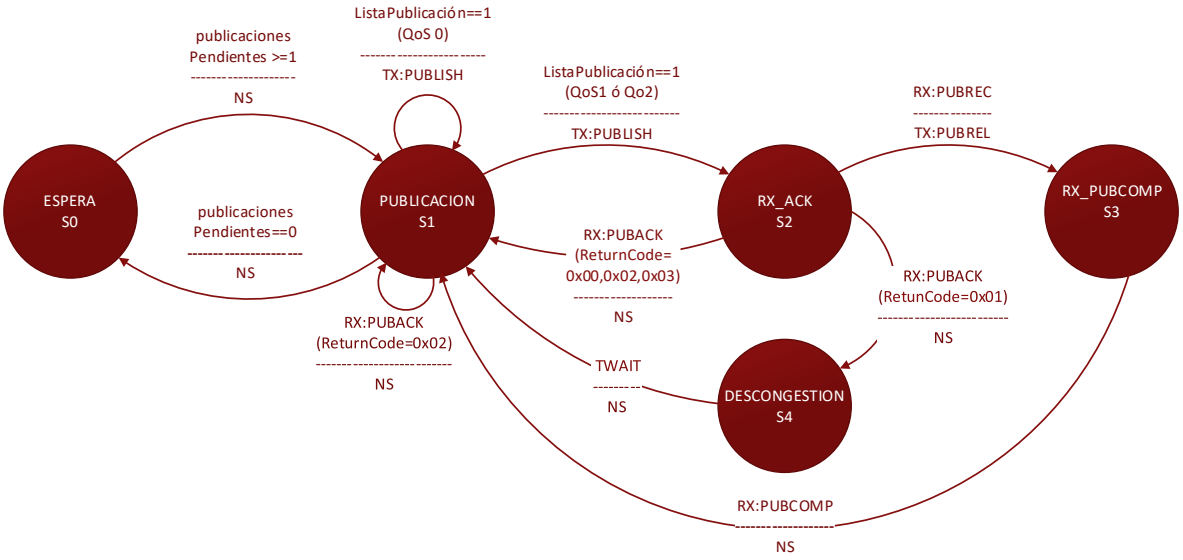


Figura 2.36 Diagrama de estados (Gateway): Procedimiento de publicación del gateway.

Forma normal

Se define una 6-tupla {Q, Σ, S, δ, λ, q0} donde:

Q= {ESPERA(S0), PUBLICACIÓN(S1), RX_ACK(S2), RX_*PUBCOMP*(S3), DESCONGESTIÓN(S4)};

Σ= {publicacionesPendientes>=1, publicacionesPendientes=0, listaPublicación(QoS0)=1, listaPublicación(QoS1 o QoS2)=1, *PUBACK*(0x02), *PUBACK*(0x00,0x03), *PUBACK*(0x01), *PUBREC*, *PUBCOMP*, TWAIT};

S= {*PUBLISH*, *PUBREL*};

q0= ESPERA

δ= Función de transición:

Tabla 2.24 Función de transición del gateway (Procedimiento de publicación del gateway).

ESTADO ACTUAL		**ESTADO SIGUIENTE**										
ENTRADA:		publicaciones Pendientes>=1	publicaciones Pendientes=0	listaPub (QoS0) =1	listaPub (QoS1)=1	listaPub (QoS2) =1	PUBACK (0x02)	PUBACK (0x00,0x03)	PUBACK (0x01)	PUBREC	PUBCOMP	TWAIT
ESPERA	**S0**	S1	S0	S0	S0	S0	S0	S0	S0	S0	S0	S0
PUBLICACION	**S1**	S1	S0	S1	S2	S2	S1	S1	S1	S1	S1	S1
RX_ACK	**S2**	S2	S2	S2	S2	S2	S1	S1	S4	S3	S2	S2
RX_PUBCOMP	**S3**	S3	S3	S3	S3	S3	S3	S3	S3	S3	S1	S3
DESCONGEST	**S4**	S4	S4	S4	S4	S4	S4	S4	S4	S4	S4	S1

λ: Función de salida:

Tabla 2.25 Función de salida del gateway (Procedimiento de publicación del gateway).

ESTADO ACTUAL			**SALIDA**									
ENTRADA:		publicaciones Pendientes>=1	publicaciones Pendientes=0	listaPub (QoS0) =1	listaPub (QoS1)=1	listaPub (QoS2) =1	PUBACK (0x02)	PUBACK (0x00,0x03)	PUBACK (0x01)	PUBREC	PUBCOMP	TWAIT
ESPERA	**S0**	NS	NS	PUBLISH	PUBLISH	PUBLISH	NS	NS	NS	NS	NS	NS
PUBLICACION	**S1**	NS	NS	NS	NS	NS	NS	NS	NS	NS	NS	NS
RX_ACK	**S2**	NS	NS	NS	NS	NS	NS	NS	NS	PUBREL	NS	NS
RX_PUBCOMP	**S3**	NS	NS	NS	NS	NS	NS	NS	NS	NS	NS	NS
DESCONGEST	**S4**	NS	NS	NS	NS	NS	NS	NS	NS	NS	NS	NS

2.5.9.2 Cliente

Durante este procedimiento el nodo cliente se comporta igual que el nodo gateway del procedimiento del literal 2.5.6. Lo único que cambia es el nombre del estado inicial que ahora será denominado como Activo Espera Publicación, ya que el cliente debe estar activo para que dé inicio a este procedimiento. Desde el estado inicial se atenderá a mensajes *PUBLISH*, con su respectivo nivel de QoS, enviados por el gateway, como se muestra en el diagrama de estados de la Figura 2.37.

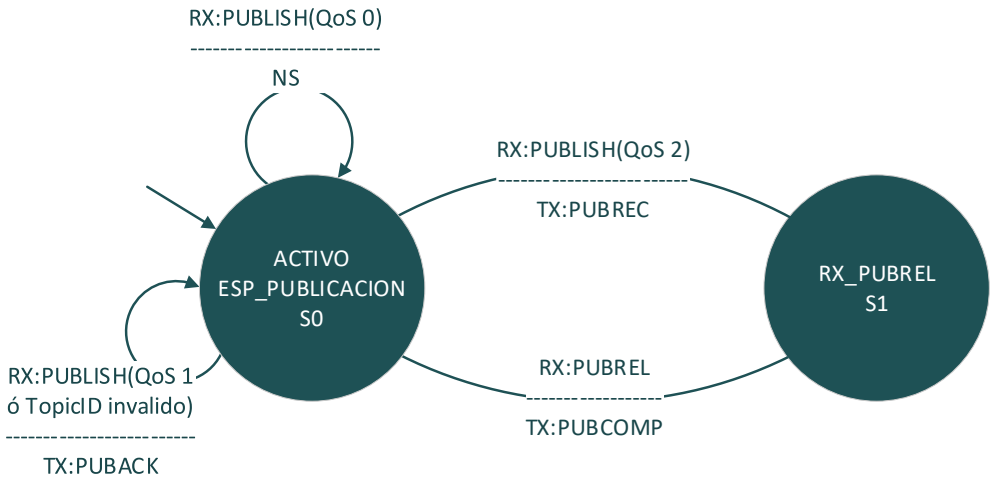


Figura 2.37 Diagrama de estados (Cliente): Procedimiento de publicación del gateway.

Forma normal

Se define una 6-tupla {Q, Σ, S, δ, λ, q0} donde:

Q= {ACTIVO ESPERA PUBLICACIÓN(S0), RX_*PUBREL*(S1)};

Σ= {*PUBLISH*(QoS0), *PUBLISH*(QoS1), *PUBLISH* (ID invalido), *PUBLISH*(QoS2), *PUBREL*};

S= {*PUBACK*, *PUBREC*, *PUBCOMP*};

q0= ACTIVO ESPERA PUBLICACIÓN

δ= Función de transición:

Tabla 2.26 Función de transición del cliente (Procedimiento de publicación del gateway).

ESTADO ACTUAL		**ESTADO SIGUIENTE**				
ENTRADA:		PUBLISH (QoS0)	PUBLISH (QoS1)	PUBLISH (QoS2)	PUBLISH (ID invalido)	PUBREL
ACT_ESP_PUBLICACIÓN	**S0**	S0	S0	S1	S0	S0
RX_PUBREL	**S1**	S1	S1	S1	S1	S0

λ: Función de salida:

Tabla 2.27 Función de salida del cliente (Procedimiento de publicación del gateway).

ESTADO ACTUAL		SALIDA				
ENTRADA:		PUBLISH (QoS0)	PUBLISH (QoS1)	PUBLISH (QoS2)	PUBLISH (ID invalido)	PUBREL
ACT_ESP_PUBLICACIÓN	**S0**	NS	PUBACK	PUBREC	PUBACK	NS
RX_PUBREL	**S1**	NS	NS	NS	NS	PUBCOMP

2.5.10 PROCEDIMIENTO KEEP ALIVE Y PING

2.5.10.1 Gateway

Este procedimiento KEEP ALIVE y PING es utilizado para conocer si el cliente o gateway están funcionando con normalidad.

Durante este procedimiento el nodo gateway solo puede permanecer en dos estados, el estado inicial Espera y el estado RX_*PINGRESP*. Cuando el nodo se encuentra en estado inicial, tiene la capacidad de recibir un mensaje *PINGREQ* a lo cual se responde de inmediato con un mensaje *PINGREQ*. También se puede recibir la señal que indica que el temporizador *TKA* ha finalizado, debido a que el cliente ha dejado de enviar su mensaje *PINGREQ*. El nodo pasa a estado RX_*PINGRESP* mediante la señal *listaPeticion=1*, cuando quiere verificar si el nodo cliente se encuentra funcionando. En el segundo estado el nodo solo espera la respuesta del cliente para retornar al estado inicial, como se muestra en el diagrama de estados de la Figura 2.38.

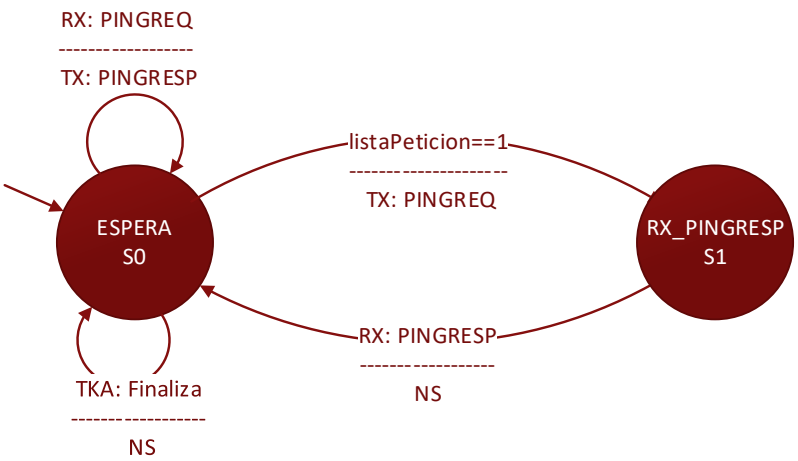


Figura 2.38 Diagrama de estados (Gateway): Procedimiento keep alive y PING.

Forma normal

Se define una 6-tupla {Q, Σ, S, δ, λ, q0} donde:

Q= {ESPERA (S0), RX_*PINGRESP*(S1)};

Σ= {*PINGREQ*, TKA, listaPeticion=1, *PINGRESP*};

S= {*PINGRESP*, *PINGREQ*};

q0= ESPERA

δ= Función de transición:

Tabla 2.28 Función de transición del Gateway (Procedimiento keep alive y PING).

ESTADO ACTUAL		**ESTADO SIGUIENTE**			
ENTRADA:		PINGREQ	TKA	listaPeticion=1	PINGRESP
ESPERA	**S0**	S0	S0	S1	S0
RX_PINGRESP	**S1**	S1	S1	S1	S0

λ: Función de salida:

Tabla 2.29 Función de salida del Gateway (Procedimiento keep alive y PING).

ESTADO ACTUAL		**SALIDA**			
ENTRADA:		PINGREQ	TKA	listaPeticion=1	PINGRESP
ESPERA	**S0**	PINGRESP	NS	PINGREQ	NS
RX_PINGRESP	**S1**	NS	NS	NS	NS

2.5.10.2 Cliente

En este procedimiento también se manejan dos estados Activo supervisión y RX_*PINGRESP*. En el estado inicial el nodo espera a que finalice el temporizador TKA para después enviar el mensaje *PINGREQ*. La respuesta al mensaje enviado se la espera en el estado RX_*PINGRESP*, una vez que el nodo recibe el mensaje *PINGRESP* regresa al estado inicial, por lo cual el nodo considera que el gateway está operando con normalidad. El estado inicial también se debe responder a un mensaje *PINGREQ* enviado por el gateway. En la Figura 2.39 se muestra el diagrama de estados de este procedimiento.

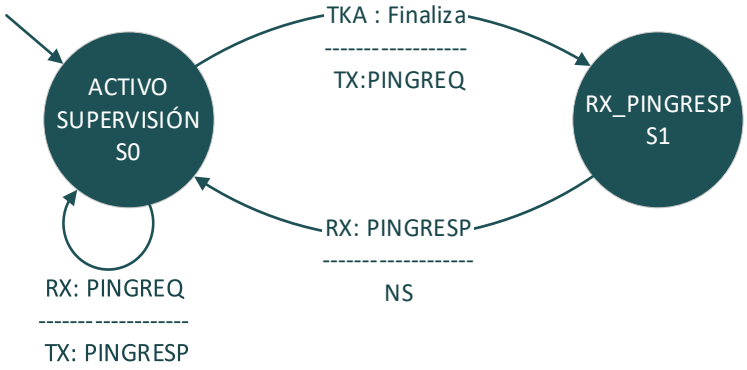


Figura 2.39 Diagrama de estados (Cliente): Procedimiento KEEP ALIVE y PING.

Forma normal

Se define una 6-tupla {Q, Σ, S, δ, λ, q0} donde:

Q= {ACTIVO SUPERVISIÓN (S0), RX_*PINGRESP*(S1)};

Σ= {TKA, *PINGRESP*, *PINGREQ*};

S= {*PINGREQ*, *PINGRESP*};

q0= ACTIVO SUPERVISIÓN

δ= Función de transición:

Tabla 2.30 Función de transición del cliente (Procedimiento keep alive y PING).

ESTADO ACTUAL		**ESTADO SIGUIENTE**		
ENTRADA:		TKA	PINGRESP	PINGREQ
ACT_SUPERVISIÓN	**S0**	S1	S0	S0
RX_PINGRESP	**S1**	S1	S0	S1

λ: Función de salida:

Tabla 2.31 Función de salida del cliente (Procedimiento keep alive y PING).

ESTADO ACTUAL		**SALIDA**		
ENTRADA:		TKA	PINGRESP	PINGREQ
ACT_SUPERVISIÓN	**S0**	PINGREQ	NS	PINGRESP
RX_PINGRESP	**S1**	NS	NS	NS

2.5.11 PROCEDIMIENTO DE DESCONEXIÓN DEL CLIENTE

2.5.11.1 Gateway

Durante este estado el nodo solo debe mantenerse en un único estado denominado Espera desconexión**.** En este estado el gateway espera por un mensaje *DISCONNECT* enviado por el cliente cuando desee desconectarse. El mensaje es respondido con otro mensaje *DISCONNECT* para a continuación regresar al estado inicial. También se puede enviar un mensaje *DISCONNECT* cuando se recibe la señal *clienteDesconocido=1*, esta señal debe ser establecida cuando un mensaje recibido no puede ser identificado por el gateway. A continuación, la Figura 2.40 muestra el diagrama de estados de este procedimiento.

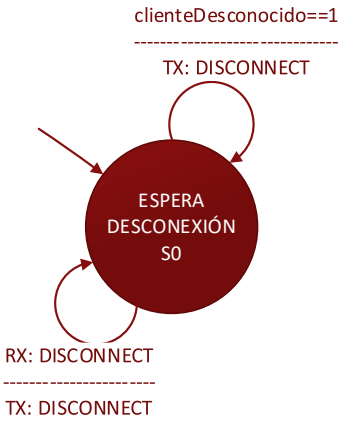


Figura 2.40 Diagrama de estados (Gateway): Procedimiento de desconexión del cliente.

Forma normal

Se define una 6-tupla {Q, Σ, S, δ, λ, q0} donde:

Q= {ESPERA DESCONEXIÓN (S0)};

Σ= {clienteDesconocido=1, *DISCONNECT*};

S= {*DISCONNECT*}

q0= ESPERA DESCONEXIÓN

δ= Función de transición: S0[clienteDesconocido=1]= S0; S0[*DISCONNECT*]= S0

λ: Función de salida: S0[clienteDesconocido=1]=*DISCONNECT*; S0[*DISCONNECT*]=*DISCONNECT*

2.5.11.2 Cliente

Un cliente debe estar activo para ejecutar este procedimiento. Debido a lo anterior, el nodo cliente inicia el procedimiento en el estado denominado Activo Desconexión. En el estado inicial si el cliente requiere desconectarse debe recibir una señal *cerraConexión=1* para a continuación transmitir un mensaje *DISCONNECT*. Ya que el cliente requiere que el gateway confirme su desconexión debe pasar al estado RX_DIS*CONNECT*. Cuando es recibida la confirmación de desconexión el nodo por fin llegara al estado desconectado. El cliente permanecerá en este estado hasta que se desee establecer conexión nuevamente mediante la señal *iniConexión=1*.

También, desde el estado inicial se puede recibir un mensaje *DISCONNECT* el cual indica que el cliente no puede ser reconocido. Recibido el mensaje *DISCONNECT* el nodo pasara al estado Establecer conexión que hace referencia al estado que se maneja en el procedimiento configuración de conexión del cliente del literal 2.5.3. Debido a que si un cliente recibe un mensaje *DISCONNECT* deberá intentar conectarse al gateway nuevamente. A continuación, la Figura 2.41 muestra el diagrama de estados de este procedimiento.

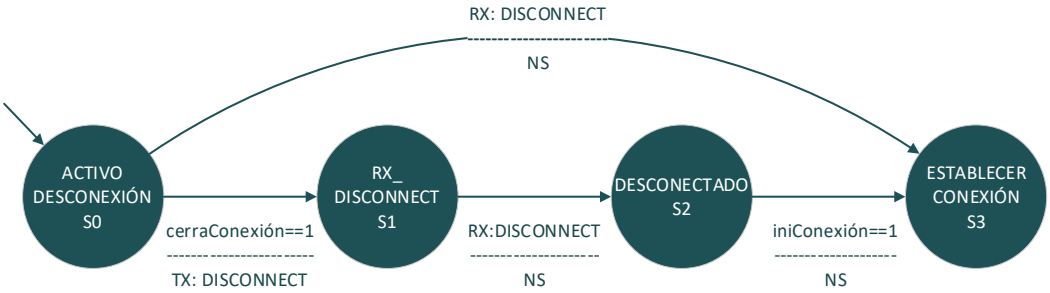


Figura 2.41 Diagrama de estados (Cliente): Procedimiento de desconexión del cliente.

Forma normal

Se define una 6-tupla {Q, Σ, S, δ, λ, q0} donde:

Q= {ACTIVO DESCONEXIÓN(S0), RX_*DISCONNECT*(S1), DESCONECTADO(S2), ESTABLECER CONEXIÓN(S3)};

Σ= {cerraConexión=1, *DISCONNECT*, iniConexión=1};

S= {*DISCONNECT*};

q0= ACTIVO DESCONEXIÓN

δ= Función de transición:

Tabla 2.32 Función de transición del cliente (Procedimiento de desconexión del cliente).

ESTADO ACTUAL		**ESTADO SIGUIENTE**		
ENTRADA:		cerraConexión=1	DISCONNECT	iniConexión=1
ACT_DESCONEXIÓN	**S0**	S1	S3	S0
RX_DISCONNECT	**S1**	S1	S2	S1
DESCONECTADO	**S2**	S2	S2	S3
EST_CONEXIÓN	**S3**	S3	S3	S3

λ: Función de salida:

Tabla 2.33 Función de salida del cliente (Procedimiento de desconexión del cliente).

ESTADO ACTUAL		**SALIDA**		
ENTRADA:		cerraConexión=1	DISCONNECT	iniConexión=1
ACT_DESCONEXIÓN	**S0**	DISCONNECT	NS	NS
RX_DISCONNECT	**S1**	NS	NS	NS
DESCONECTADO	**S2**	NS	NS	NS
EST_CONEXIÓN	**S3**	NS	NS	NS

2.5.12 PROCEDIMIENTO DE RETRANSMISIÓN DEL CLIENTE

2.5.12.1 Cliente

Este procedimiento solo es requerido por el cliente**.** Cuando el nodo se encuentra en estado inicial, al cual se lo llamara Activo, puede enviar un mensaje unicast al cliente una vez que se reciba la señal correspondiente (en este caso se espera la señal *listoMsgUnicast=1*). Si el mensaje enviado requiere una confirmación el nodo pasa al estado RX_ACK, en donde si el nodo recibe el mensaje esperado debe retornar al estado inicial. Si el cliente regresa al estado Activo podrá continuar con otros procedimientos. Por el contrario, si no se recibe el mensaje de confirmación se debe esperar a que el temporizador TRETRY finalice y retransmita el mensaje unicast enviado. El nodo regresa al estado RX_ACK hasta que reciba el mensaje de confirmación o en su defecto alcance un número máximo de reintentos y deba pasar al estado Establecer conexión mediante la señal *Nretry=Nmax*. Si el estado llega al estado S2 debe iniciar el procedimiento configuración de conexión del cliente del literal 2.5.3. A continuación, la Figura 2.42 muestra el diagrama de estados de este procedimiento.

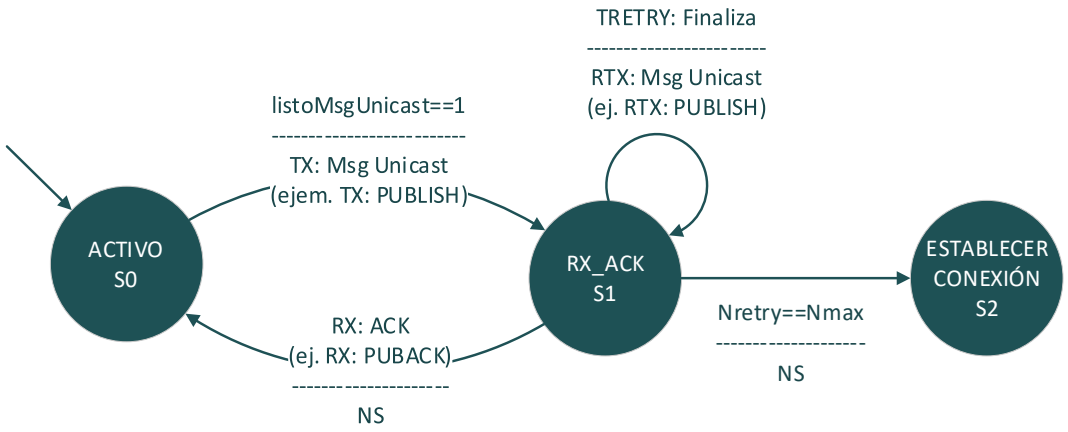


Figura 2.42 Diagrama de estados (Cliente): Procedimiento de retransmisión del cliente.

Forma normal

Se define una 6-tupla {Q, Σ, S, δ, λ, q0} donde:
Q= {ACTIVO(S0), RX_ACK(S1), ESTABLECER CONEXIÓN(S2)};
Σ= {listoMsgUnicast=1, ACK, Tretry, Nretry=Nmax};
S= {MsgUnicast};
q0= ACTIVO
δ= Función de transición:

Tabla 2.34 Función de transición del cliente (Procedimiento de retransmisión del cliente).

ESTADO ACTUAL		ESTADO SIGUIENTE			
ENTRADA:		listoMsg Unicast=1	ACK	Tretry	Nretry =Nmax
ACTIVO	**S0**	S1	S0	S0	S0
RX_ACK	**S1**	S1	S0	S1	S2
EST_CONEXIÓN	**S2**	S2	S2	S2	S2

λ: Función de salida:

Tabla 2.35 Función de salida del cliente (Procedimiento de retransmisión del cliente).

ESTADO ACTUAL		SALIDA			
ENTRADA:		listoMsg Unicast=1	ACK	Tretry	Nretry =Nmax
ACTIVO	**S0**	MsgUnicast	NS	NS	NS
RX_ACK	**S1**	NS	NS	MsgUnicast	NS
EST_CONEXIÓN	**S2**	NS	NS	NS	NS

2.5.13 SOPORTE PARA AHORRO DE ENERGÍA

2.5.13.1 Gateway

Para que el gateway brinde soporte a clientes que requieran ahorro de energía, el nodo debe manejar dos estados: espera supervisión (estado inicial) y publicación. En el estado de espera supervisión el nodo espera recibir diferentes señales de tal forma que puede conocer el estado en que se encuentra el cliente.

Si el gateway recibe un mensaje *DISCONNECT* (que incluya o no el campo duración) cuando se encuentra en el estado inicial, envía un mensaje *DISCONNECT* para confirmar la recepción. Una vez que el nodo haya recibido un mensaje *DISCONNECT* que incluya la duración del tiempo en que el cliente permanecerá dormido, también se podrá recibir la señal del temporizador de sueño *Ts*. De esta manera el nodo gateway podrá monitorear al cliente cuando se encuentra en estado dormido o perdido. No se toma en cuenta el estado activo del cliente porque ya es mencionado en el procedimiento de conexión del cliente (literal 2.5.3).

El nodo pasa a estado Publicación si en el estado inicial recibe un mensaje *PINGREQ*, a partir de esta recepción se considera que el cliente se encuentra despierto y se puede enviar o recibir publicaciones pendientes (mediante el procedimiento de publicación correspondiente). Si no existen publicaciones pendientes el nodo recibe la señal *publicacionesPendientes=0* y de inmediato se envía un mensaje *PINGRESP*, se considera que el cliente se encuentra dormido y el nodo gateway regresa al estado inicial donde continúa supervisando el estado del cliente. En la Figura 2.43 se muestra el diagrama de estados de este procedimiento.

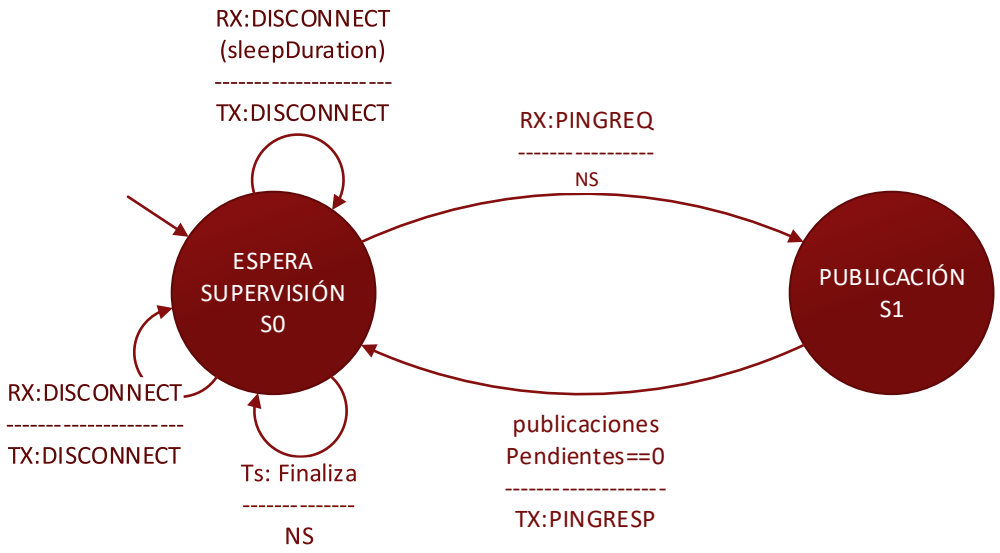


Figura 2.43 Diagrama de estados (Gateway): Soporte para ahorro de energía.

Forma normal

Se define una 6-tupla {Q, Σ, S, δ, λ, q0} donde:

Q= {ESPERA SUPERVISIÓN(S0), PUBLICACIÓN(S1)};

Σ= {*DISCONNECT*, *DISCONNECT* (sleepDuration), Ts, *PINGREQ*,
publicacionesPendientes=0};

S= {*DISCONNECT*, *PINGRESP*};

q0= ESPERA SUPERVISIÓN

δ= Función de transición:

Tabla 2.36 Función de transición del gateway (Soporte para ahorro de energía).

ESTADO ACTUAL		**ESTADO SIGUIENTE**				
ENTRADA:		DISCONNECT	DISCONNECT (sleepDuration)	Ts	PINGREQ	Publicaciones Pendientes=0
ESPERA SUPERVISIÓN	**S0**	S0	S0	S0	S1	S0
PUBLICACIÓN	**S1**	S1	S1	S1	S1	S0

λ: Función de salida:

Tabla 2.37 Función de salida del gateway (Soporte para ahorro de energía).

ESTADO ACTUAL		**SALIDA**				
ENTRADA:		DISCONNECT	DISCONNECT (sleepDuration)	Ts	PINGREQ	Publicaciones Pendientes=0
ESPERA SUPERVISIÓN	**S0**	DISCONNECT	DISCONNECT	NS	NS	NS
PUBLICACIÓN	**S1**	NS	NS	NS	NS	PINGRESP

2.5.13.2 Cliente

Cuando un nodo cliente tiene la capacidad de ahorrar energía puede entrar en diferente estado, por lo cual, se considera para este nodo un diagrama con los siguientes estados: activo, dormido, despierto, desconectado, confirmar sueño, RX_DIS*CONNECT*, y establecer conexión.

Desde el estado inicial el nodo puede recibir las señales *ahorrarEnergía=1* o *cerrarConexión=1* y enviar un mensaje *DISCONNECT,* si se desea pasar el estado confirmar sueño o RX_*DISCONNECT* respectivamente. Hay que tener en cuenta que el mensaje *DISCONNECT* enviado cuando se requiere ahorrar energía debe tener el campo Duration el cual indica el tiempo que el nodo permanecerá dormido.

Si desde el estado confirmar sueño el nodo recibe un mensaje *DISCONNECT*, pasara a estado dormido. Durante el estado dormido el nodo debe esperar a que el timer que controla la duración del sueño *Ts* finalice para pasar al estado despierto después de enviar el mensaje *PINGREQ*, en el estado despierto el nodo puede recibir o enviar publicaciones hasta recibir el mensaje *PINGRESP* del gateway. A continuación, el nodo regresa al estado dormido a esperar que el temporizador *Ts* finalice nuevamente.

Durante el estado dormido el nodo también puede modificar la duración del sueño enviando otro mensaje *DISCONNECT*, el gateway debe confirmar la nueva duración del temporizador *Ts*.

Tambien, el nodo puede regresar al estado activo o estado desconectado al recibir las señales *ahorrarEnergía=0* o *cerrarConexión=1*. Para regresar al estado activo primero se debe pasar por el estado establecer conexión, donde se debe ejecutar un procedimiento para establecer conexión (literal 2.5.3). Si la conexión es aceptada el nodo regresa al estado activo nuevamente, por lo cual durante el estado establecer conexión se debe recibir la señal *proc2Count=1*.

Finalmente, para pasar al estado desconectado el nodo debe recibir un mensaje *DISCONNECT* de confirmación (enviado por el gateway) durante el estado RX_DIS*CONNECT*. Si se desea regresar al estado activo el nodo debe iniciar una nueva conexión mediante la señal *iniConexión=1*. En la Figura 2.44 se muestra el diagrama de estados de este procedimiento.

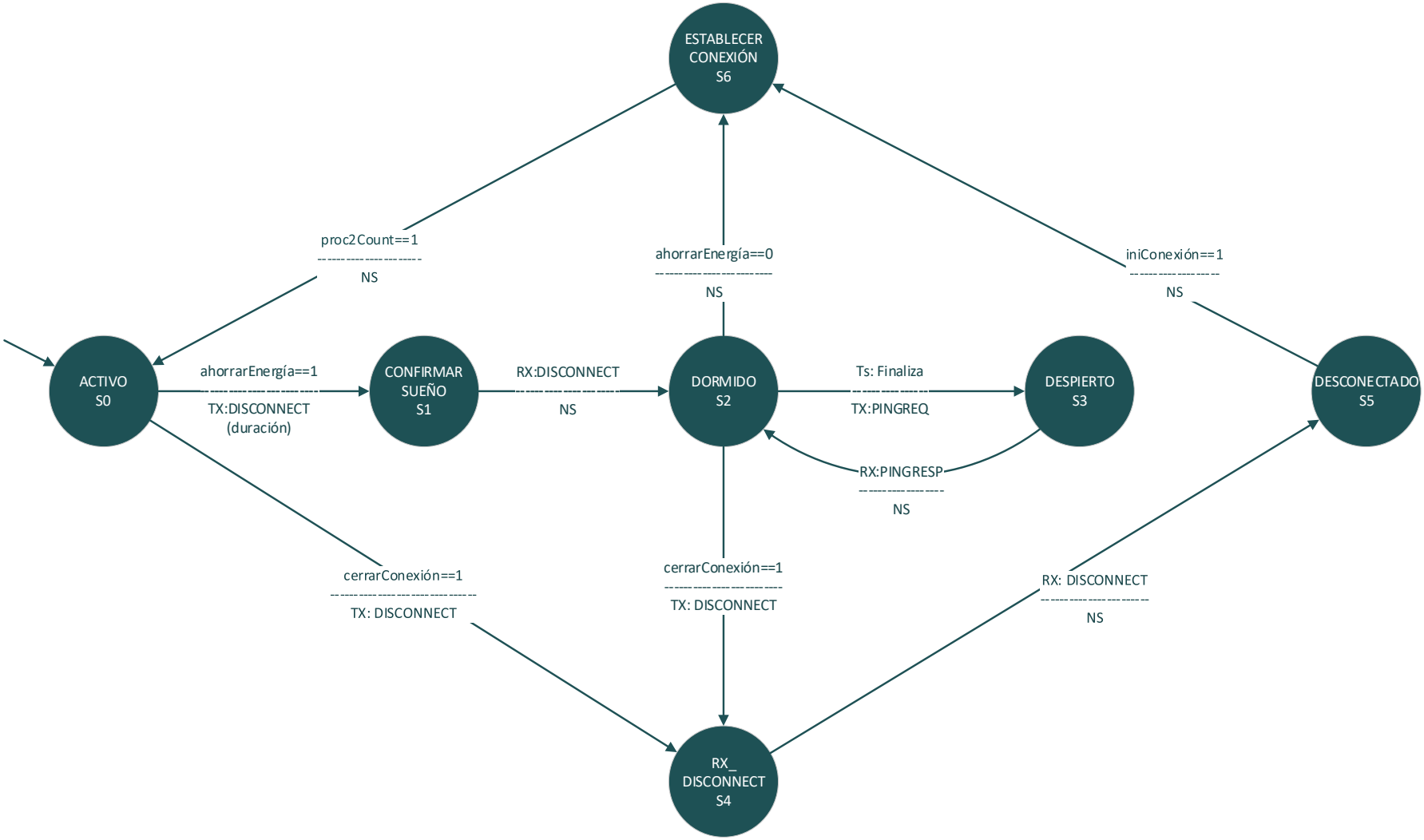


Figura 2.44 Diagrama de estados (Cliente): Soporte para ahorro de energía.

Forma normal

Se define una 6-tupla {Q, Σ, S, δ, λ, q0} donde:

Q= {ACTIVO(S0), CONFIRMAR SUEÑO(S1), DORMIDO(S2), DESPIERTO(S3),
RX_*DISCONNECT*(S4), DESCONECTADO(S5), ESTABLECER CONEXIÓN(S6)};

Σ= {ahorrarEnergía=1, *DISCONNECT*, Ts, *PINGRESP*, ahorrarEnergía=0,
cerrarConexión=1, iniConexión=1, proc2Count=1};

S= {DISCONECCT, *DISCONNECT* (sleepDuration), *PINGREQ*};

q0= ACTIVO

δ= Función de transición:

Tabla 2.38 Función de transición del cliente (Soporte para ahorro de energía).

ESTADO ACTUAL		**ESTADO SIGUIENTE**							
ENTRADA:		ahorrar Energía=1	DISCONNECT	Ts	PINGRESP	ahorrar Energía=0	Cerrar Conexión=1	ini Conexión=1	proc2 Count=1
ACTIVO	**S0**	S1	S0	S0	S0	S0	S4	S0	S0
CONFIRMAR SUEÑO	**S1**	S1	S2	S1	S1	S1	S1	S1	S1
DORMIDO	**S2**	S1	S2	S3	S2	S6	S4	S2	S2
DESPIERTO	**S3**	S3	S3	S3	S2	S3	S3	S3	S3
RX_DISCONNECT	**S4**	S4	S5	S4	S4	S4	S4	S4	S4
DESCONECTADO	**S5**	S5	S5	S5	S5	S5	S5	S6	S5
ESTABLECER CONEXIÓN	**S6**	S6	S6	S6	S6	S6	S6	S6	S0

λ: Función de salida:

Tabla 2.39 Función de salida del cliente (Soporte para ahorro de energía).

ESTADO ACTUAL		**SALIDA**							
ENTRADA:		ahorrar Energía=1	DISCONNECT	Ts	PINGRESP	ahorrar Energía=0	Cerrar Conexión=1	ini Conexión=1	proc2 Count=1
ACTIVO	**S0**	DISCONNECT (duración)	NS	NS	NS	NS	DISCONNECT	NS	NS
CONFIRMAR SUEÑO	**S1**	NS	NS	NS	NS	NS	NS	NS	NS
DORMIDO	**S2**	DISCONNECT (duración)	NS	PINGREQ	NS	NS	DISCONNECT	NS	NS
DESPIERTO	**S3**	NS	NS	NS	NS	NS	NS	NS	NS
RX_DISCONNECT	**S4**	NS	NS	NS	NS	NS	NS	NS	NS
DESCONECTADO	**S5**	NS	NS	NS	NS	NS	NS	NS	NS
ESTABLECER CONEXIÓN	**S6**	NS	NS	NS	NS	NS	NS	NS	NS

2.6 LENGUAJE DE ESPECIFICACIÓN Y DESCRIPCIÓN

El Lenguaje de Especificación y Descripción o SDL está basado en máquinas de estados finitos, fue desarrollado originalmente, para especificar y describir el comportamiento funcional de sistemas de telecomunicaciones [35]. Este lenguaje permite describir la estructura, comunicación, comportamiento y los datos de un sistema. SDL también, es utilizado como la especificación de protocolos de comunicación de datos [36].

Se puede seleccionar entre dos tipos de notaciones SDL: SDL/GR (Representación gráfica -Graphical Representation) y SDL/PR (Representación Textual -Phrase Representation) [35]. En este proyecto solo se utiliza la notación SDL/GR.

- **Entidades:** SDL están compuestos por entidades organizados de manera jerárquica. La entidad principal es el sistema, el cual está compuesto por bloques. Los cuales están acoplados unos con otros y con el entorno a través de canales, los cuales transportan señales.

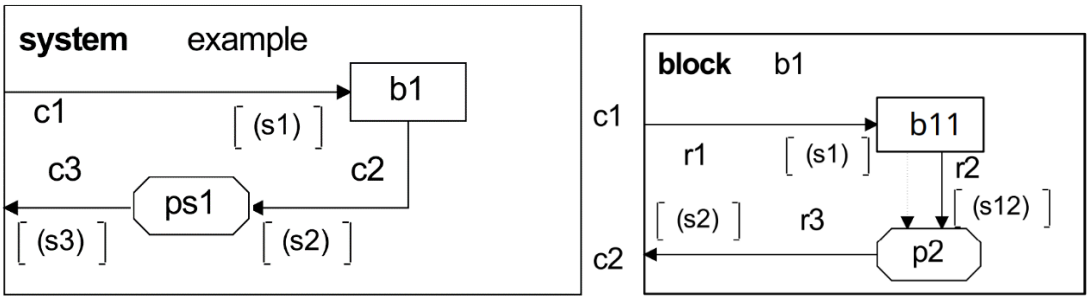
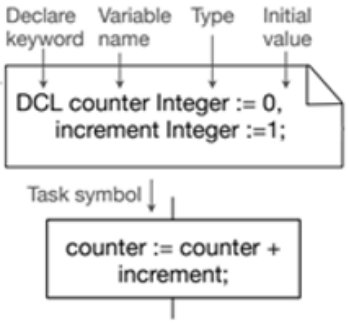
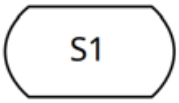
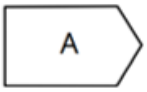
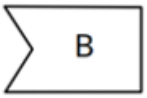
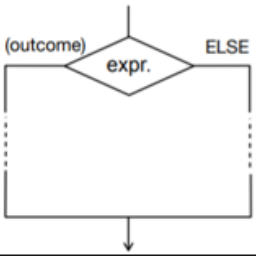
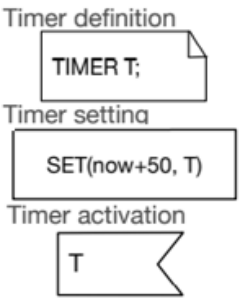


Figura 2.45 Ejemplo de un Sistema (entidad principal) y un Bloque [36].

Dentro de los bloques tenemos procesos, que están definidos por una máquina de estados finitos extendida, con variables, temporizadores y acciones. Los procesos también se comunican entre sí por medio de señales. Un bloque puede contener varias entidades procesos y entidades bloques [36].

- **Entidad proceso:** Se comporta como una MEF, la cual puede recibir, guardar y enviar señales. Además, permite definir, configurar temporizadores variables y otros procesos [36]. A continuación, se presenta con los diferentes elementos que forman parte de un proceso.

Tabla 2.40 Elementos que forman parte de un proceso SDL [35] [36].

NOMBRE Y DESCRIPCIÓN	SIMBOLO
Cuadro de texto: Contiene la declaración de las estructuras, variables, temporizadores y señales. Para declarar variables se debe utilizar la palabra reservada DCL seguido del nombre (uno o varios) y el tipo de la variable que queremos declarar.	
Tarea: Usado para realizar tareas generales, como por ejemplo asignación de variables u operaciones sobre temporizadores. Puede contener una lista de asignaciones separadas por comas.	
Inicio: Indica el comienzo de un proceso. Debe existir una vez por cada proceso.	
Estado: Simboliza un estado del sistema.	
Entrada: Indica la espera de una señal de entrada por lo que se tratará de un símbolo bloqueante. Debe encontrarse inmediatamente después de un símbolo de estado. Si la señal es recibida se consume y se produce una transición de estado poniendo a disposición del proceso la información transportada por la señal.	
Salida: Envía una señal, usualmente se realizará al final de una transición.	
Decisión: Es usada para escoger entre dos rutas alternativas según el resultado de una condición.	
Temporizador: Los temporizadores son mensajes automáticos que se agregan a la cola de entrada, deben ser definidos, agregados y activados.	

2.7 REPRESENTACIÓN SDL DEL PROTOCOLO MQTT-SN PARA SU OPERACIÓN SOBRE EL NODO RCB256RFR2

Las máquinas de Mealy conseguidas en el literal 2.5 son de gran ayuda para el desarrollo de los diagramas SDL de los procedimientos del protocolo, debido a que los diagramas SDL manejan los mismos estados señales. Sin embargo, a pesar de que se utilicen los mismos estados y señales tanto en la representación SDL como en una máquina de Mealy, en esta última no se puede representar un cambio de estado que dependa de un contenido en algún campo de un mensaje recibido, tampoco es posible representar el momento en que se debe agregar un temporizador.

2.7.1 ENTIDAD PRINCIPAL

Para representar el funcionamiento del protocolo MQTT-SN para su operación sobre el nodo RCB256RFR2 mediante SDL, primero se debe especificar lo anterior a nivel de sistema. Como se muestra a continuación (Figura 2.46):

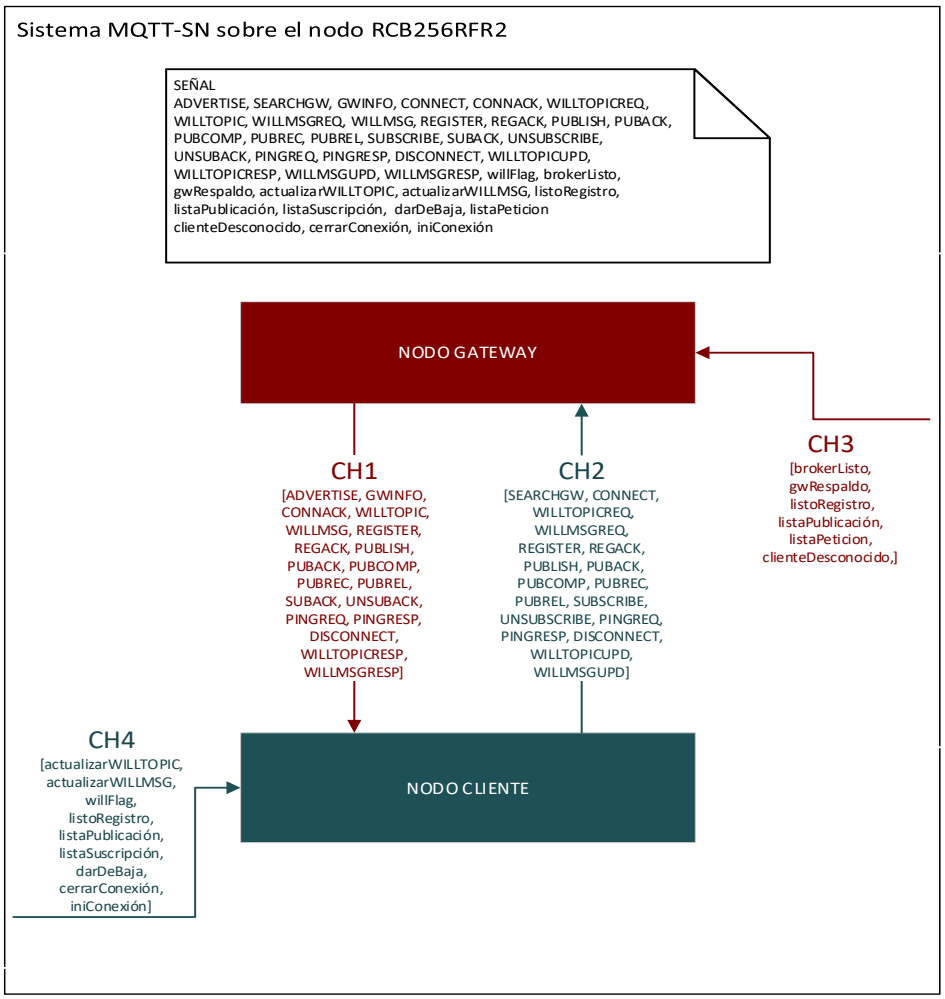


Figura 2.46 Entidad principal SDL: Sistema MQTT-SN sobre el nodo RCB256RFR2.

El sistema consta de cuatro canales y dos bloques (NODO CLIENTE y NODO GATEWAY) los cuales interactúan entre sí por medio de los canales uno y dos. Los canales restantes sirven para recibir señales que permitan iniciar el envío de mensajes requerido por los bloques. Además, existe un cuadro de texto que indica todas las señales a ser utilizadas por el sistema.

2.7.2 BLOQUE NODO GATEWAY

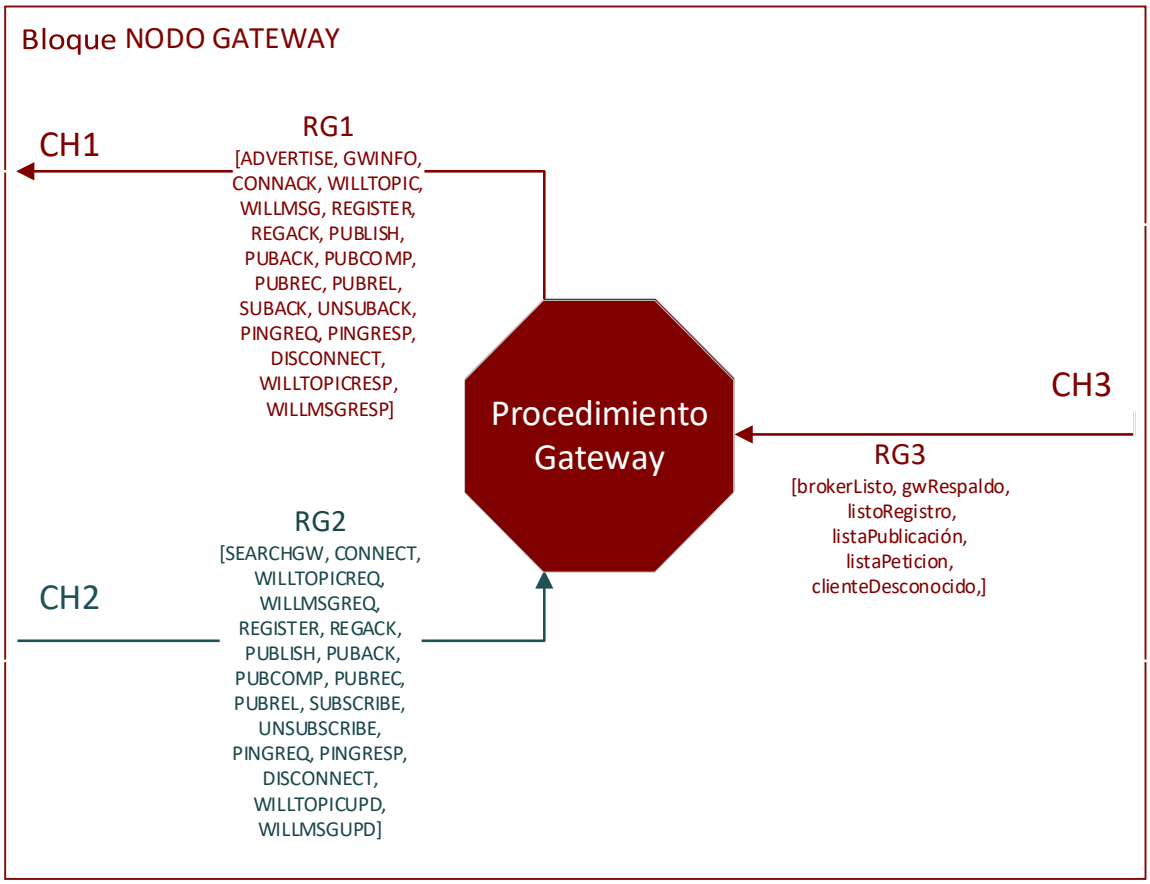


Figura 2.47 Bloque NODO GATEWAY.

El bloque NODO GATEWAY está compuesto por tres entradas (CH1, CH2, CH3) correspondientes a los canales de la entidad principal, y un proceso denominado procedimiento gateway. Las tres entradas están conectadas a los tres canales (RG1, RG2, RG3) que pertenecen a este bloque, los cuales están encargados de transportar las señales indicadas en el bloque sistema. No se requiere de un cuadro de texto ya que todas las señales ya han sido mencionadas.

2.7.3 BLOQUE NODO CLIENTE

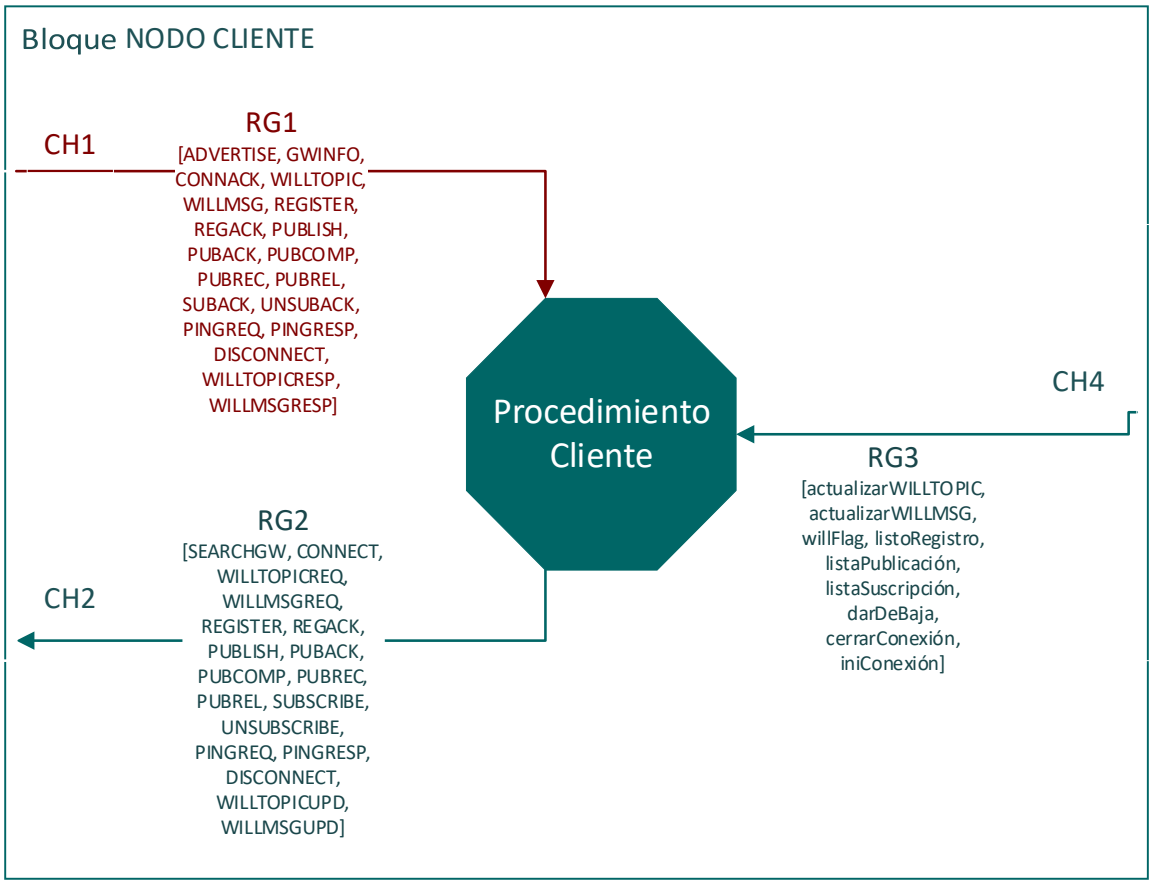


Figura 2.48 Bloque NODO CLIENTE.

Similar a lo anterior, el bloque NODO CLIENTE está compuesto de tres entradas (CH1, CH2, CH4) correspondientes a los canales de la entidad principal y un proceso, denominado procedimiento cliente. Las tres entradas están conectadas a los tres canales (RC1, RC2, RC3) que pertenecen a este bloque los cuales están encargados de transportar las señales indicadas en el bloque sistema.

2.7.4 PROCESOS

Estas entidades presentes en los bloques NODO CLIENTE y NODO CLIENTE, hacen referencia a cada uno de los procedimientos mencionados en los capítulos anteriores, contienen la máquina de estados perteneciente a los procedimientos que se deben ejecutar por parte del cliente y gateway. Contienen los mismos estados de las máquinas de Mealy y de manera similar, depende de cada procedimiento, los mensajes y señales que se reciben y transmiten. Además, se detalla de mejor manera el funcionamiento de los temporizadores y los cambios de estado debido al contenido de mensajes recibidos.

A continuación, se describe un proceso correspondiente al Procedimiento anuncio y descubrimiento de gateway.

2.7.4.1 Proceso Gateway

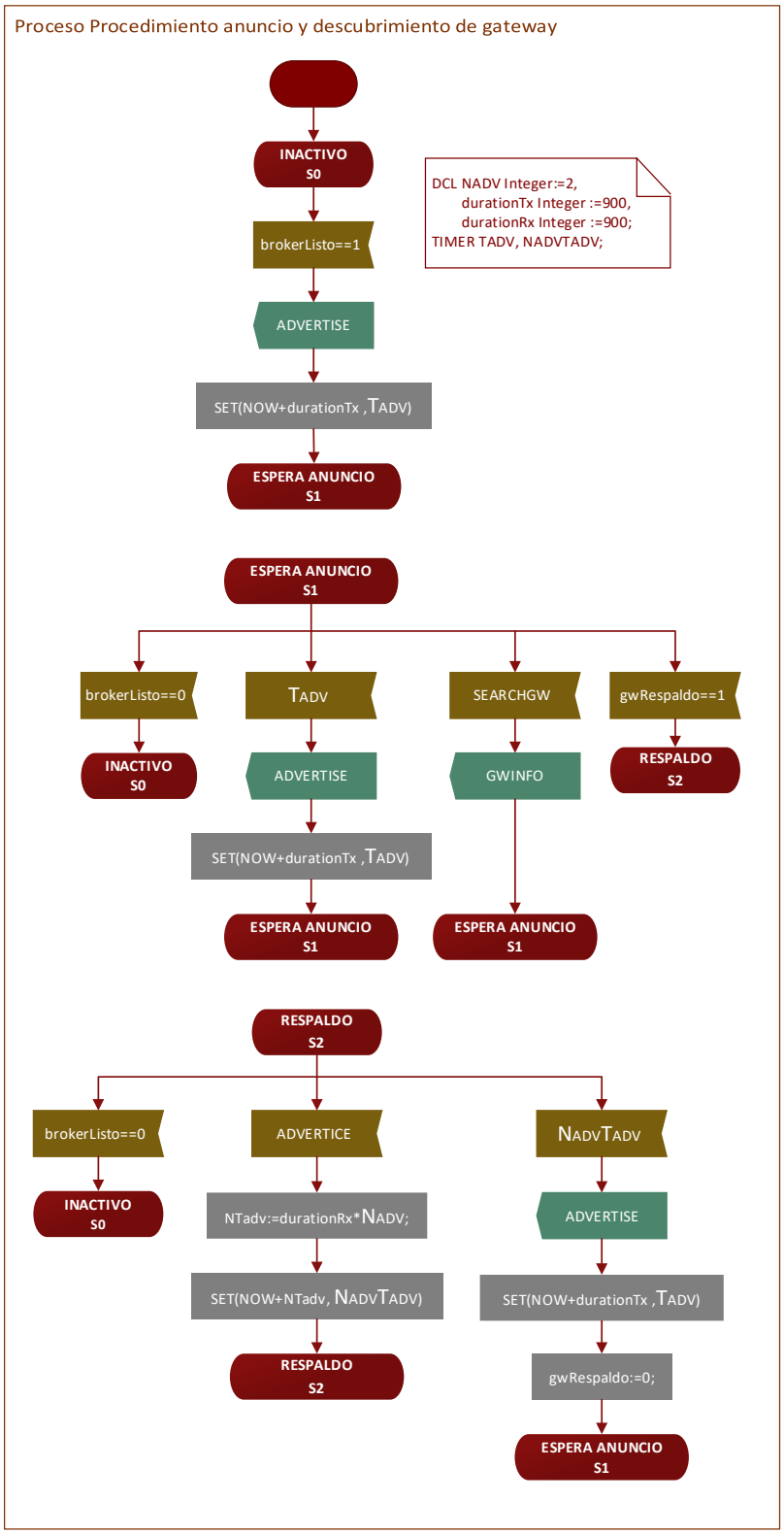


Figura 2.49 máquinas: Procedimiento anuncio y descubrimiento de gateway.

El proceso es similar a la máquina de Mealy del mismo procedimiento, con la diferencia de que aquí se agregan los temporizadores que generalmente inician al momento de enviar o recibir un mensaje *ADVERTISE*. Además, se agrega las variables durationRX, durationTX y NADV para controlar la duración de los temporizadores.

2.7.4.2 Proceso cliente

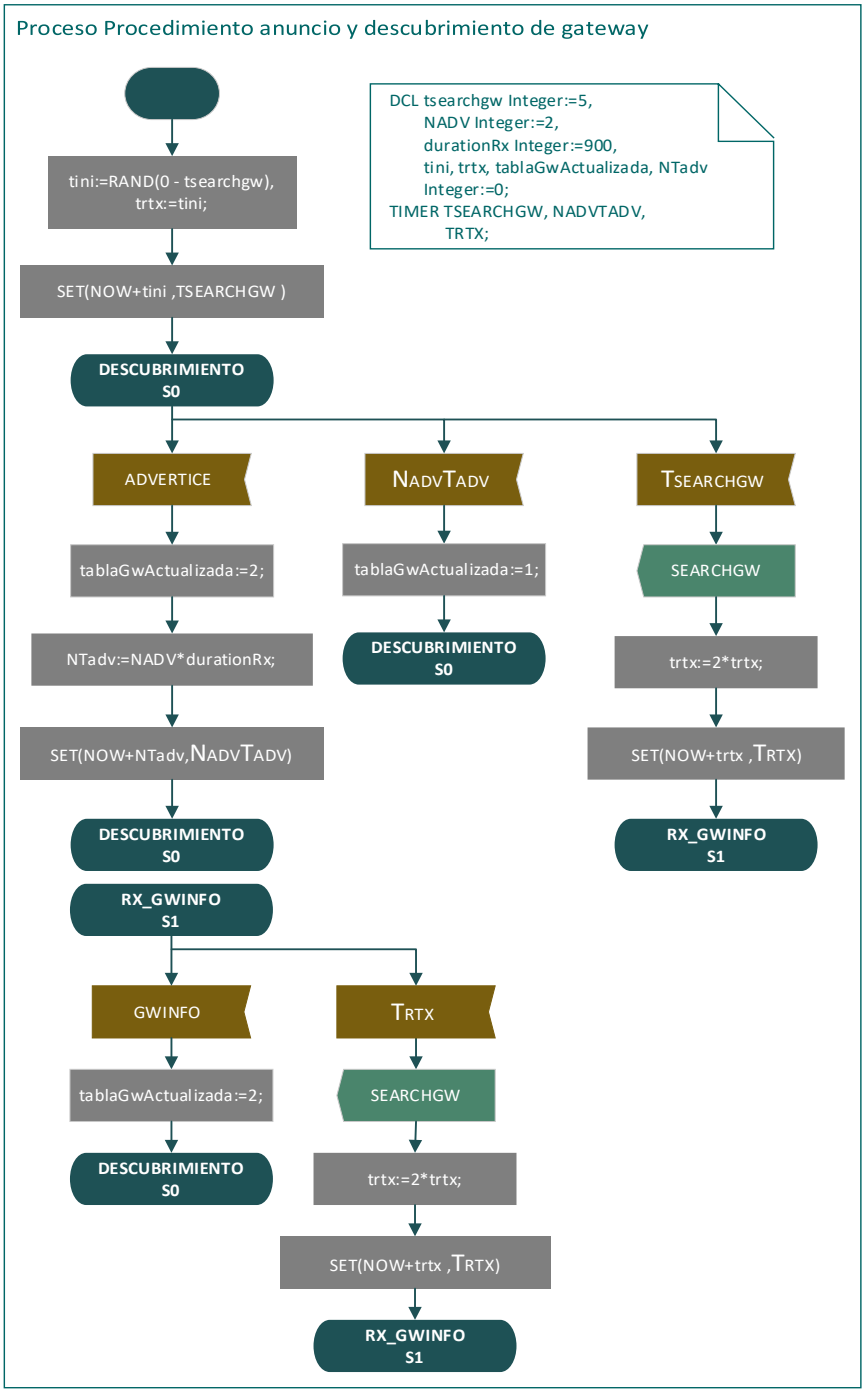


Figura 2.50 Proceso cliente: Procedimiento anuncio y descubrimiento de gateway.

En el proceso se muestra que los temporizadores son agregados e iniciados cuando se recibe *ADVERTISE* o se envía un *SERACHGW*. También se agrega las variables *tSEARCHGW, NADV, durationRx, tini, trtx, NTadv* para controlar la duración de los temporizadores. Por otra parte, la variable *tablaGwActualizada* es utilizada para indicar que cuando se ha intercambiado la secuencia correcta de mensajes, se debería actualizar la tabla de gateways de un cliente.

El resto de los diagramas se especifican en el ANEXO A. Los valores de temporizadores y contadores son asignados de acuerdo con la Tabla 2.41; sin embargo, para la codificación los valores pueden cambiar para poder realizar pruebas de funcionamiento.

Tabla 2.41 Valores sugeridos para temporizadores y contadores [28].

Temporizadores y contadores	Valores recomendados
TADV	Mayor a 15 min
NADV	2 -3
T*SEARCHGW*	5 seg
T*GWINFO*	5 seg
TWAIT	Mayor a 5 min
Tretry	10 - 15 seg
Nretry	3 - 5

2.8 IMPLEMENTACIÓN DE ESTADOS EN LOS NODOS RCB256RFR2

A continuación, se describe los principales archivos, librerías, funciones y variables utilizadas para codificar los estados definidos previamente. El código completo se encuentra en el ANEXO C.

2.8.1 FUNCIÓN PRINCIPAL (MAIN)

La función *main*, se encuentra en el archivo *Main.c* y permite ejecutar todas las funcionalidades programadas en el nodo. Dentro de la función existe un bucle *while* que se ejecuta infinitamente para que el nodo reciba y transmita mensajes de manera constante. Se crea automáticamente al crear un nuevo proyecto en el IDE de Atmel studio.

```
int main(void)
{
    /* Inicia todas las funcionalidades del Nodo.*/
    wireless_init();
    }
    modules_init();

    while (1)/*Este lazo permite ejecutar sus funciones de manera indefinida
    {
        WirelessTask();
    }
}
```

Código 2.1 Función Main.

2.8.2 ARCHIVO USR_WIRELESS.C

El archivo *usr_wireless.c* contiene las diferentes funciones que permiten la recepción y transmisión de datos. Debido a esto el archivo también contiene las funciones que controlan los diferentes estados en los que el nodo debe permanecer.

La función *main,* mediante la función *WirelessTask ()*, llama al archivo *usr_wireless.c* y procede a ejecutar las funciones *usr_wireless_app_task* y *usr_frame_received_cb* las cuales permanecen en funcionamiento constante debido al lazo de la función principal.

```
 void usr_wireless_app_task(void)
{
       if (indiceEstado==0)     // INACTIVO
       {
              estadoInicial();
       }
}

void usr_frame_received_cb(frame_info_t *frame)
{

}
```

Código 2.2 Función usr_wireless_app_task y usr_frame_received_cb.

La codificación de cada procedimiento MQTT-SN estará presente en este archivo, sin embargo, se debe ejecutar un procedimiento a la vez.

2.8.3 LIBRERÍAS Y VARIABLES REQUERIDAS PARA IMPLEMENTACIÓN DEL CÓDIGO

Las librerías utilizadas para el desarrollo del proyecto son librerías predefinidas que se insertan en el código al momento de crear un proyecto nuevo, excepto la librería Mensajes_ MQTT_SN .h la cual es creada específicamente para la codificación.

Por otra parte, se utilizan distintas variables de tipo *int*, para controlar las funciones que manejan los estados del nodo, la función de recepción y transmisión del nodo, los temporizadores, entre otros.

```
#include "usr_wireless.h"//Inicializa el funcionamiento del nodo
#include "wireless_config.h"// Permite el acceso a los archivos que contienen las
                            // características de transmisión del nodo
#include "periodic_timer.h"// Permite utilizar temporizadores
#include "Mensajes_MQTT_SN.h"// Permite el acceso a los mensajes MQTT-SN
                             //almacenados en el nodo
////////////////////////////////////////////////////////////////////////////////
int indiceEstado=0;         //Variables para controlar los estados
int indiceEstadoTemporal=0; //que puede manejar un nodo

int controldeEstado=1; //Variable para controlar las funciones que representan los
                       //estados del nodo
int habilitarRecepcion=0;   //Variables que habilitan la transmisión y recepción del
int habilitarTransmision=-1;//nodo cuando se encuentra en un determinado estado

int longitudMSG=0;// Almacena el tamaño del Mensaje MQTT-SN a ser enviado
int controldeEstadoRecepcionExitosa=-1;//Variable utilizada para indicar la
                                       //recepción de un mensaje MQTT-SN en específico.
tiempoDuracionTimer=0;//Variable para asignar la duración de los temporizadores.
int agregarTimer=0;//Variable para activar el temporizador cuando sea necesario
TimerFinaliza=0;// Variable para indicar que la curación de temporizador ha finalizo

int salto=0;// Se utiliza para asignar un retardo después de utilizar la función
            // Transmit_sample_frame()
```

Código 2.3 Variables a utilizar dentro del proyecto.

Estas variables son comunes para todos los procedimientos del protocolo, sin embargo, existen más variables que dependen del procedimiento que se ejecuta dentro del nodo cliente o gateway de acuerdo a los diagramas SDL de secciones anteriores.

2.8.4 CODIFICACIÓN DE PROCEDIMIENTO MQTT-SN

Los procedimientos MQTT-SN son codificados según los indicado en los diagramas SDL en el archivo *usr_wireless.c.* De acuerdo al procedimiento, se declaran las variables que se utilizarán para asignar la duración de los temporizadores, indicar el máximo número de retransmisiones de un mensaje, indicar como termina un intercambio de mensajes entre nodos cliente y gateway, e incluso se puede representar señales que permiten realizar cambios de estados.

```
//********** PROCEDIMIENTO ANUNCIO Y DESCUBRIMIENTO DE GATEWAY**************//
//*
int durationTX=25;//int durationTX=900;
int durationRX=25;//int durationTX=900;
int listoBroker=1; //Señal que indica el estado del broker
int gwRespaldo=0;//Señal para pasar a estado respaldo
int NADV=2; //Número máximo de veces que se puede dejar pasar
            //la recepción de un mensaje de advertencia.

void estadoInicial(void);// INACTIVO S0
```

```
void estado_1(void);     // ESPERA_ANUNCIO S1

void estado_2(void);     // RESPALDO S2


void usr_wireless_app_task(void)
{
       if (indiceEstado==0)     // INACTIVO
       {
              estadoInicial();
       }
       else if (indiceEstado==1)// ESPERA_ANUNCIO S1
       {
              estado_1();
       }
       else if (indiceEstado==2)// RESPALDO S2
       {
              estado_2();
       }
}*/
```

Código 2.4 Función usr_wireless_app_task(void).

Después de declarar las variables se continua con las funciones que permiten que el nodo se mantenga en un estado o pase a otro estado. El número de funciones depende del número de estados del procedimiento de acuerdo a los diagramas de los literales 2.5 y 2.6. Las funciones son ejecutadas una a la vez dentro de la función *usr_wireless_app_task,* y debido a que esta última se ejecuta permanentemente el nodo permanece en el estado que la función indique.

Para la codificación de los procedimientos se consideró que cada uno de ellos opera de forma independiente, su código completo se encuentra en el ANEXO C. A continuación, se explica las partes más importantes del código desarrollado.

2.8.5 CODIFICACIÓN DE LA FUNCIÓN ESTADO

```
void estadoInicial(void){ // ACTIVO S0
       // Denominación de estado
       uint8_t Estado[48]="|========ACTIVO========||       CL S-00        |";
       if (controldeEstado==1)
       {
              controldeEstado=2;
              delay_ms(1500);// Tiempo de espera para apreciar el inicio de estado
       }
       else if (controldeEstado==2)
       {
              controldeEstado=3;
              transmit_sample_frame(Estado,48,1);// Indica el estado del Nodo
       }
       else if (controldeEstado==3)
       {
              // Recepción de señales que propician un cambio de estado
              // (mensajes, temporizadores, señales internas, etc.)
              ////////////////////////////////////////////////////////////////////
              // RX de mensaje ADVERTISE
```

```
            if (controldeEstadoRecepcionExitosa==0)
            {
                  controldeEstadoRecepcionExitosa=-1;
                  stop_timer1();
                  tablaGwActualizada=2;
                  controLED(tablaGwActualizada);
                  agregarTimer=1;// Pasa a agregar Timer
            }
            else if (agregarTimer==1)// Agregar Timer NADVTADV
            {
                  agregarTimer=0;
                  NTadv=NADV*durationRx;
                  tiempoDuracionTimer=NTadv
                  start_timer1();//
                  // CAMBIO DE ESTADO
                  auxTimer=1;
                  indiceEstadoTemporal=0;
                  controldeEstado=4; // Salida del estado
            }

      ////////////////////////////////////////////////////////////////////////
      }
      else if (controldeEstado==4)
      {
            controldeEstado=5;
            delay_ms(1500);//tiempo de espera para apreciar el cambio de estado
      }
      else if (controldeEstado==5)
      {
            controldeEstado=1; //Se iguala a 1 para reutilizar la variable
            //Indica que el estado actual ha terminado
            transmit_sample_frame((uint8_t*)"|  Termina  ||---CL S-00 ---|",48,1);
            indiceEstado=indiceEstadoTemporal; //Cambio de estado
            indiceEstadoTemporal=-1;//Se iguala a -1 para reutilizar la variable
      }
}
```

Código 2.5 Función para controlar el estado del nodo.

Un estado este compuesto de cinco condicionales, los cuales son utilizados para indicar el estado en el que se encuentra el nodo. Los dos primeros condicionales indican que se ha llegado a un nuevo estado, mientras que los dos últimos indican que se produce un cambio de estado. El tercer condicional es utilizado para recibir los diferentes eventos que provocan un cambio de estado (recepción de mensajes, temporizadores o señales propias de cada procedimiento), también el nodo responde con algún mensaje de ser necesario.

2.8.6 ENTRADAS

A continuación, se describe el código necesario que permite al nodo recibir las diferentes entradas para que este pueda ejecutar un cambio de estado.

2.8.6.1 Estructura para recibir una trama IEEE 802.15.4

```
#define max_dato 101
typedef struct
{
```

```
        uint8_t longitud;                       // Longitud de Trama
        uint16_t fcf;                           // fcf
        uint8_t num_sec;                        // Numero de secuencia
        uint16_t d_PAN;                         // Direccion PAN
        uint16_t d_dstn;                        // Direccion ORIGEN
        uint16_t d_orgn;                        // Direccion DESTINO
        uint8_t mensaje_MQTT_SN[max_dato];      // Payload
        uint16_t fcs;                           // frame control secquence
}trama_ieee_802_15_42;

trama_ieee_802_15_42 trama_recibida;
```

Código 2.6 Estructura para recibir un mensaje 802.15.4.

Se define una estructura que almacena una trama de datos IEEE 802.15.4 recibida por el nodo. Además, el campo que contiene la carga útil es el encargado de encapsular cualquier tipo de mensaje MQTT-SN.

2.8.6.2 Función usr_frame_received_cb

```
void usr_frame_received_cb(frame_info_t *frame)
{
        if (habilitarRecepcion==1)
        {     // Reserva el espacio de memoria
              memset(&trama_recibida,0,sizeof(trama_recibida));
              // Copia la información de la memoria fuente(buffer) al destino
              memcpy(&trama_recibida,frame->mpdu,sizeof(trama_recibida));
              //Elimina los datos del buffer, evita superposición.
              bmm_buffer_free(frame->buffer_header);
              // Se extrae el campo tipo del mensaje MQTT-SN de la estructura que
              // contiene la trama 802.15.4
              uint8_t msgTypeMQTT_SN;
              msgTypeMQTT_SN=trama_recibida.mensaje_MQTT_SN[1];

              if (msgTypeMQTT_SN==0x00) // RX: Mensaje MQTT-SN
              {       // Deshabilita la recepción para evitar interferencias
                      habilitarRecepcion=0;
                      // Esta variable indica a la función estado la recepción del
                      // mensaje esperado
                      controldeEstadoRecepcionExitosa=1;
              }

              else if (msgTypeMQTT_SN==0x01) // RX: Mensaje MQTT-SN
              {       // Deshabilita la recepción para evitar interferencias
                      habilitarRecepcion=0;
                      // Esta variable indica a la función estado la recepción del
                      // mensaje esperado
                      controldeEstadoRecepcionExitosa=1;
              }
        }
```

```
}
```

Código 2.7 Función para recepción de trama 802.15.4.

Esta función permite al nodo recibir una trama 802.15.4, la cual es almacenada en una estructura del mismo nombre. Después de recibir la trama, se analiza la carga útil para conocer la recepción de un mensaje MQTT-SN esperado por el estado del nodo.

Para conocer el tipo de mensaje recibido se analiza directamente la posición dos del mismo. La posición mencionada contiene el campo tipo de mensaje, mediante este campo y la variable *controldeEstadoRecepcionExitosa* permiten que el nodo cambie de estado.

2.8.6.3 Temporizadores

Los temporizadores deben ser configurados en los archivos *periodic_timer.h* y *usr_periodic_timer.c,* el primero contiene la duración del temporizador y el segundo contiene la función a ejecutarse cuando la duración del temporizador finalice. La función asigna el valor de uno a la variable *TimerFinaliza* cuando tiempo asignado al temporizador termina. Cuando una función estado recibe esta variable, considera que el temporizador ha finalizado. A continuación, se ejecuta, de acuerdo al tipo de procedimiento MQTT-SN, un envío de menaje, una agregación de un nuevo temporizador, o un cambio de estado.

```
// Variable para indicar que el temporizador ha finalizado
int TimerFinaliza;
// Asignación del tiempo de duración del temporizador
int tiempoDuracionTimer;
#define TIMER_DURATION 1000000*tiempoDuracionTimer

#endif
```

Código 2.8 Variables y constantes dentro del archivo periodic_timer.h

```
void usr_app_timer_cb(void *parameter)
{
// Variable utilizada para indicar la finalización del temporizador el código
// principal.
       TimerFinaliza=1;
}
```

Código 2.9 Función del temporizador dentro del archivo usr_periodic_timer.c

2.8.6.4 Pulsador

El nodo RCB256RFR2 posee un pulsador al cual se tiene acceso por medio de la función *ioport_get_pin_level(GPIO_PUSH_BUTTON_0*).

Los cambios de estado de un nodo, no solo se dan a través de la recepción de mensajes o a los temporizadores por lo cual se utiliza el pulsador del nodo para recrear la recepción de las señales requeridas por el nodo, para que este pueda cambiar de estado.

```
if (!ioport_get_pin_level(GPIO_PUSH_BUTTON_0))
{
	delay_ms(200); //Retardo para que funcione el pulsador
	anuncioBroker=1;// Variable para ejecutar una acción requerida por la función
			// de estado después de recibir una señal del pulsador.
}
```

Código 2.10 Verificación del pulso del nodo.

2.8.6.5 Mensajes MQTT-SN

Para crear los diferentes mensajes MQTT-SN primero se declaran un conjunto de vectores, utilizados para almacenar los diferentes mensajes tipos de MQTT-SN. Si el nodo requiere enviar mensajes MQTT-SN utiliza estos vectores.

```
uint8_t ADVERTICE[5];
uint8_t SEARCHGW[3];
uint8_t GWINFO[6];
uint8_t CONNECT[29];
uint8_t CONNACK[3];
uint8_t WILLTOPICREQ[2];
uint8_t WILLTOPIC[29];
uint8_t WILLMSGREQ[2];
uint8_t WILLMSG[29];
uint8_t REGISTER[29];
uint8_t REGACK[7];
uint8_t PUBLISH[60];
uint8_t PUBACK[7];
uint8_t PUBCOMP[4];
uint8_t PUBREC[4];
uint8_t PUBREL[4];
uint8_t SUBSCRIBE[29];
uint8_t SUBACK[8];
uint8_t UNSUBSCRIBE[29];
uint8_t UNSUBACK[4];
uint8_t PINGREQ[4];
uint8_t PINGRESP[2];
uint8_t DISCONNECT[4];
uint8_t WILLTOPICUPD[29];
uint8_t WILLMSGUPD[29];.
uint8_t WILLTOPICRESP[3];
uint8_t WILLMSGRESP[3];
```

Código 2.11 Vectores para almacenar Mensajes MQTT-SN.

Para llenar a los vectores antes mencionados, con los diferentes campos MQTT-SN se crean varias funciones que reciben al vector y otros datos adicionales. La función retorna un entero que hace referencia a la longitud del mensaje MQTT-SN a ser enviado.

```
int funcionADVERTICE(uint8_t *msg);
int funcionSEARCHGW(uint8_t *msg);
int funcionGWINFO(uint8_t *msg, char tipoNodo);
int funcionCONNECT(uint8_t *msg, uint8_t flags, uint8_t *clientId, int clidLongitud);
int funcionCONNACK(uint8_t *msg,uint8_t returnCode);
int funcionWILLTOPICREQ(uint8_t *msg);
int funcionWILLTOPIC(uint8_t *msg, uint8_t flags, uint8_t *willTopic, int willTopicLongitud);
int funcionWILLMSGREQ(uint8_t *msg);
int funcionWILLMSG(uint8_t *msg, uint8_t *willMsg, int willMsgLongitud);
int funcionREGISTER(uint8_t *msg, uint8_t *topicName, int topicNameLongitud);
int funcionREGACK(uint8_t *msg, uint8_t returnCode);
int funcionPUBLISH(uint8_t *msg, uint8_t flags, uint8_t *data, int dataLongitud);
int funcionPUBACK(uint8_t *msg, uint8_t returnCode);
int funcionPUBCOMP(uint8_t *msg);
int funcionPUBREC(uint8_t *msg);
int funcionPUBREL(uint8_t *msg);
int funcionSUBSCRIBE(uint8_t *msg, uint8_t flags, uint8_t *topicNameoId, int topicNameoIdLongitud);
int funcionSUBACK(uint8_t *msg, uint8_t flags, uint8_t returnCode);
int funcionUNSUBSCRIBE(uint8_t *msg, uint8_t flags, uint8_t *topicNameoId, int topicNameoIdLongitud);
int funcionUNSUBACK(uint8_t *msg);
int funcionPINGREQ(uint8_t *msg);
int funcionPINGRESP(uint8_t *msg);
int funcionDISCONNECT(uint8 t *msg, char tipoDISC);
```

Código 2.12 Funciones para asignar campos de mensajes MQTT-SN.

Para que las funciones mencionadas rellenen un vector con los diferentes campos MQTT-SN, lo que hacen es declarar variables uint8_t que representan los diferentes campos de un mensaje MQTT-SN. A continuación, las variables son agregadas a su respectivo vector para después retornar la longitud del mismo. Después de llamar a alguna de estas funciones su respectivo vector ya se puede utilizar como un mensaje MQTT-SN. También, hay que mencionar que las funciones también pueden recibir cadenas de caracteres, y otros campos que no estén predefinidos.

```
int funcionSEARCHGW(uint8_t *msg){
	//Campos del mensaje MQTT-SN
	uint8_t Length=0x03;
	uint8_t MsgType=0x01;
	uint8_t Radius=0x00;
	// Asignación de campos al vector que representa un mensaje MQTT-SN
	msg[0]=Length;
	msg[1]=MsgType;
	msg[2]=Radius;
	// Retorno de longitud del mensaje
	return sizeof(SEARCHGW);
}

int funcionPUBLISH(uint8_t *msg, uint8_t flags, uint8_t *data, int dataLongitud){
	//Campos del mensaje MQTT-SN
	uint8_t Length=0x07+dataLongitud;
	uint8_t MsgType=0x0C;
	uint8_t Flags=flags;
	uint8_t topicId1=0x00;
	uint8_t topicId2=0x01;
	uint8_t msgId1=0x00;
	uint8_t msgId2=0x01;
	// Asignación de campos al vector que representa un mensaje MQTT-SN
```

```
        msg[0]=Length;
        msg[1]=MsgType;
        msg[2]=Flags;
        msg[3]=topicId1;
        msg[4]=topicId2;
        msg[5]=msgId1;
        msg[6]=msgId2;
        int j=7;
        // Asignación de campos que requiere cadenas de caracteres

        for (int i=0;i<dataLongitud;i++)
        {
                msg[j]=data[i];
                j++;
        }
        return Length;
}
```

Código 2.13 Funciones para construir mensaje MQTT-SN.

El Resto de funciones utilizadas para generar todos los mensajes MQTT-SN se encuentran en el ANEXO C. Los mensajes MQTT se verán se la siguiente manera.

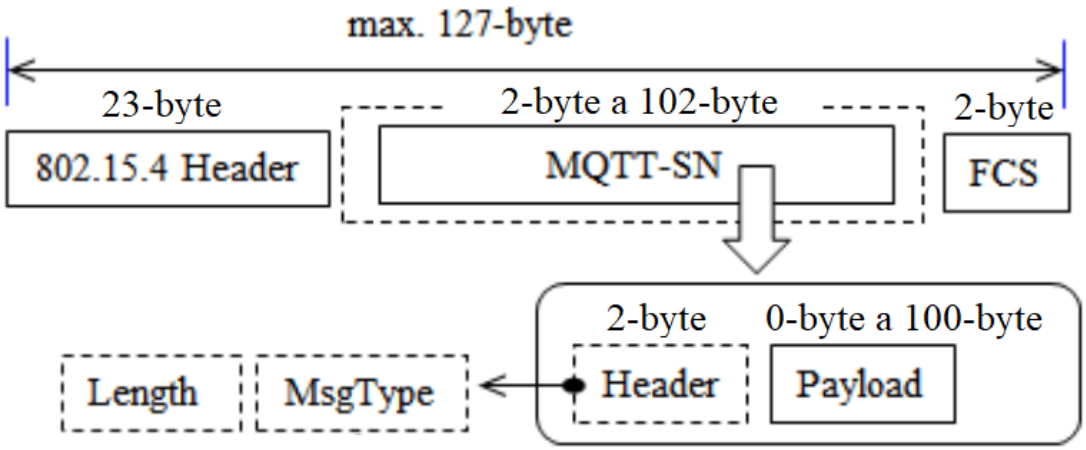


Figura 2.51 Mensaje MQTT-SN sobre IEEE 802.15.4.

2.8.7 SALIDAS

Si el nodo se encuentra en algún estado y requiere enviar algún mensaje MQTT-SN, la función estado que este ejecutando el nodo debe llamar a la función *transmit_sample_frame().* La función envía el mensaje MQTT-SN dentro de la carga útil de una trama IEEE802.15.4. Previo al envío se debe llenar el vector que contiene el mensaje, que se requiere enviar, mediante su función correspondiente la cual retorna el tamaño del menaje.

```
longitudMSG=funcionADVERTISE(ADVERTISE);// Invocación de la función que crea el
mensaje a ser enviado, retorna la longitud del mensaje
transmit_sample_frame(ADVERTISE,longitudMSG,0);//Envío del mensaje MQTT-SN
longitudMSG=0;// La longitud regresa la longitud del mensaje 0 para evitar la
```

```
        //superposición
```

Código 2.14 Envío de mensajes MQTT-SN.

La función *transmit_sample_frame()* requiere del vector que contiene el mensaje MQTT-SN y longitud la cual esta almacenada en la variable *longitudMSG*. La variable que almacena la longitud debe igualarse a cero despues del envío del mensaje ya que debe ser reutilizada para obtener la longitud de otros mensajes.

2.8.8 CÓDIGO PARA NODOS INTERMEDIOS

Para que los nodos intermedios retransmitan los mensajes generados por el nodo cliente y nodo gateway, se debe configurar la función usr_*frame_received_cb()*. La función es la encarga de analizar el tipo de mensaje MQTT-SN recibido y su dirección de origen para posteriormente enviarlo al siguiente nodo, hasta que llegue a su destino.

```
void usr_frame_received_cb(frame_info_t *frame)
{
       memset(&trama_recibida,0,sizeof(trama_recibida));// recerba el espacio de
                                                           memoria
       // copia la informacion de la memoria fuente(buffer) al detino
       memcpy(&trama_recibida,frame->mpdu,sizeof(trama_recibida));
       //Elimina los datos del buffer, evita superpocicion.
       bmm_buffer_free(frame->buffer_header);
       uint16_t dirOrigen;//Direccion del mensaje Recibido
       uint8_t msgTypeMQTT_SN;//Tipo de mensaje MQTT-SN
       int longitudMQTT_SN;//longitud de mensaje MQTT-SN

       dirOrigen=trama_recibida.d_orgn;
       longitudMQTT_SN= sizeof(trama_recibida.mensaje_MQTT_SN);
       msgTypeMQTT_SN=trama_recibida.mensaje_MQTT_SN[1];

       if (dirOrigen==0x001)//recibe la direccion origen del nodo 0x001
       {// Mediante la variable msgTypeMQTT_SN se procede a comprobar el tipo de
           mensaje recido.
              if (msgTypeMQTT_SN==0x00)//tipo de mensaje MQTT-SN recibido
              {
       //envió de mensaje MQTT-SN
       transmit_sample_frame(trama_recibida.mensaje_MQTT_SN,longitudMQTT_SN,3);
              }
              else if (msgTypeMQTT_SN==0x01)//mensaje MQTT-SN recibido
              {
       //envió de mensaje MQTT-SN
       transmit_sample_frame(trama_recibida.mensaje_MQTT_SN,longitudMQTT_SN,3);
              }
              else if (msgTypeMQTT_SN==0x02)//mensaje MQTT-SN recibido
              {
       //envió de mensaje MQTT-SN
       transmit_sample_frame(trama_recibida.mensaje_MQTT_SN,longitudMQTT_SN,3);
              }
```

Código 2.15 Función *usr_frame_received_cb()* para nodos intermedios.

2.9 SIMULACIONES DE LAS MÁQUINAS DE ESTADOS

El simulador de máquinas de estados JFLAP permite simular diferentes tipos de máquinas de estados, entre ellas la máquina de Mealy, de esta forma se puede recrear los diferentes cambios de estados del nodo (establecidos en los literales 2.5). El simulador requiere de una cadena de entrada que representa las diferentes secuencias de mensajes MQTT-SN con las que el nodo debe trabajar. En la siguiente figura se muestra el entorno brindado por JFLAP, donde se puede crear y simular una máquina de estados.

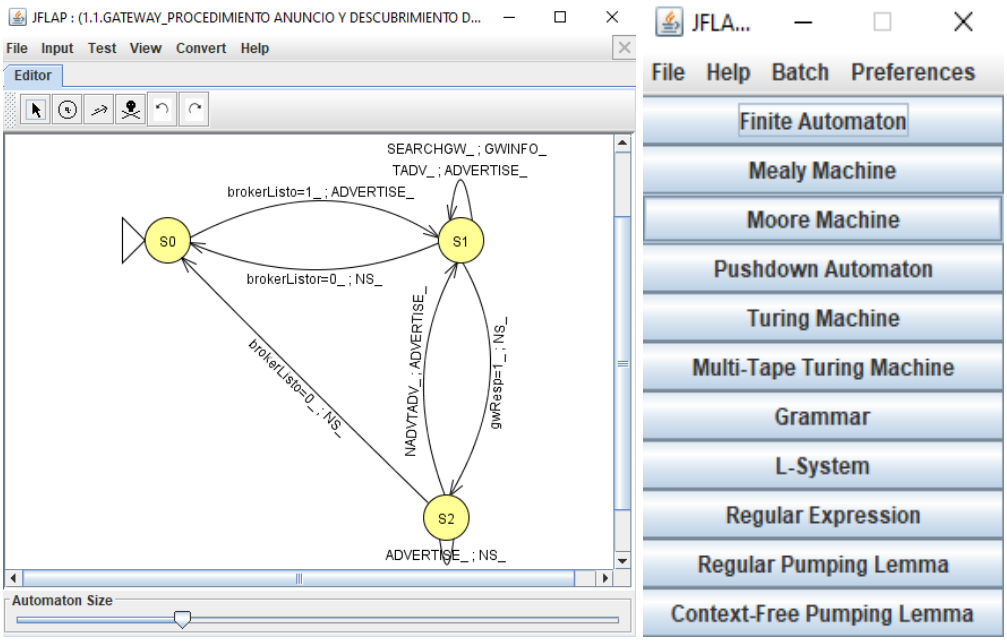


Figura 2.52 Simulador de máquina de estados JFLAP.

Para iniciar una simulación se requiere de una secuencia de entrada, la cual puede ser ingresada de forma manual o a través de archivo de texto previamente creados. Ingresada la secuencia de simulación el programa permite apreciar las transiciones de una máquina de Mealy paso a paso mediante la barra de control ubicada en la parte inferior de la pantalla de simulación. Además, junto a la barra de control se muestra la secuencia de entrada, la secuencia de salida y estado actual; también el estado actual en que se encuentra la MEF se muestra en el diagrama de la simulación, todo lo anterior se muestra en la Figura 2.53.

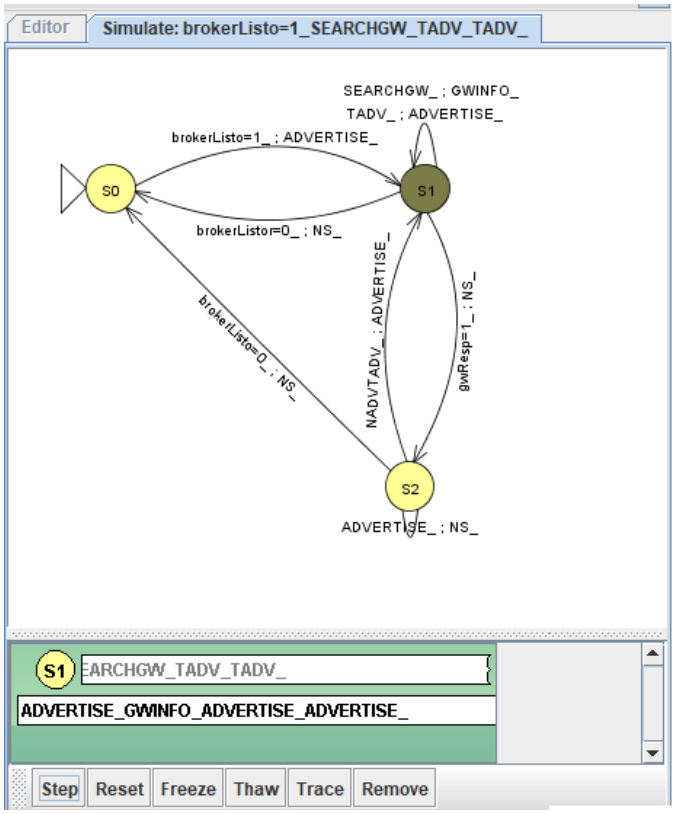


Figura 2.53 Ejemplo de simulación de máquina de Mealy.

3 RESULTADOS Y DISCUSIÓN

Para verificar que los nodos son capaces de responder a los estados planteados, se realiza pruebas de funcionamiento de los procedimientos previamente codificados en los nodos. Los mensajes enviados por los nodos son capturados por la herramienta SmartRF Packet Sniffer y se realiza una comparación con su respectiva simulación.

3.1 ESCENARIO DE PRUEBAS

Es necesario establecer una red de pruebas, con topología lineal, para comprobar los diferentes cambios de estado realizados por el nodo cliente y el nodo gateway. La red consta de cuatro nodos RCB256RFR2, los nodos de borde contienen el código que permite representar su respectiva máquina de estados. Los dos nodos internos son los encargados de transportar cada uno de los mensajes MQTT-SN generados por los nodos de borde (cliente y gateway). Se podría agregar más nodos de tal forma que se aprecie de mejor manera la topología lineal, sin embargo, con dos nodos intermedios es suficiente para realizar las pruebas de funcionamiento.

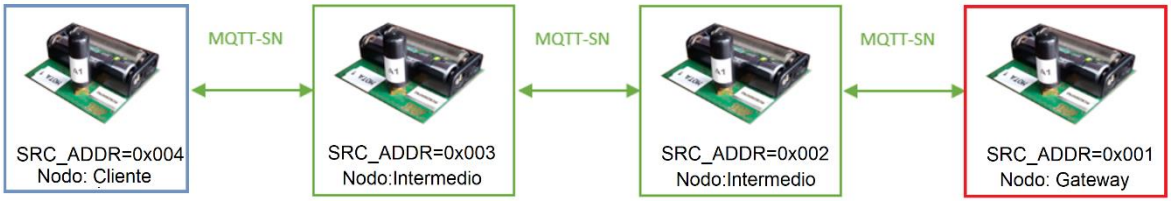


Figura 3.1 Red de pruebas.

Cada uno de los nodos contiene una dirección única con la cual los nodos se comunican entre sí. El nodo cliente (SRC_ADDR=0x004) establece comunicación con el nodo intermedio más cercano (SRC_ADDR=0x003), el nodo gateway (SRC_ADDR=0x002) también debe establecer comunicación con el nodo intermedio más cercano (SRC_ADDR=0x002) y los nodos intermedios se comunican entre si, de esta forma los mensajes MQTT-SN generados por los nodos de borde son transportados por los nodos intermedios.

3.2 CONFIGURACIÓN DE LA HERRAMIENTA DE MONITOREO

Para verificar el estado en que se encuentra un nodo, se hace uso de la herramienta de monitoreo SmartRF Packet Sniffer. Mediante la herramienta se puede capturar y apreciar los campos de las tramas 802.15.4 mediante una interfaz gráfica. Las tramas contienen los mensajes MQTT-SN generados y enviados por los nodos. También, se pude observar el campo payload de la trama en formato de texto o en formato hexadecimal.

Debido a que se insertó mensajes que indican el estado en que se encuentra el nodo, solo se puede apreciar estos mensajes cuando al campo payload se lo aprecia en formato de texto, mientras que los mensajes MQTT-SN solo se los puede apreciar cuando el campo está en formato hexadecimal. A continuación, se explica cómo apreciar la duración de un estado y los mensajes que los nodos reciben y transmiten.

3.2.1 REPRESENTACIÓN DE UN ESTADO

Para conocer el estado en el que se encuentra un nodo se envía un mensaje el cual indica el estado del mismo, el nodo permanece en este estado hasta que recibe una trama IEEE 802.15.4 la cual contiene un mensaje MQTT-SN. Cuando el nodo recibe el mensaje, dependiendo del caso se puede transmitir otro mensaje de respuesta, para a continuación realizar una transición de estado. Previo a realizar la transición de estado el nodo envía otro mensaje el cual indica que el estado anterior ha terminado. Los mensajes que indican el inicio y el fin de un estado se muestran en la Figura 3.2.

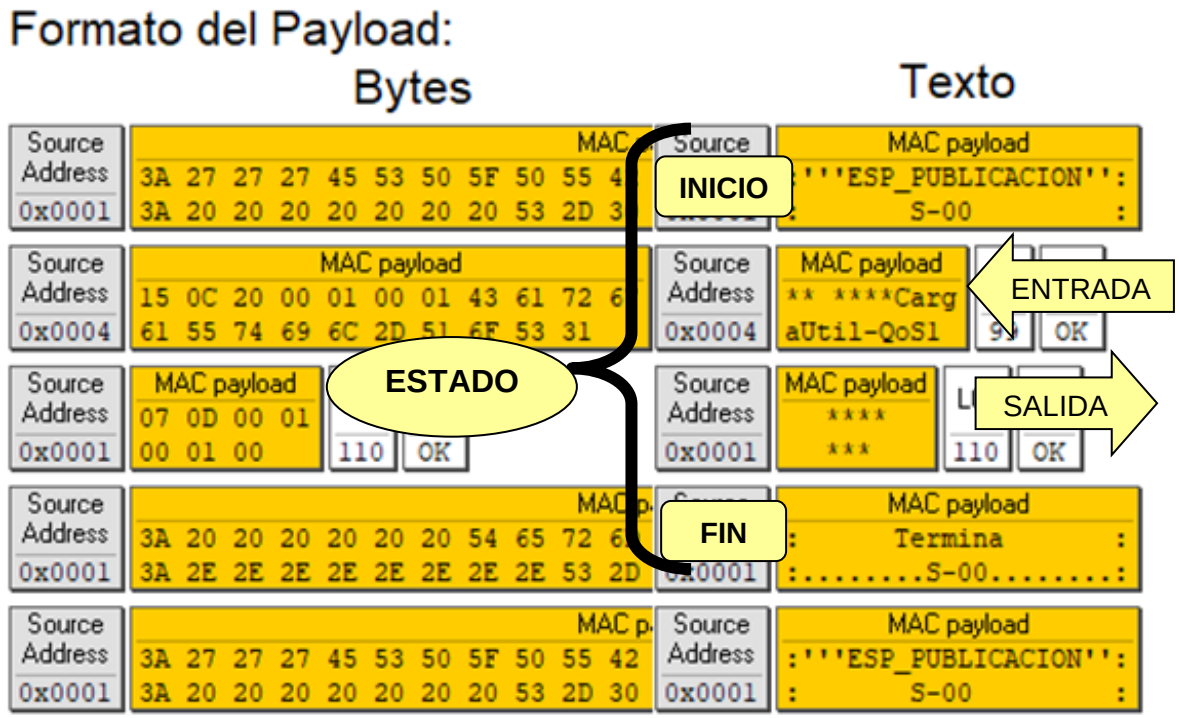


Figura 3.2 Apreciación del inicio y fin de un estado en la interfaz del Sniffer.

3.2.2 MENSAJES MQTT-SN

Los mensajes MQTT-SN al ser utilizados como entradas y salidas de una máquina de estados, deben estar ubicados entre los mensajes que indican el inicio y final de un estado. Los mensajes solo pueden ser apreciados si el formato de visualización del payload de la trama IEEE 802.15.4 se encuentra en formato hexadecimal.

Debido a que el Sniffer no puede clasificar los campos de los mensajes MQTT-SN es necesario, revisar manualmente los campos que permitan verificar que mensaje ingresa y sale del nodo. Para verificar el tipo de mensaje solo basta con revisar los dos primeros bytes del campo payload de la trama que contiene el mensaje MQTT-SN. El primer byte indica la longitud del mensaje y el segundo indica el tipo de mensaje, los dos bytes forman parte del encabezado fijo, mientras que el resto de bits representan la parte variable del mensaje MQTT-SN. La Figura 3.3 muestra cómo identificar los diferentes campos de los mensajes MQTT-SN.

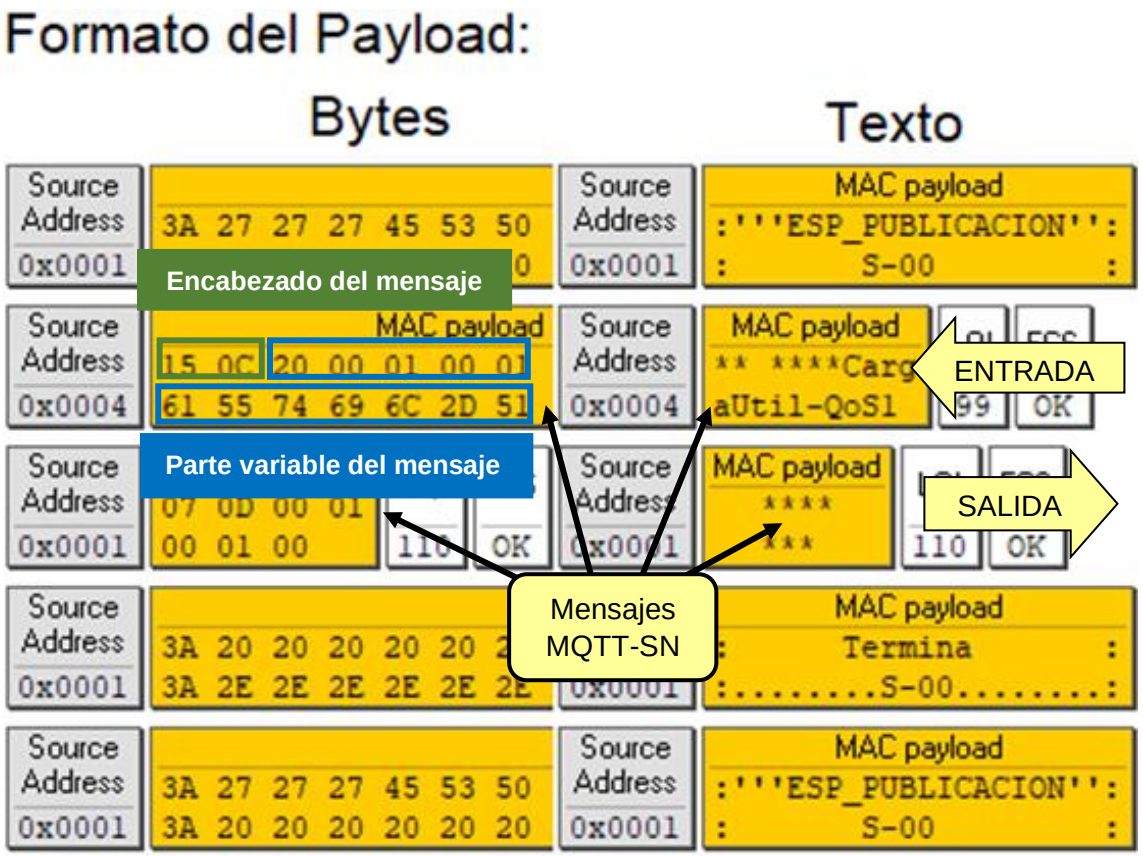


Figura 3.3 Apreciación del mensaje MQTT-SN en la interfaz del Sniffer.

3.3 PRUEBAS REALIZADAS

El nodo cliente y el nodo gateway generan una secuencia de mensajes. Las secuencias generadas se intercambian entre dispositivos con el fin de comprobar el funcionamiento de los dispositivos. Las mismas secuencias son las que se ingresan al simulador. El simulador devuelve una secuencia de mensajes según el procedimiento, sin embargo, no siempre una transición de estados retorna un mensaje; por lo cual se utiliza un par de caracteres (NS) para indicar que no hay salida o respuesta ante la recepción de un mensaje o señal

especifica. Las secuencias de salida del simulador son similares a las que secuencias de mensajes MQTT-SN enviados por los nodos.

A continuación, se presentan los resultados de las capturas del sniffer, donde se muestra las entradas, salidas, y estados que manejan los nodos cliente y gateway. Las capturas realizadas se contrastan con los resultados de las simulaciones. Se presentan las capturas realizadas por medio del sniffer y las simulaciones de los procedimientos más importantes del protocolo, el resto de las pruebas están ubicadas en el ANEXO B.

3.3.1 PROCEDIMIENTO ANUNCIO Y DESCUBRIMOS DEL GATEWAY

3.3.1.1 Gateway

Simulación: La simulación de este procedimiento muestra los diferentes estados por los que pasa el nodo ante la recepción de una secuencia de entrada, la cual contiene varias señales que deben ser generadas internamente por el nodo, la secuencia también permite que el nodo llegue al estado Espera Anuncio donde permanecerá enviando mensaje *ADVERTISE* mientras le llegue las señales que indican la terminación de un temporizador *TADV.*

Secuencia de entrada: brokerListo=1_*SEARCHGW*_TADV_TADV_

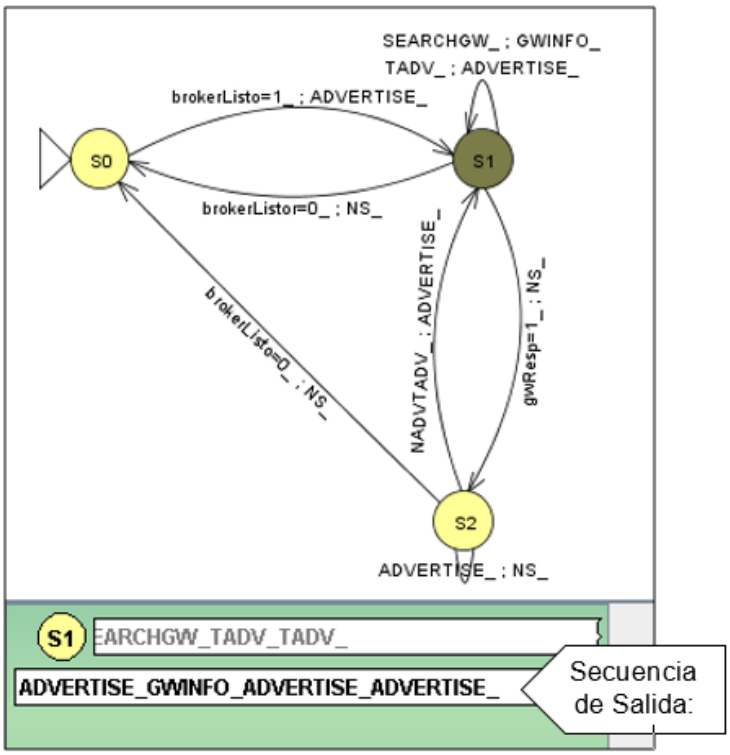


Figura 3.4 Simulación (Gateway): Procedimiento anuncio y descubrimos del gateway.

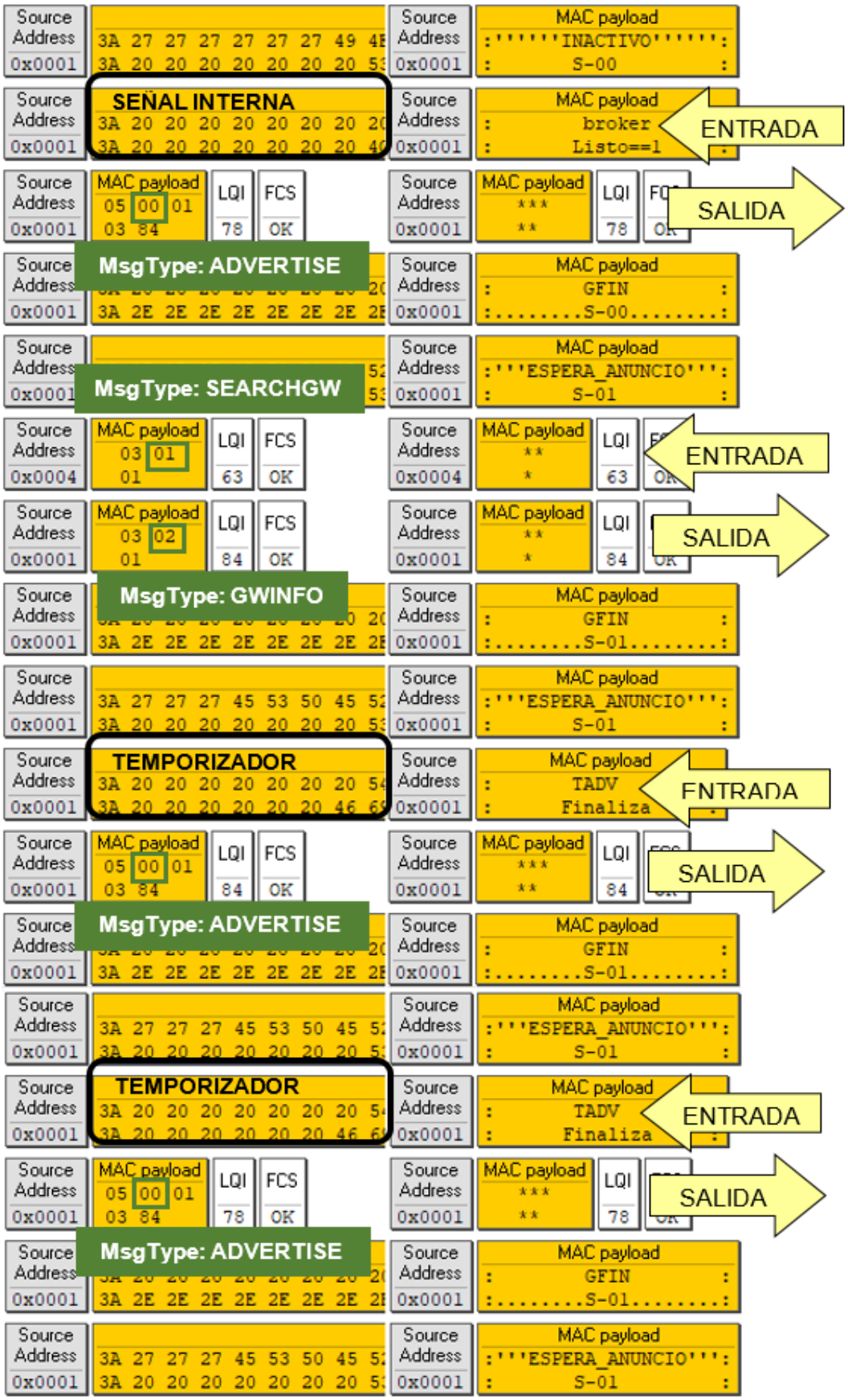


Figura 3.5 Capturas (Gateway): Procedimiento anuncio y descubrimos del gateway.

Capturas Sniffer: En las capturas se puede verificar que el nodo se comporta de la misma forma que la simulación, ante la recepción de la misma secuencia. El nodo pasa de estado Inactivo (S0) al estado Espera Anuncio (S1), estado en el que permanece mientras le lleguen las señales del temporizador *TADV.* Al estado S2 solo se llega cuando se realiza la configuración respectiva del código para el nodo gateway.

La señal interna que debería ser enviada por el broker es generada mediante el pulsador del nodo, mientras que las señales *TADV* se generan automáticamente al terminar los temporizadores previamente configurados. Las secuencias de entrada y salida de mensajes MQTT-SN se muestran a continuación y se las puede verificar en la Figura 3.5.

Secuencia de señales y mensajes recibidos: brokerListo=1, *SEARCHGW*, TADV, TADV

Secuencia de salida: *ADVERTISE*, *GWINFO*, *ADVERTISE*, *ADVERTISE*

3.3.1.2 Cliente

Simulación: En la simulación muestra los diferentes estados por los que pasa el nodo ante la recepción de una secuencia de entrada. La secuencia de entrada contiene varios mensajes MQTT-SN y una señal de temporizador, la secuencia permite pasar del estado S0 al estado S1 para regresar al estado S0 nuevamente.

Secuencia de entrada: T*SEARCHGW_GWINFO_ADVERTISE_ADVERTISE_*

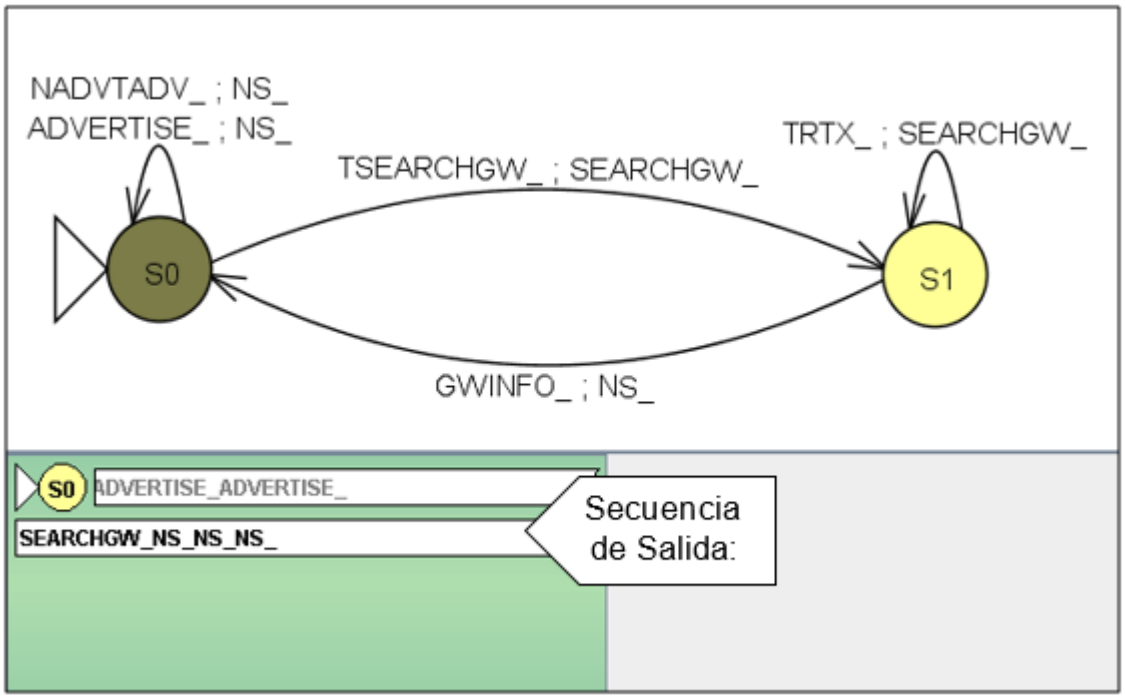


Figura 3.6 Simulación (Cliente): Procedimiento anuncio y descubrimos del gateway.

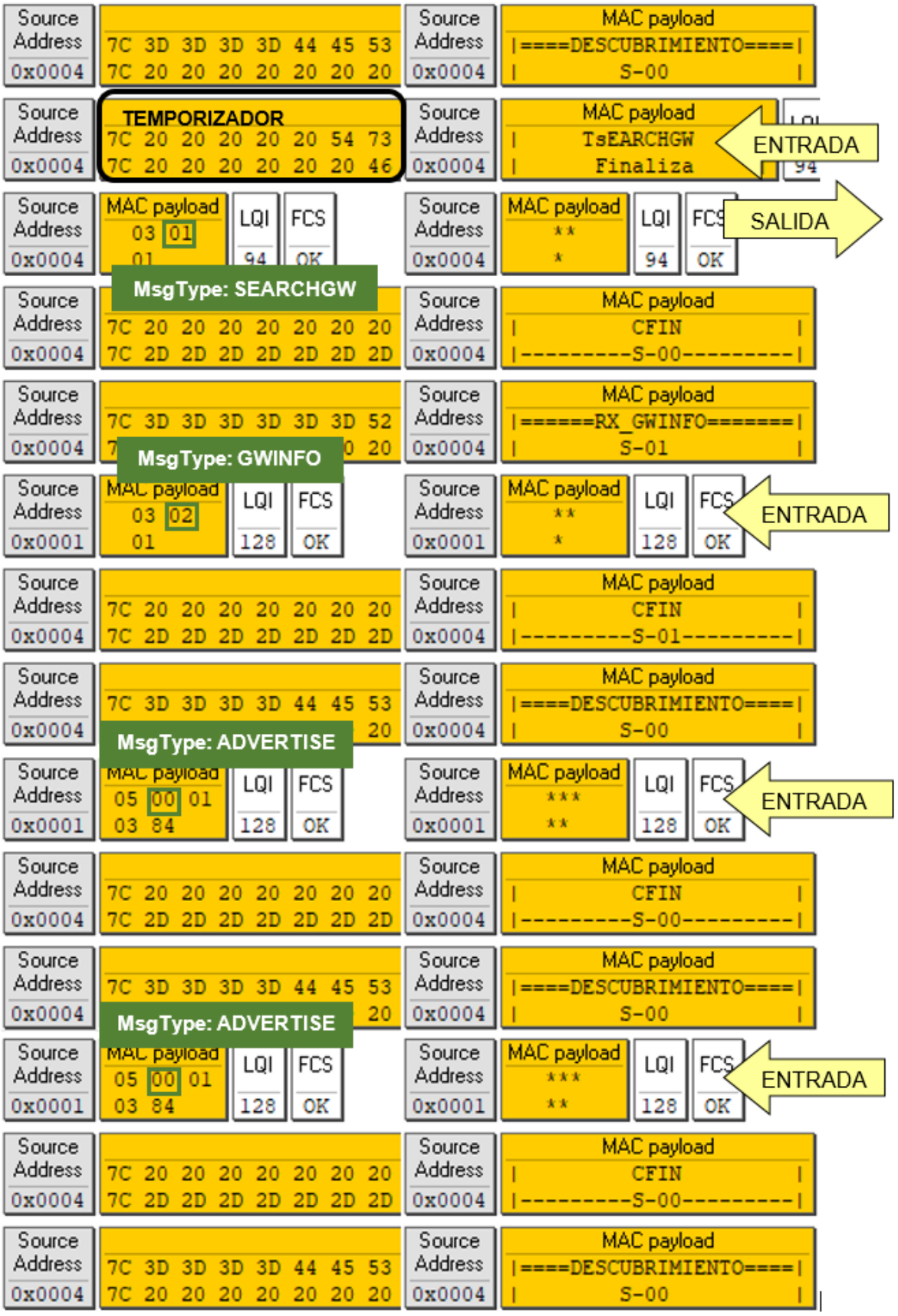


Figura 3.7 Capturas (Cliente): Procedimiento anuncio y descubrimos del gateway.

Capturas Sniffer: Las capturas muestran que el nodo se comporta de la misma forma que la simulación, ante la recepción de la misma secuencia. El nodo pasa del estado Descubrimiento (S0) al estado RX_*GWINFO* (S1) para a continuación, regresar al estado inicial. La señal *TSEARHGW* se genera automáticamente al terminar el temporizador previamente configurado. Cuando el nodo retorna al estado inicial se puede dar por concluido el procedimiento. Las secuencias de entrada y salida se muestran a continuación, la Figura 3.7 permite verificarlas.

Secuencia de señales y mensajes recibidos: T*SEARCHGW*, *GWINFO*, *ADVERTISE*, *ADVERTISE*

Secuencia de salida: *SEARCHGW*

3.3.2 CONFIGURACIÓN DE CONEXIÓN DEL CLIENTE

3.3.2.1 Gateway

Simulación: En la simulación se muestra los cambios de estado de la máquina de estados requerida para que un cliente desea conectarse a un gateway. Se recibe una secuencia de entrada la cual solo contiene mensajes MQTT-SN, sin embargo, se especifica que el mensaje *CONNECT* contiene un campo will establecido en uno. El campo will permite la transición del estado S0 hacia el estado S1 y posteriormente al estado S2. Finalmente, la MEF regresa al estado inicial ya que desde este estado puede esperar por otra secuencia de conexión.

Secuencia de entrada: *CONNECT*(willFlag=1)_*WILLTOPIC_WILLMSG_*

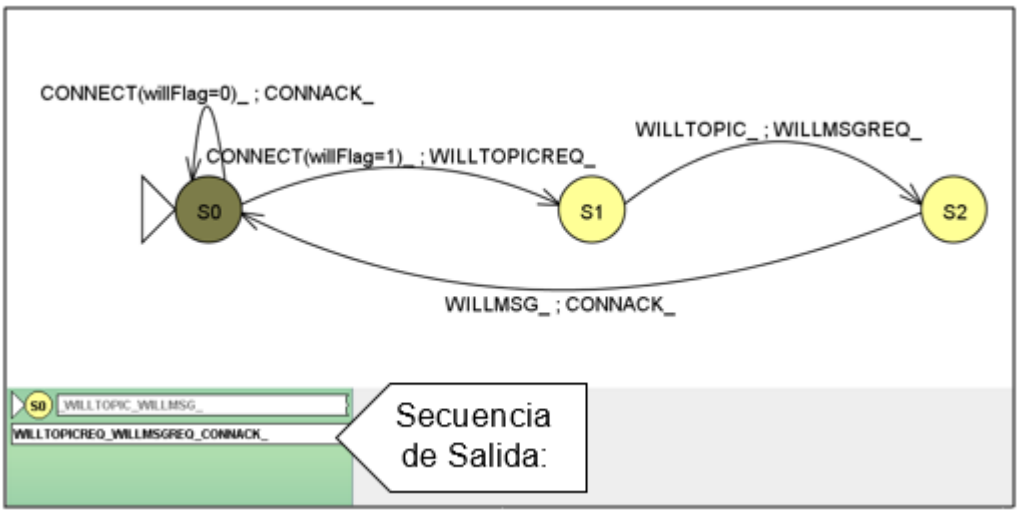


Figura 3.8 Simulación (Gateway): Configuración de conexión del cliente.

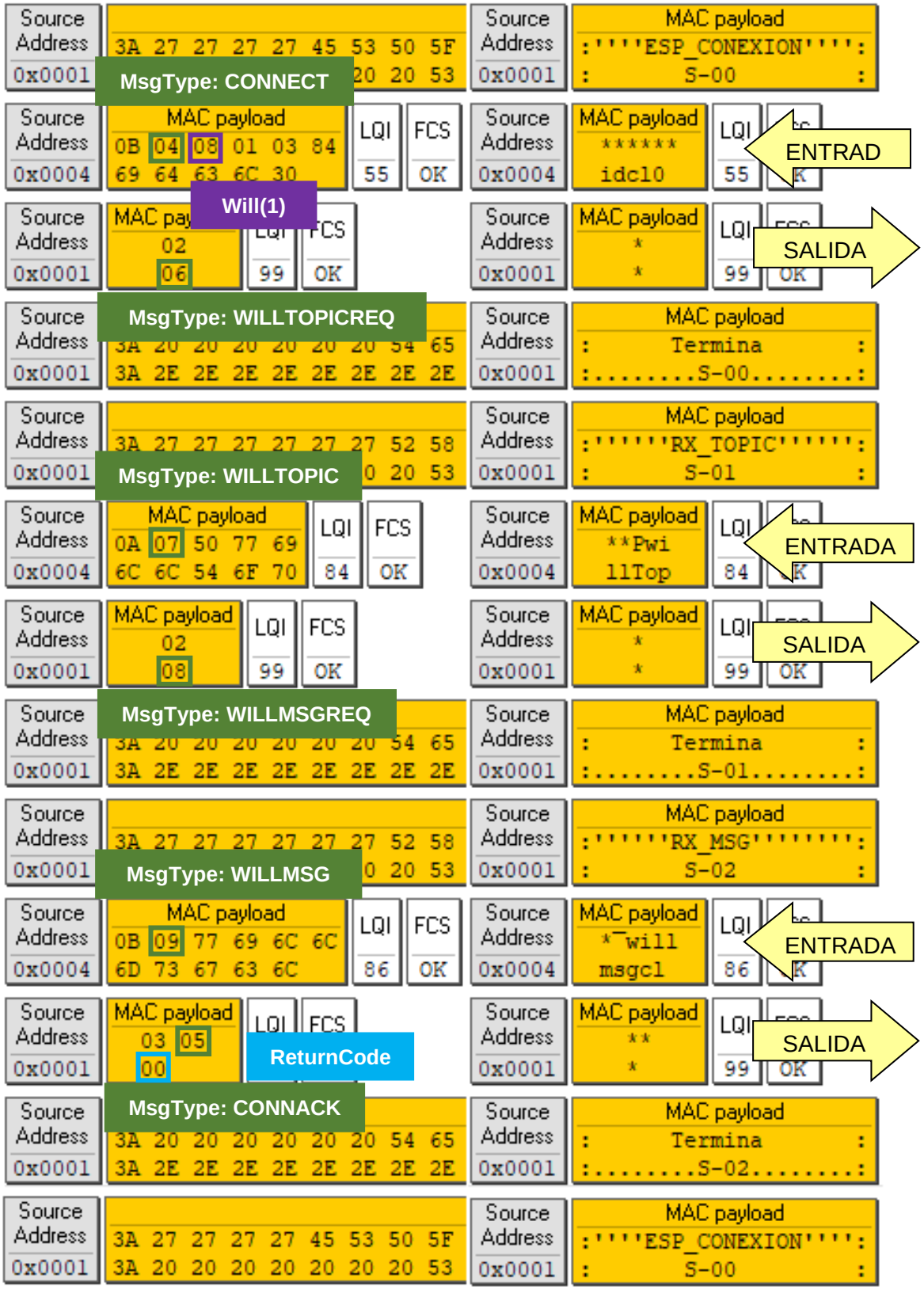


Figura 3.9 Capturas (Gateway): Configuración de conexión del cliente.

Capturas Sniffer: Mediante el sniffer se puede ver que el nodo se comporta de la misma forma que en la simulación, si recibe la misma secuencia. El nodo empieza en estado Espera Conexión (S0) y al recibir un mensaje *CONNECT* con la bandera will establecida en uno (campo will =0b00001000 o 0x08), pasara a los estados RX_TOPIC y RX_MSG según la recepción de la respectiva secuencia de mensajes enviada por el cliente. El nodo regresa al estado inicial después de aceptar la conexión mediante en mensaje *CONNACK* (campo returnCode=0x00), se puede esperar por otra secuencia de mensajes enviada por el cliente. Las secuencias de entrada y salida se muestran a continuación, Figura 3.9 permite verificarlas.

Secuencia de señales y mensajes recibidos: *CONNECT*(willFlag=1), *WILLTOPIC*, *WILLMSG*

Secuencia de salida: *WILLTOPICREQ*, *WILLMSGREQ*, *CONNACK*

3.3.2.2 Cliente

Simulación: La simulación muestra las diferentes transiciones de estado que debe realizar un cliente si quiere conectarse a un gateway. La secuencia de mensajes contiene una señal, que indica el valor de la bandera will antes de enviar el mensaje *CONNECT*. Según el valor de la bandera se producirá un cambio de estado. En nuestro caso la secuencia indica que la bandera will fue establecida en uno por lo cual pasara a los estados S1, S2, y S3. Cuando la MEF se encuentra en estado S3 regresa al estado inicial ya que recibe un mensaje *CONNACK* sin rechazos por congestión.

Secuencia de entrada: Willflag=1_*WILLTOPICREQ*_*WILLMSGREQ*_*CONNACK*_

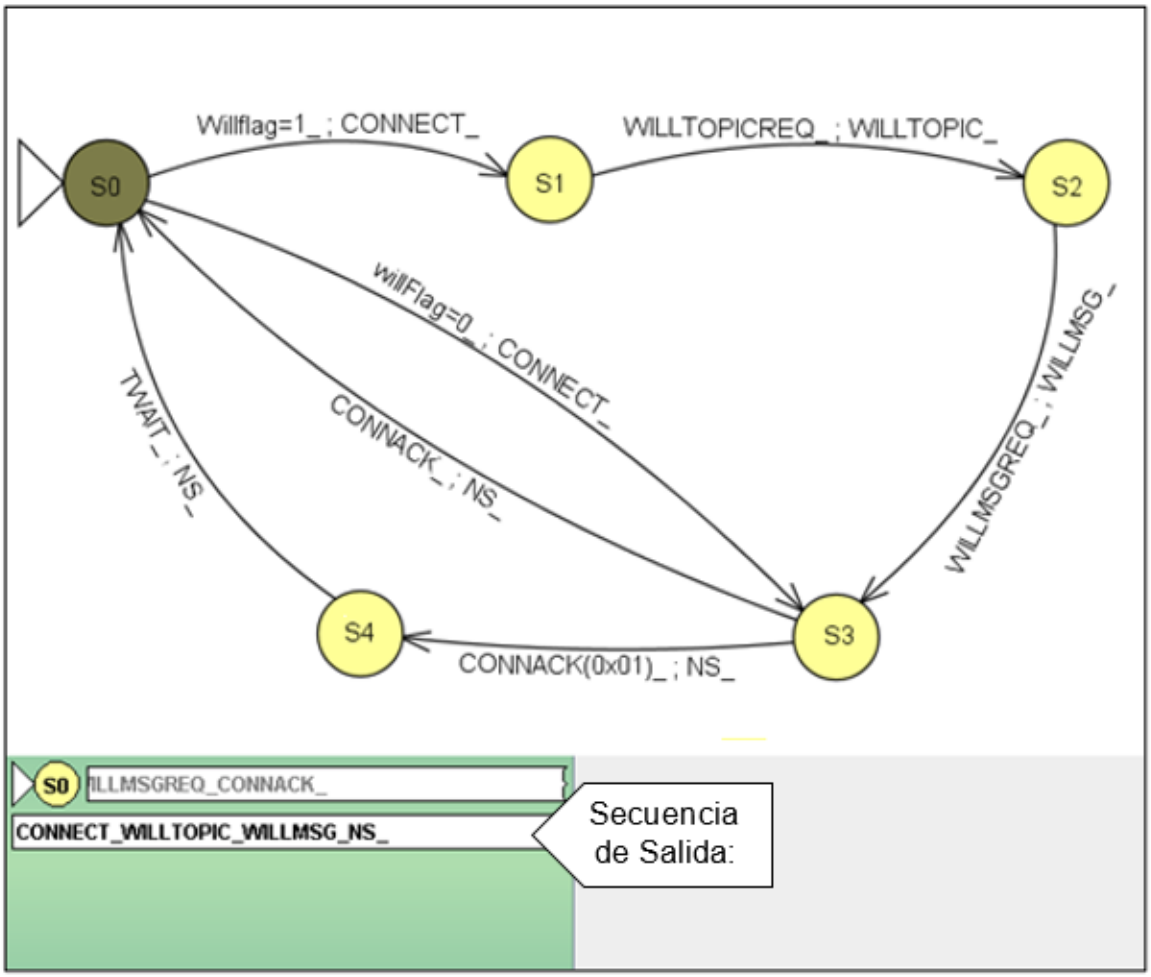


Figura 3.10 Simulación (Cliente): Configuración de conexión del cliente.

Capturas Sniffer: Las capturas muestran que el nodo inicia su operación en el estado Establecer Conexión(S0) para después pasar al estado RX_TOPIC_REQ(S1), después de enviar un mensaje *CONNECT* con la bandera will establecida en uno (campo will=0b00001000 o 0x08). La señal que indica el valor de la bandera es recreada por medio del pulsador del nodo, es enviado un mensaje que indica que la señal ha sido recibida. De esta forma el nodo sabe a qué estado avanzar.

Según la secuencia recibida el nodo pasa del estado S1 al estado RX_MSG_REQ(S2), hasta llegar al estado RX_*CONNACK*(S3) donde espera el mensaje *CONNACK* con el campo returnCode= 0x00. El nodo regresa al estado inicial donde puede reiniciar el proceso de conexión.

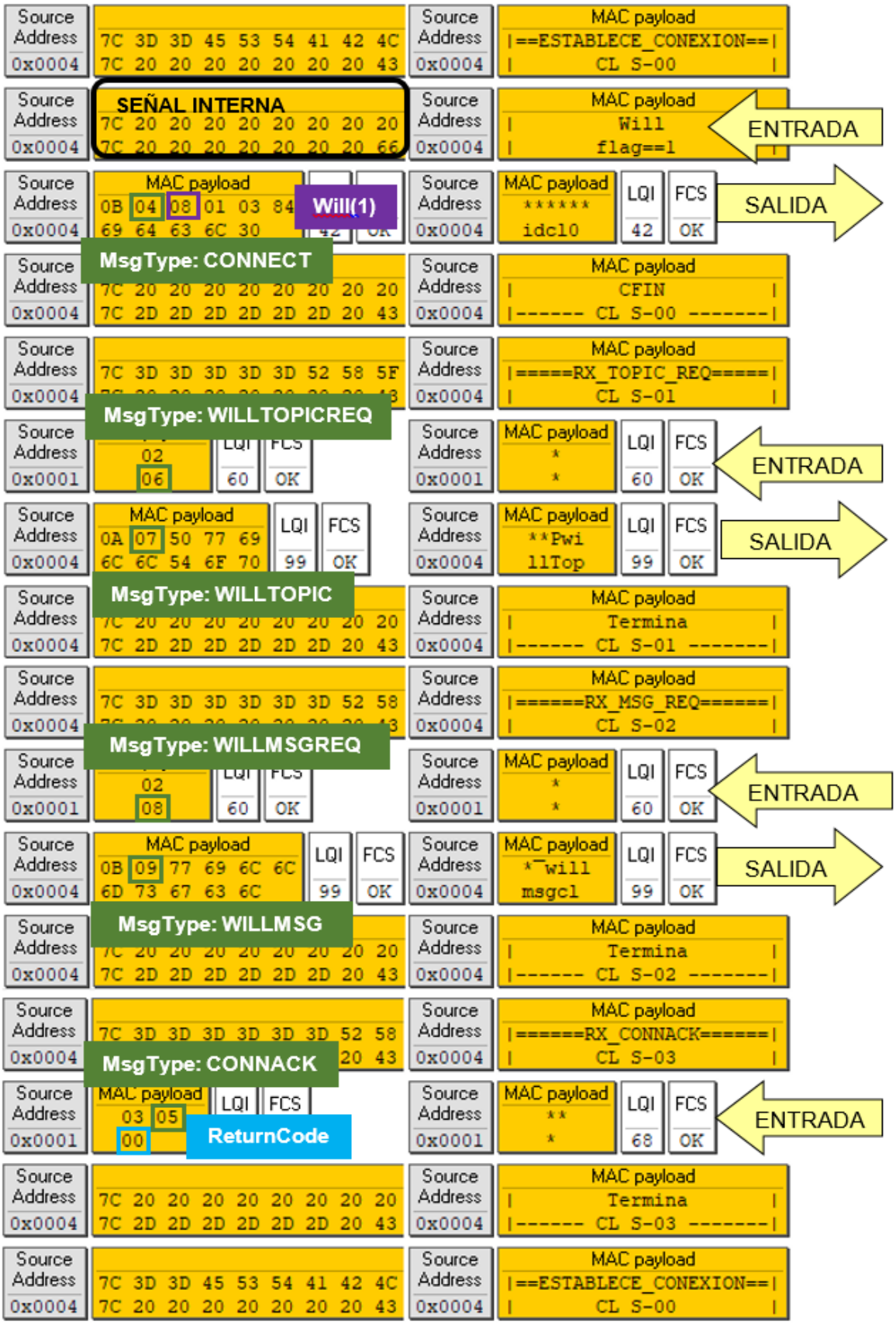


Figura 3.11 Capturas (Cliente): Configuración de conexión del cliente.

Si el mensaje recibido en el estado S3 es un *CONNACK* con un campo returnCode= 0x01 el nodo pasara al estado Descongestión, donde activara un temporizador y esperara a que este termine para regresar al estado inicial y poder reiniciar el procedimiento. Las secuencias de entrada y salida se muestran a continuación, la Figura 3.11 permite verificarlas.

Secuencia de señales y mensajes recibidos: Willflag=1, *WILLTOPICREQ*, *WILLMSGREQ*, *CONNACK*

Secuencia de salida: *CONNECT*, *WILLTOPIC*, *WILLMSG*

3.3.3 PROCEDIMIENTO PARA REGISTRAR NOMBRES DE TEMAS

3.3.3.1 Gateway

Simulación: La simulación muestra las diferentes transiciones que realiza un gateway cuando atiende un registro y requiere registrar un nombre de tema. Se utiliza una secuencia donde, el primer mensaje permite que se responda con un mensaje *REGACK*, para después retornar al estado inicial S0. La siguiente entrada corresponde a una señal que permite iniciar un registro con el cliente, por lo cual el gateway pasa al estado S1 a esperar el último mensaje de la secuencia. Como en procedimientos anteriores también se toma en cuenta rechazos por congestión.

Secuencia de entrada: *REGISTER*_listoRegistro=1_*REGACK*_

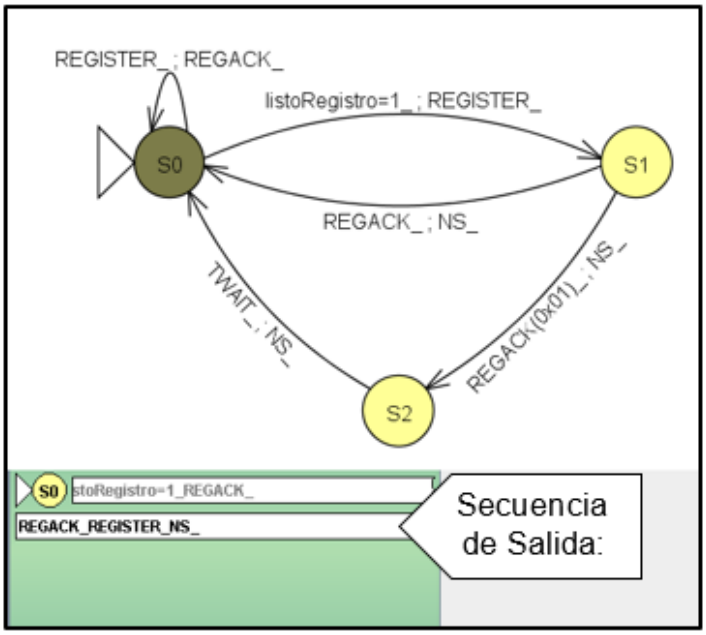


Figura 3.12 Simulación (Gateway): Procedimiento para registrar nombres de temas.

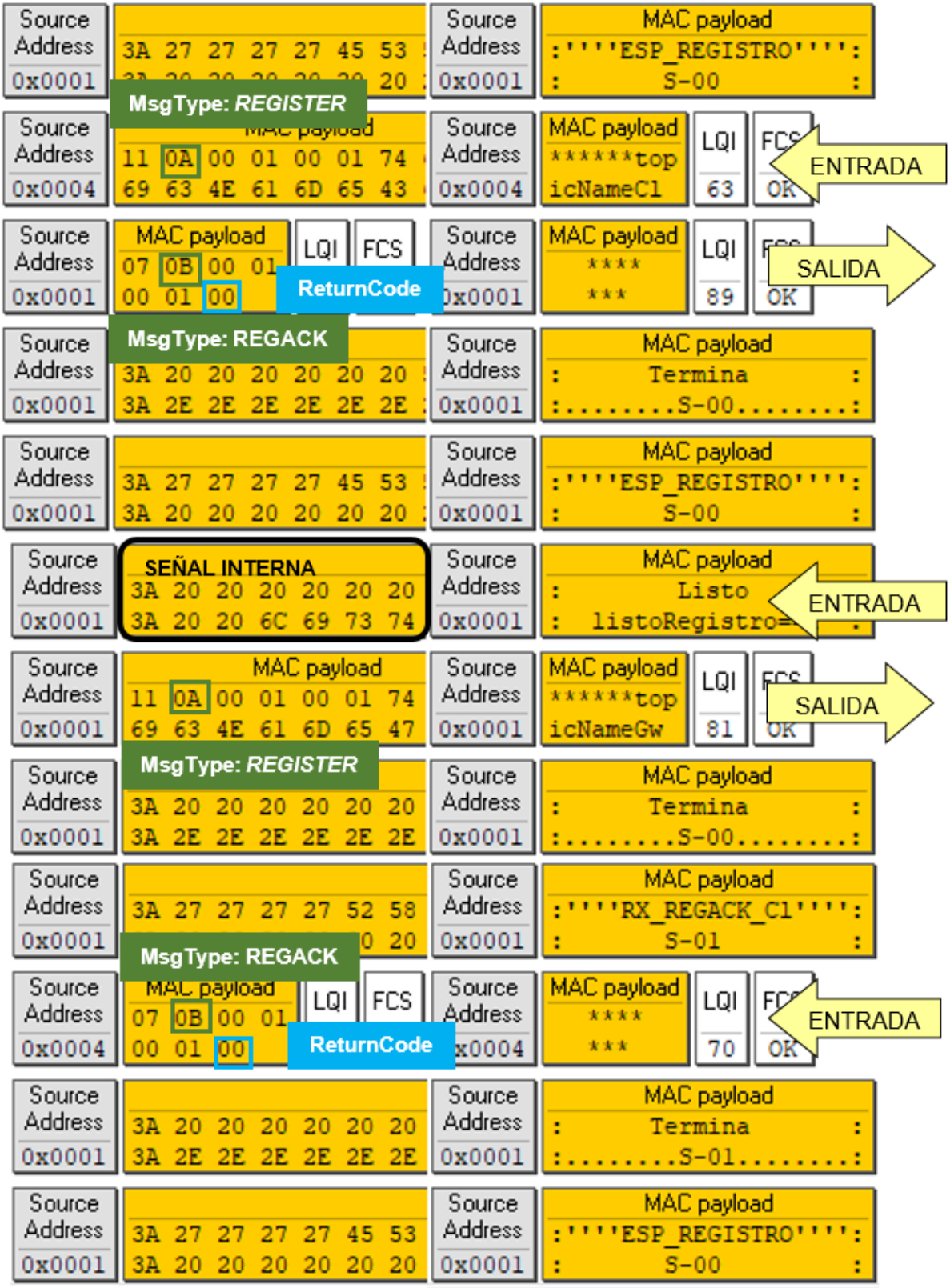


Figura 3.13 Capturas (Gateway): Procedimiento para registrar nombres de temas.

Capturas Sniffer: En las capturas se puede apreciar el comportamiento del nodo gateway ante la recepción de la misma secuencia de mensajes de la simulación. El nodo inicia en el estado Espera Registro(S0) donde primero recibe un mensaje *REGISTER* para después retornar al mismo estado. Después, en el estado inicial se recibe una señal que indica que está listo un mensaje de registro para ser enviado hacia nodo cliente. Luego de enviar el mensaje *REGISTER* el nodo pasa al estado RX*_REACK*(S1) donde espera su respectiva respuesta. Finalmente, el nodo regresa al estado inicial donde se puede iniciar un nuevo procedimiento de registro. Las secuencias de entrada y salida se muestran a continuación, la Figura 3.13 permite verificarlas.

Secuencia de señales y mensajes recibidos: *REGISTER*, listoRegistro=1, *REGACK*

Secuencia de salida: *REACK*, *REGISTER*

3.3.3.2 Cliente

Simulación: La simulación muestra las diferentes transiciones que realiza un cliente cuando requiere registrar un nombre de tema o atender un registro. De manera similar al gateway se utiliza una secuencia donde, la primera entrada corresponde a una señal que permite iniciar un registro con el gateway, por lo cual el cliente pasa al estado S1 a esperar el segundo mensaje de la secuencia. El último mensaje permite que se responda con un mensaje *REGACK*, para después retornar al estado inicial S0.

Secuencia de entrada: listoRegistro=1*_REGACK_REGISTER_*

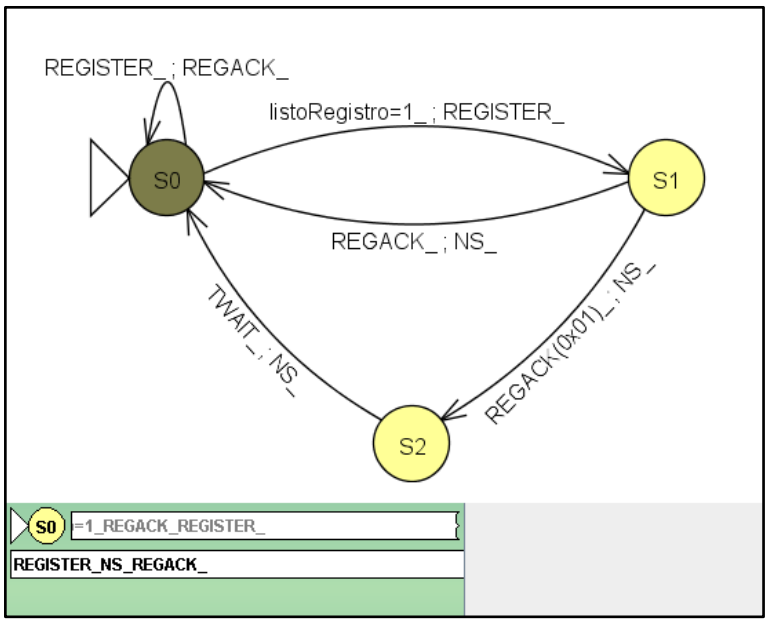


Figura 3.14 Simulación (Cliente): Procedimiento para registrar nombres de temas.

Capturas Sniffer: En las capturas se puede apreciar el comportamiento del nodo cliente ante la recepción de la misma secuencia de mensajes de la simulación. El nodo inicia en el estado Espera Registro(S0) se recibe una señal que indica que está listo un mensaje de registro para ser enviado hacia el nodo gateway. Después, de enviar el mensaje *REGISTER* el nodo pasa al estado RX_*REACK*(S1) donde espera su respectiva respuesta, para regresar al estado inicial. Luego, en el estado inicial recibe un mensaje *REGISTER* para enviar un *REACK* y después retornar al mismo estado. Los campos returCode de los mensajes *REGISTER* han sido establecidos en 0x00 de esta forma se asume que no existen problemas por rechazos a los mensajes *REGISTER.*

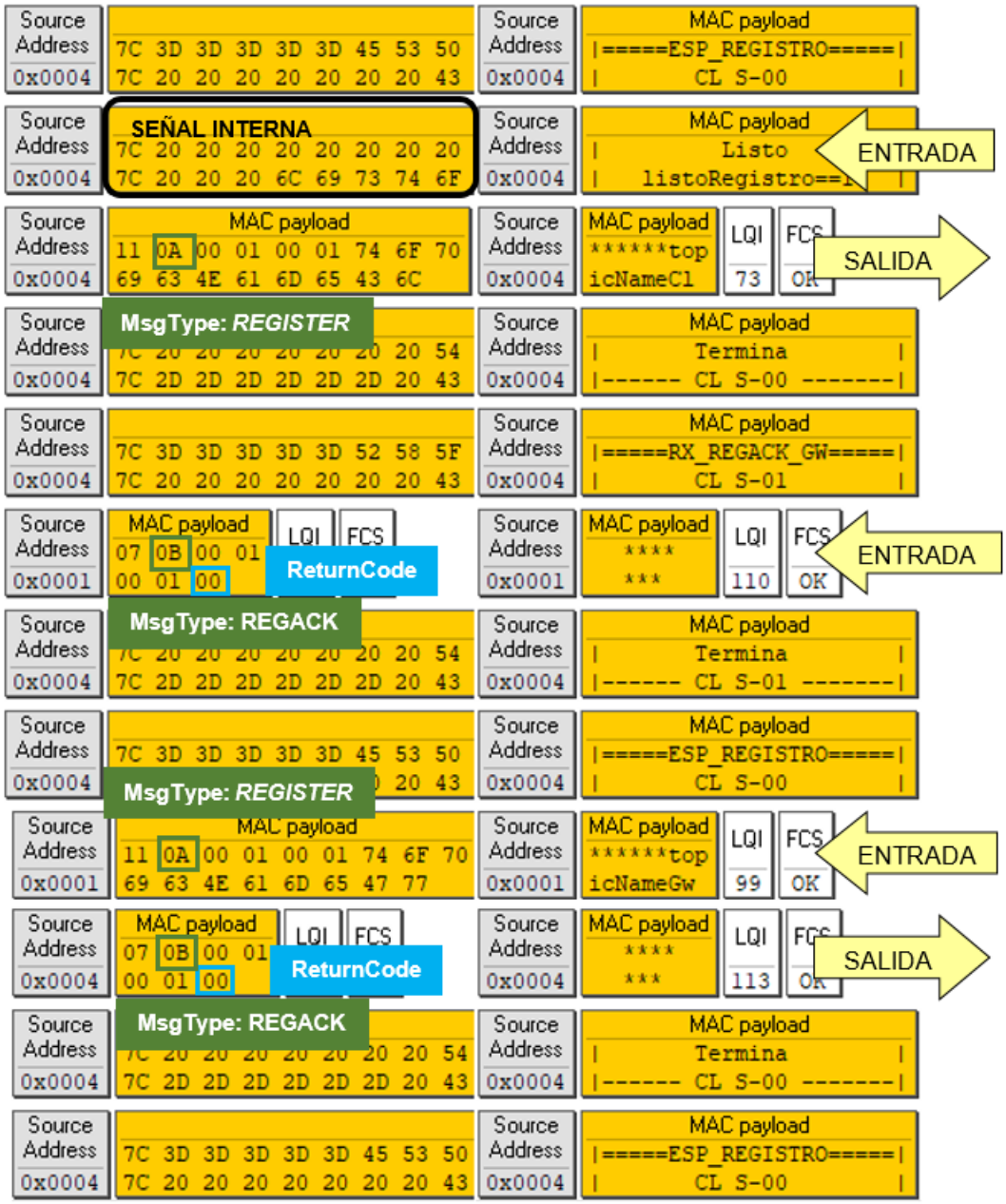


Figura 3.15 Capturas (Cliente): Procedimiento para registrar nombres de temas.

Desde el estado inicial el nodo puede iniciar un nuevo procedimiento de registro. Las secuencias de entrada y salida se muestran a continuación, la Figura 3.15 permite verificarlas.

Secuencia de señales y mensajes recibidos: listoRegistro=1, *REGACK*, *REGISTER*

Secuencia de salida: *REGISTER*, *REACK*

3.3.4 PROCEDIMIENTO DE PUBLICACIÓN(CLIENTE)

3.3.4.1 Gateway

Simulación: La simulación muestra el comportamiento del gateway cuando este recibe mensajes *PUBLISH* con los tres niveles de calidad de servicio. Según la secuencia ingresada, con los mensajes *PUBLISH,* con nivel de QoS 0 y 1, no se presenta un cambio de estado. Mientras que con el mensaje de publicación con nivel QoS 2 se debe pasar del estado inicial al estado S1, donde espera por el mensaje PUREL.

Secuencia de entrada: *PUBLISH*(QoS0)_*PUBLISH*(QoS1)_*PUBLISH*(QoS2)_*PUBREL*_

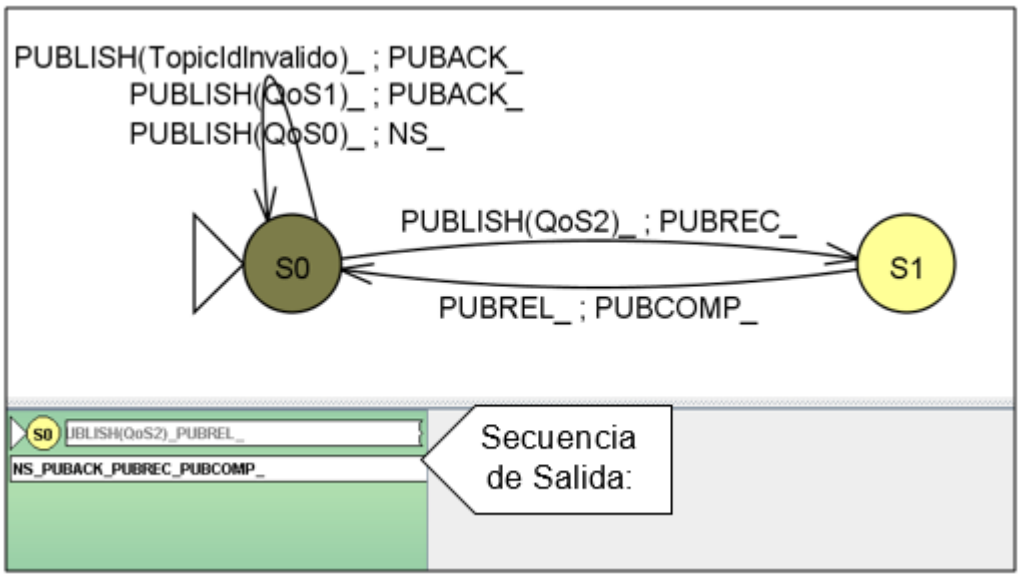


Figura 3.16 Simulación (Gateway): Procedimiento de publicación(cliente).

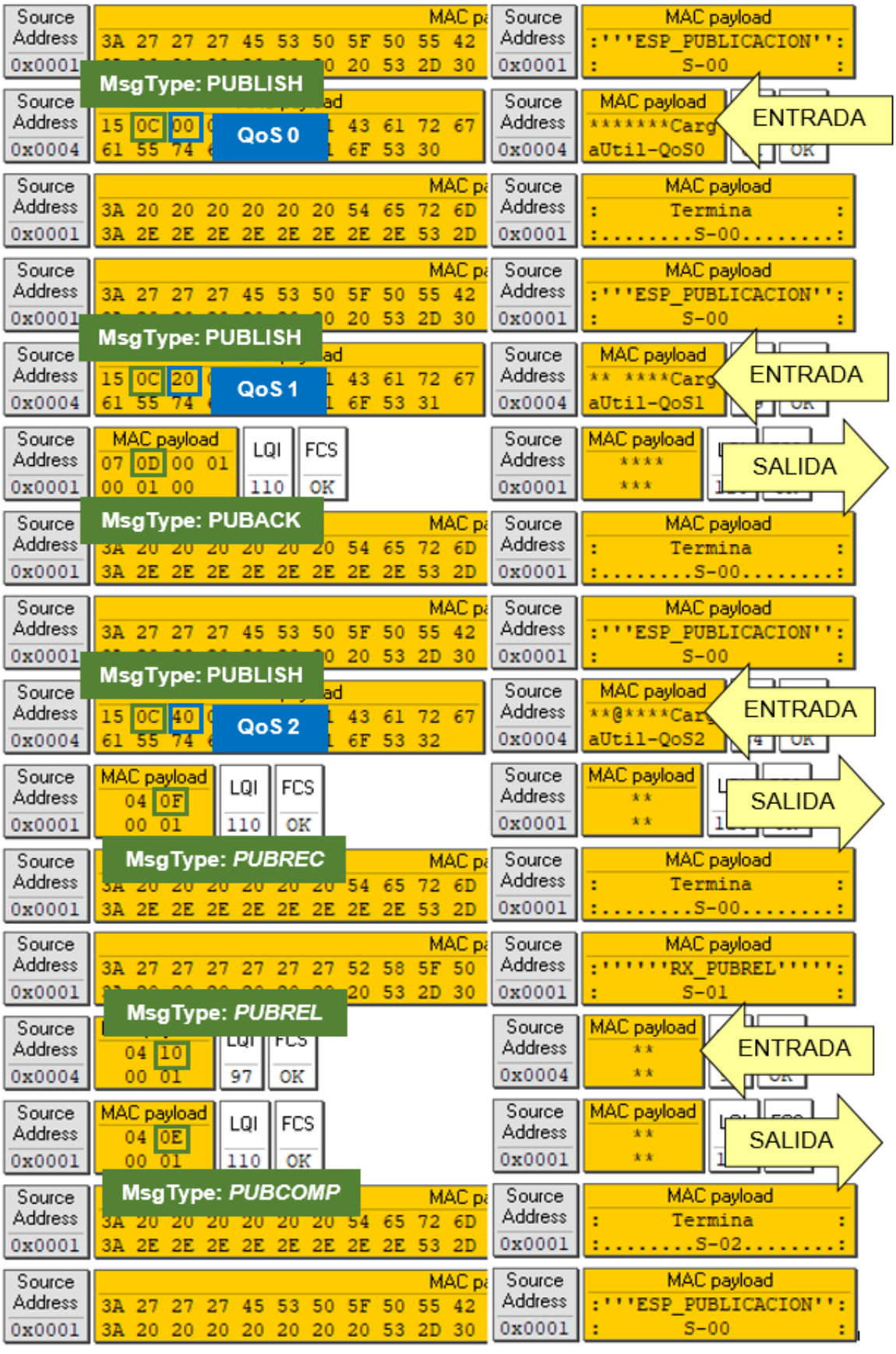


Figura 3.17 Capturas (Gateway): Procedimiento de publicación(cliente).

Capturas Sniffer: Las capturas muestran el comportamiento del nodo ante la recepción de mensajes *PUBLISH* con nivel de calidad de servicio 0, 1, y 2. Cuando el nodo gateway recibe un mensaje *PUBLISH* con una bandera que indica el nivel de QoS 0 (Campo flags=0x00 o 0b00000000), el nodo no responde con ningún mensaje y se mantiene en el mismo estado Espera Publicación (S0). De igual manera cuando el nodo recibe un mensaje *PUBLISH* con una bandera que indica el nivel de QoS 1 (Campo flags= 0x20 o 0b00100000), el nodo se mantiene en el mismo estado, pero responde con un mensaje *PUBACK.* En el momento que el nodo recibe un mensaje *PUBLISH* con una bandera que indica el nivel de QoS 2 (Campo flags=0x40 o 0b01000000), el nodo pasa de estado S0 al estado RX_*PUBREL*(S1) donde espera su respuesta respectiva respuesta. El nodo regresa al estado inicial después de recibir su mensaje *PUBREL.* Desde el estado inicial el nodo puede iniciar un nuevo procedimiento de publicación. Las secuencias de entrada y salida se muestran a continuación, la Figura 3.17 permite verificarlas.

Secuencia de mensajes recibidos:

PUBLISH(QoS0), *PUBLISH*(QoS1), *PUBLISH*(QoS2), *PUBREL*

Secuencia de salida: *PUBACK*, *PUBREC*, *PUBCOMP*

3.3.4.2 Cliente

Simulación: La simulación muestra los diferentes estados por los que debe pasar el nodo cliente, cuando este debe generar mensajes *PUBLISH* con los diferentes niveles de calidad de servicio. La primera señal de la secuencia de entrada hace que el nodo se mantenga en el estado S0 después de enviar su respectivo mensaje *PUBLISH*. Las dos siguientes entradas de la secuencia hacen que el nodo pase al estado S1 donde espera el *PUBACK* para regresar al estado S0 donde espera recibir una nueva señal. Cuando el nodo recibe la señal de entrada *listaPublicacion=1(QoS2)* el nodo pasa al estado S1 a esperar un mensaje *PUBREC*, recibido dicho mensaje pasa al esto S2 donde recibe un mensaje *PUBCOMP* y poder regresar al estado inicial. Se puede notar en la secuencia la máquina debe terminar de realizar una publicación con su respectivo nivel de QoS, para regresar al estado S0 y poder iniciar una nueva publicación.

Secuencia de entrada:
listaPublicacion=1(QoS0)_listaPublicacion=1(QoS1)_*PUBACK*(0x00)_listaPublicacion=1(QoS2)_*PUBREC_PUBCOMP_*

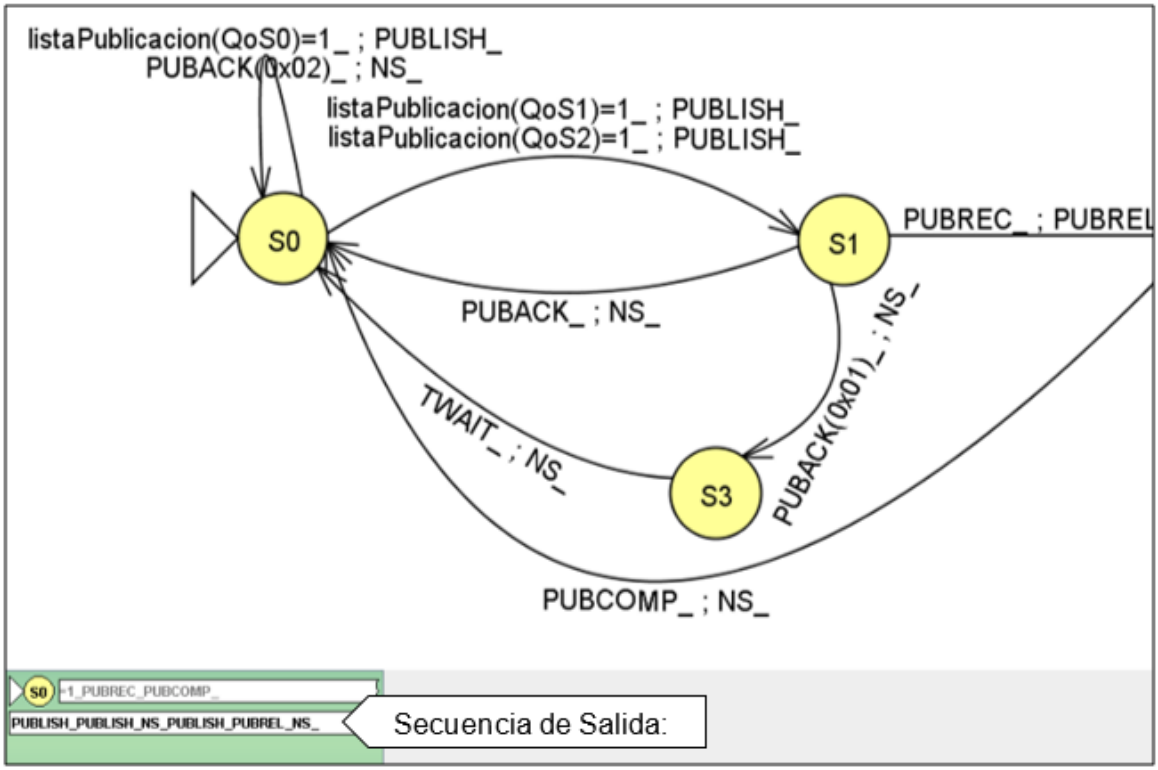


Figura 3.18 Simulación (Cliente): Procedimiento de publicación(cliente).

Capturas Sniffer: Las capturas muestran como el nodo reacciona ante la misma recepción de señales y mensajes de la secuencia de la simulación. Desde el estado inicial Activo Publicación(S0) el nodo espera a que mediante el pulsador se indique el nivel de calidad de servicio con la que se enviará el primer mensaje *PUBLISH.* Para cumplir con la secuencia de simulación y enviar un mensaje con *PUBLISH* con QoS 0 (Campo flags=0x00 o 0b00000000), hay que presionar el pulsador una vez, el nodo tomará este pulso como la señal *listaPublicacion(QoS0)=1.*

Después de enviar la primera publicación el nodo regresa al estado inicial donde se procede a enviar la señal *listaPublicacion(QoS1)=1,* mediante la cual el nodo envía un mensaje *PUBLISH* con QoS 1 (Campo flags=0x20 o 0b00100000). El nodo pasa del estado inicial al estado RX_ACK (S1) donde el nodo cliente espera su respectivo *PUBACK* para a continuación regresar al estado inicial.

Cuando se terminó de enviar la segunda publicación el nodo regresa al estado inicial donde se procede a enviar la señal *listaPublicacion(QoS2)=1,* mediante la cual el nodo envía un mensaje *PUBLISH* con QoS 2 (Campo flags=0x40 o 0b01000000). El nodo pasa del estado inicial al estado RX_ACK (S1) donde el nodo cliente espera recibir un mensaje *PUBREC*. Después de recibir el mensaje de confirmación el nodo transmite el mensaje *PUBREL* para

pasar al estado RX_*PUBCOMP*(S3). Desde el estado S3 el nodo regresa al estado inicial tras recibir su mensaje *PUBCOMP*.

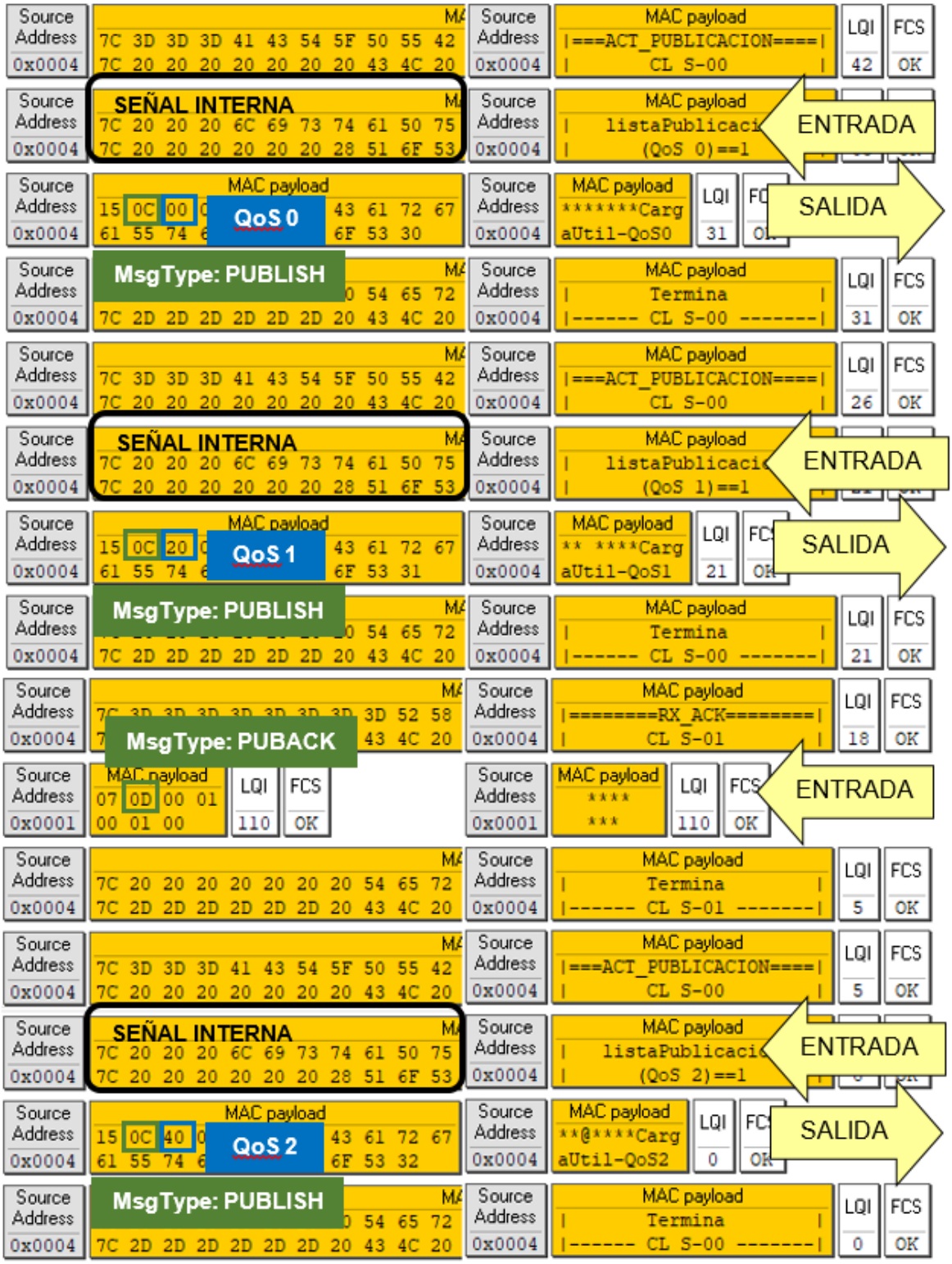


Figura 3.19 Capturas (Cliente Parte 1): Procedimiento de publicación(cliente).

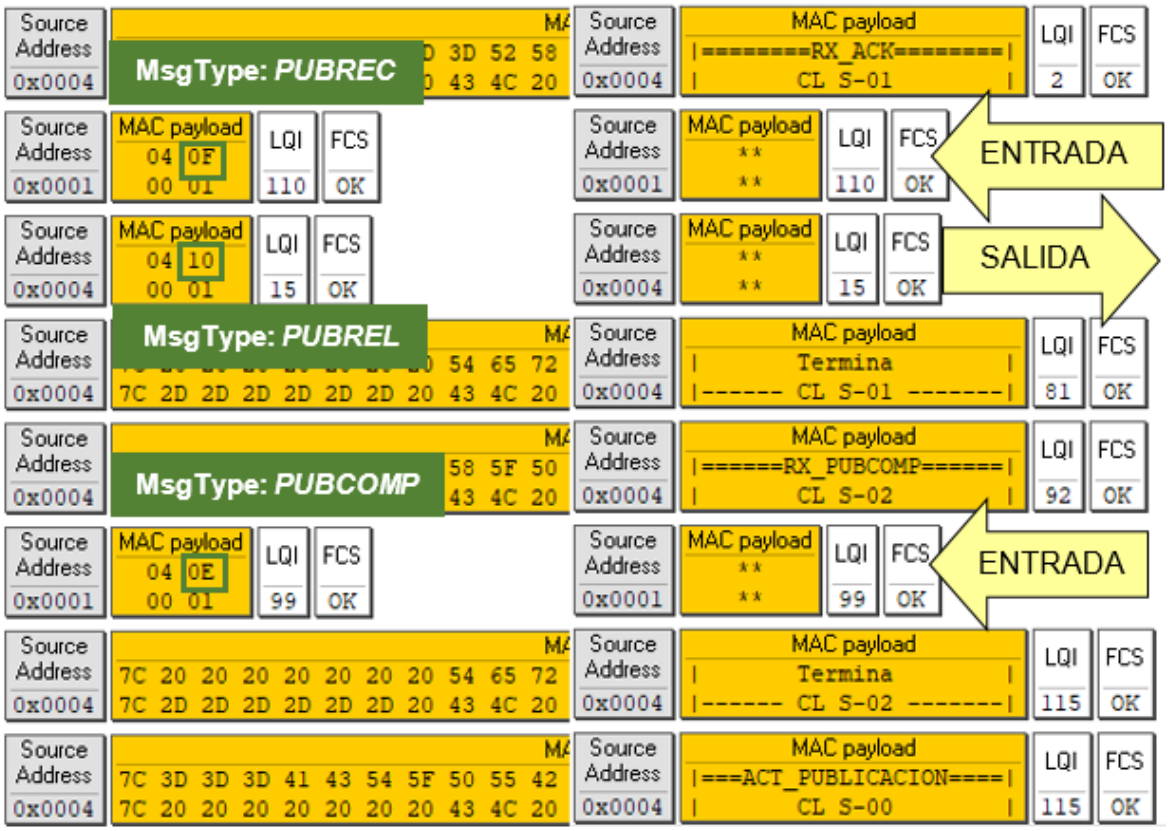


Figura 3.20 Capturas (Cliente Parte 2): Procedimiento de publicación(cliente).

Se puede apreciar que los mensajes *PUBACK* tienen el campo returnCode= 0x00 ya que se asume que todos los mensajes de publicación se han entregado correctamente. También cabe mencionar que el nodo puede pasar a un estado de descongestión tras recibir un *PUBACK* que indique este problema.

Las secuencias de entrada y salida se muestran a continuación, la Figura 3.19 y Figura 3.20 permiten verificarlas.

Secuencia de señales y mensajes recibidos:

listaPublicacion(QoS0)=1, listaPublicacion(QoS1)=1, *PUBACK*(0x00),
listaPublicacion(QoS2) =1, *PUBREC*,*PUBCOMP*

Secuencia de salida: *PUBLISH*, *PUBLISH*, *PUBLISH*, *PUBREL*

3.3.5 PROCEDIMIENTO PARA SUBSCRIBIRSE O DARSE DE BAJA DE UN TEMA

3.3.5.1 Gateway

Simulación: Esta simulación consta de un único estado el cual responde a una secuencia de mensajes enviados por el cliente, la secuencia contiene mensajes que permiten al cliente subscribirse o darse de baja de una subscripción.

Secuencia de entrada: SUSBCRIBE_UNSUSBCRIBE_

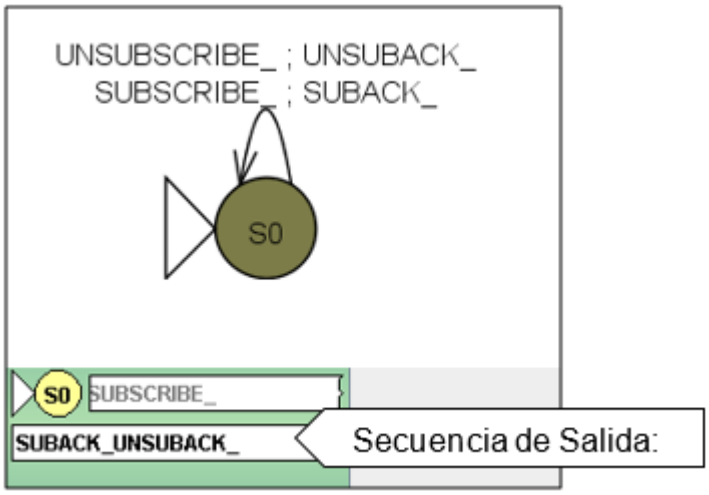


Figura 3.21 Simulación (Gateway): Procedimiento para subscribirse o darse de baja de un tema.

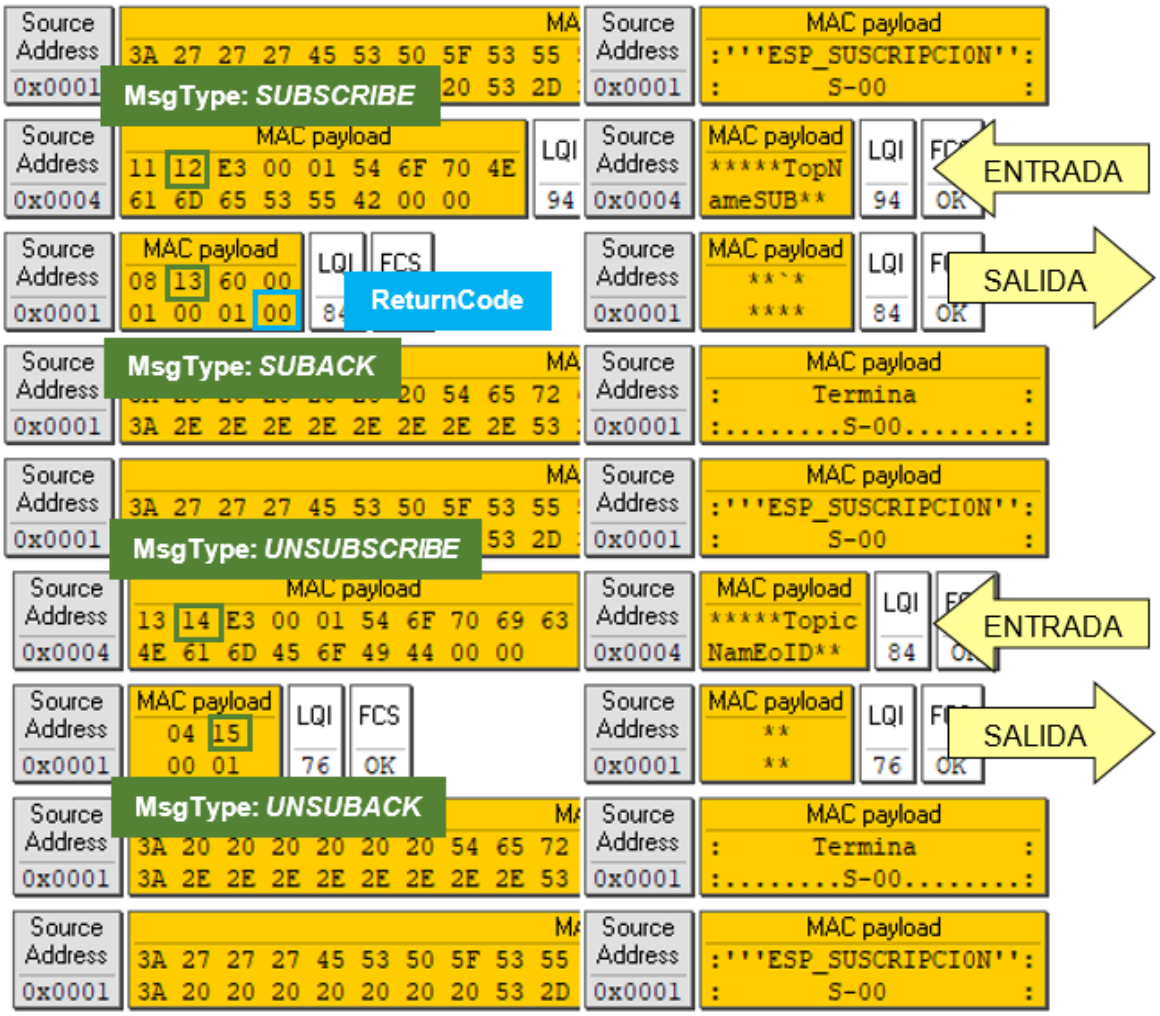


Figura 3.22 Capturas (Gateway): Procedimiento para subscribirse o darse de baja de un tema.

Capturas Sniffer: En las capturas se muestra como el nodo inicia en el estado Espera Subscripción o Dar de Baja (S0), recibe el mensaje SUSBCRIBE, responde con mensaje SUBACK y regresa al estado inicial. Después el nodo recibe UNSUSBCRIBE y responde con un UNSUBACK para retornar al estado inicial. Las secuencias de entrada y salida se muestran a continuación, la Figura 3.22 permite verificarlas.

Secuencia de mensajes recibidos: SUSBCRIBE, UNSUSBCRIBE

Secuencia de salida: *SUBACK*, *UNSUBACK*

3.3.5.2 Cliente

Simulación: La simulación muestra los diferentes estados por los que debe pasar un nodo cliente, cuando requiere subscribiese a un tema o darse de baja de un tema. Por lo cual se puede apreciar que la primera señal secuencia de entrada permite pasar del estado S0 al estado S1, del cual retornara al estado inicial después de recibir su mensaje *SUBACK*. Como se indica en la secuencia, después de recibir el mensaje que permite retornar al estado S0, el nodo puede recibir la señal que permite dar de baja de un tema. Después el nodo pasa al estado S2 donde espera un mensaje *UNSUBACK* para retornar al estado inicial.

Secuencia de entrada: listaSub=1_*SUBACK*_darDeBaja=1_*UNSUBACK*_

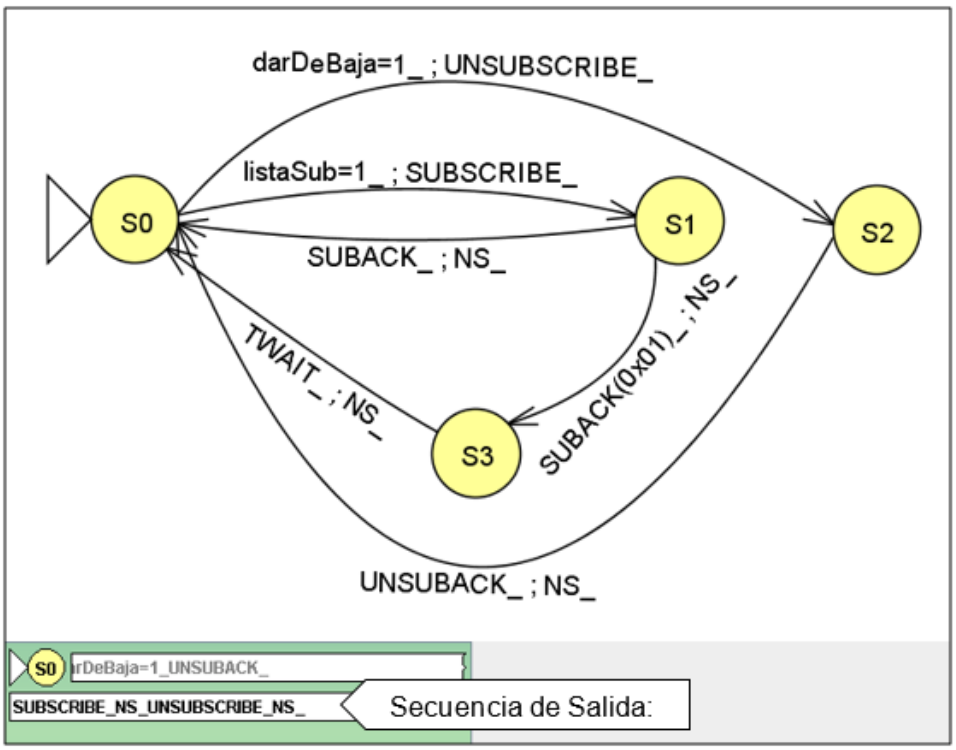


Figura 3.23 Simulación (Cliente): Procedimiento para subscribirse o darse de baja de un tema.

Capturas Sniffer: Las capturas del sniffer muestran el comportamiento del nodo cliente cuando desea subscribirse o darse de baja de un tema, al recibir una secuencia de mensajes y señales igual a la indicada en la simulación. Primero el nodo mediante un pulso recibe la señal que indica que esta lista una nueva subscripción y desde el estado Activo Subscripción (S0) pasa al estado RX_*SUBACK*(S1), después de enviar el mensaje *SUBSCRIBE*. Desde el estado S1 el nodo recibe el mensaje *SUBACK* y regresa al estado inicial donde ya puede recibir la señal para darse de baja de un tema.

Llegado al estado inicial el nodo recibe la señal *darDeBaja=1,* cuando se presiona el pulsador dos veces, para a continuación enviar el mensaje *UNSUBSCRIBE* y pasar al estado RX_*UNSUBACK*(S2). Desde el estado S2 el nodo recibe el mensaje *UNSUBACK* para regresar al estado inicial.

Los mensajes *SUBACK* contienen el campo returnCode= 0x00, por esta razón se considera que no existen rechazos al procedimiento de subscripción. Además, al igual que en procedimientos anteriores se agrega un estado de Descongestión(S3) el cual es utilizado cuando existen rechazos por congestión. Al estado S3 no se puede llegar cuando se envían mensajes *UNSUBSCRIBE.*

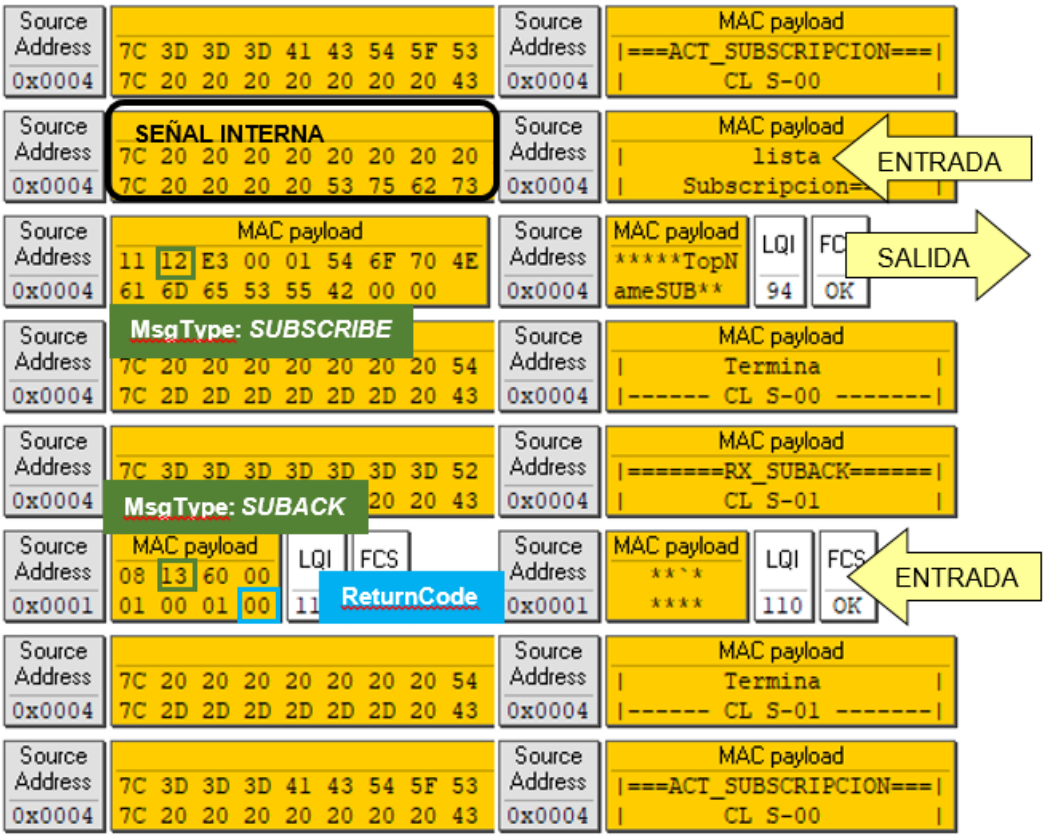


Figura 3.24 Capturas (Cliente Parte 1): Procedimiento para subscribirse o darse de baja de un tema.

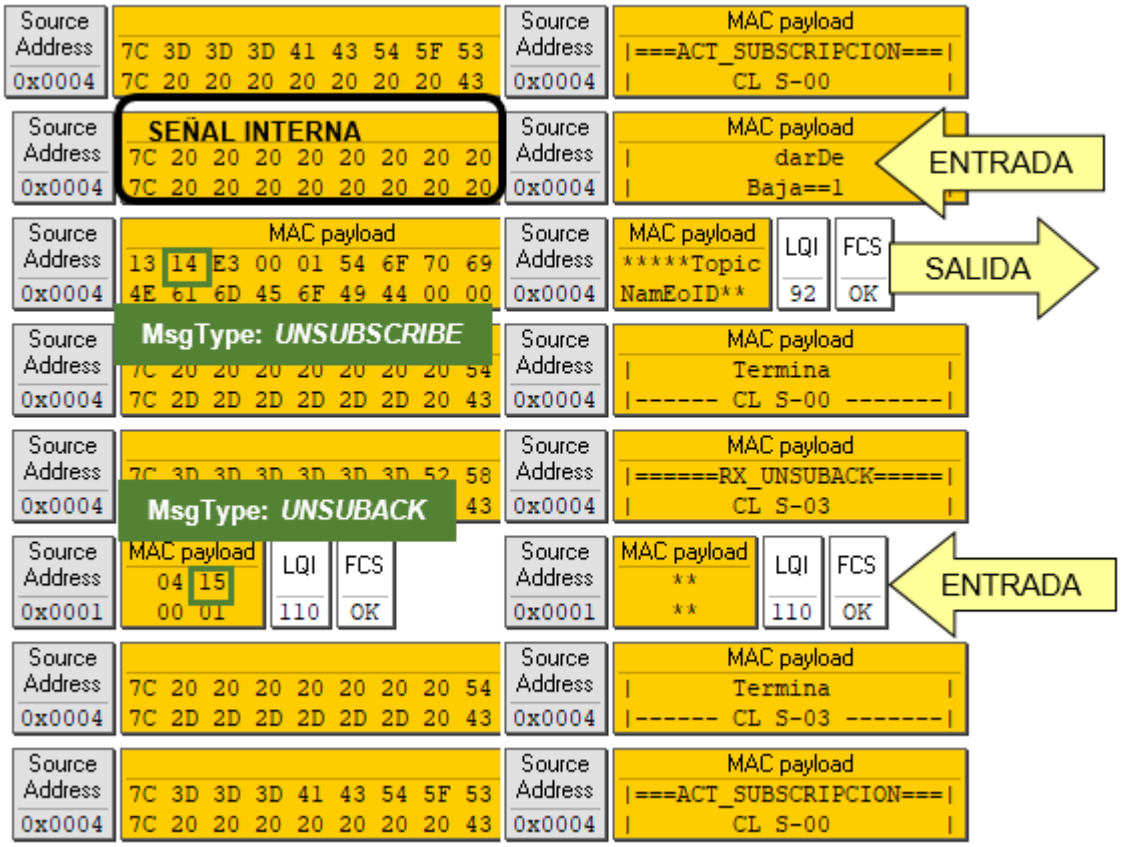


Figura 3.25 Capturas (Cliente Parte 2): Procedimiento para subscribirse o darse de baja de un tema.

Las secuencias de entrada y salida se muestran a continuación, la Figura 3.24 y la Figura 3.25 permite verificarlas.

Secuencia de mensajes recibidos: listaSub=1, *SUBACK*, darDeBaja=1, *UNSUBACK*

Secuencia de salida: *SUBSCRIBE*, *UNSUBSCRIBE*

3.4 ANÁLISIS DE RESULTADOS

En las simulaciones se muestra como la MEF regresa a su estado inicial lo cual indica que la secuencia seleccionada es la correcta, para ese procedimiento en específico. Las MEF que no regresan a su estado inicial son las que llegan a un estado que pertenece a otro procedimiento e indican que la secuencia ingresada es la necesaria para que cumpla con su objetivo.

El funcionamiento de los nodos sobre la topología lineal no presenta ningún problema a la hora de intercambiar tramas 802.15.4 que contienen mensajes MQTT-SN.

Se puede ver que al separar las simulaciones y la codificación de las MEF en procedimientos no habrá problemas, sobre todo en los nodos, por no tener en cuenta un procesamiento completo de un mensaje, así como la implementación da la funcionalidad completa de un procedimiento MQTT-SN.

Se puede ver, mediante las capturas, que los nodos RCB256RFR2 son capases de responder a cada una de las secuencias de señales (temporizadores y pulsadores) y mensajes MQTTSN recibidas, de manera similar a lo visto en las simulaciones. Además, en la mayoría de los casos el nodo regresa a su estado inicial donde pueden recibir una nueva secuencia.

También se puede verificar los cambios de estado que un nodo sufre cuando la transición depende de un campo en específico del mensaje MQTT-SN y no solo del tipo de mensaje, esto indica que la verificación de los campos del mensaje es relativamente sencilla.

4 CONCLUSIONES Y RECOMENDACIONES

4.1 CONCLUSIONES

Realizar una MEF por procedimiento MQTT-SN, para un nodo cliente y nodo gateway, ayuda a una mejor comprensión de la representación SDL, las simulaciones y las pruebas realizadas.

La máquina de Mealy permitió realizar una mejor representación del funcionamiento de cada procedimiento del protocolo MQTT-SN, ya que considera la recepción y transmisión de mensajes MQTT-SN en cada transición de estados. Además, se reduce considerablemente la cantidad de estados que se utilizaría con otro tipo de máquina de estados.

SDL tiene una visión global de la interacción entre el nodo cliente y nodo gateway, lo cual no es posible mediante la máquina de Mealy.

La representación de un procedimiento MQTT-SN mediante una máquina de Mealy es similar a la representación mediante SDL, sin embargo, este último permite representar el funcionamiento de variables, comparadores, temporizadores, y señales adicionales requeridas para la codificación de los nodos.

SDL permite realizar la codificación directa de los nodos RCB256RFR2 sin la necesidad de realizar diagramas de flujo o un pseudocódigo. Y al realizar solo la codificación de estados con sus respectivas transiciones, el código realizado no se considera complicado.

La simpleza de los mensajes MQTT-SN permite que estos puedan ser encapsulados directamente sobre una Trama IEEE802.15.4 a través del nodo RCB256RFR2, para posteriormente utilizarlos como entradas y salidas del nodo cliente y gateway.

Mediante las simulaciones y las capturas realizadas se puede comprobar que los nodos RCB256RFR2 son capaces de responder a las diferentes secuencias de mensajes generadas para cada procedimiento MQTT-SN

Una vez comprobado que los nodos son capaces de responder a las diferentes secuencias de mensajes propias del protocolo MQTT-SN, se puede decir que el funcionamiento completo de cada procedimiento puede ser implementado en los nodos RCB256RFR2.

En la topología lineal planteada los nodos intermedios solo permiten que los mensajes MQTT-SN lleguen a su destino, no se los puede considerar como clientes MQTT-SN ya que solo tienen la capacidad de recibir los mensajes MQTT-SN y reenviarlos a su destino, además si los nodos tuviesen la capacidad de responder a una secuencia de mensajes enviada por el gateway la topología de red pasaría a ser de tipo estrella.

4.2 RECOMENDACIONES

Hay que estudiar a detalle el funcionamiento del protocolo MQTT para entender al protocolo MQTT-SN ya que comparten las muchas características, y en la especificación MQTT-SN no se especifica el funcionamiento de varias características compartidas por ambos protocolos.

Para implementar la funcionalidad completa de cada procedimiento se puede utilizar las funciones de estado, ya que estas funciones contemplan la recepción y transición de mensajes MQTT-SN.

Hay que mencionar que al menos los procedimientos más importantes (procedimiento de registro, publicación y subscripción) del protocolo MQTTSN pueden operar sobre el nodo RCB256RFR2, no obstante, debido a la topología lineal, al servicio de transferencia y recepción de información de los nodos, se debe esperar a que un intercambio de mensajes entre dispositivos, durante la ejecución de un procedimiento, debe terminar antes de poder iniciar otro intercambio de mensajes.

A pesar de que las máquinas de estados están separadas, varios estados tienen una denominación similar con el fin de poder combinar las MEF a futuro, sin embargo, para combinar los diferentes procedimientos es necesario realizar un procesamiento completo de los mensajes MQTT-SN.

Si solo se requiere que el nodo cliente envié mensajes de publicación, una buena opción sería el uso del procedimiento de publicación con nivel de QoS-1 ya que no requiere de otros procedimientos para su funcionamiento.

5 BIBLIOGRAFÍA

[1] C. E. Acosta, F. Gil-Castiñeira, y E. Costa-Montenegro, «Red inalámbrica de sensores con topología lineal sin capa de red», *Rev. Investig. En Tecnol. Inf.*, vol. 9, n.º 17, Art. n.º 17, ene. 2021.

[2] U. Hunkeler, H. L. Truong, y A. Stanford-Clark, «MQTT-S; A *PUBLISH*/*SUBSCRIBE* protocol for Wireless Sensor Networks», *2008 3rd Int. Conf. Commun. Syst. Softw. Middlew. Workshop COMSWARE 08*, pp. 791-798, ene. 2008, doi: 10.1109/COMSWA.2008.4554519.

[3] T. Yildirim, «DATA SHARING USING MQTT AND ZIGBEE-BASED DDS ON RESOURCE-CONSTRAINED CONTIKI-BASED DEVICES», MIDDLE EAST TECHNICAL UNIVERSITY, 2020. [En línea]. Disponible en: https://etd.lib.metu.edu.tr/upload/12625183/index.pdf

[4] J. X. Arciniega Barahona, «DISEÑO E IMPLEMENTACIÓN DE UNA HERRAMIENTA DE MONITOREO Y CAPTURA DE TRAMAS EN REDES IEEE 802.15.4», ESCUELA POLITÉCNICA NACIONAL, Quito, 2016. [En línea]. Disponible en: http://bibdigital.epn.edu.ec/handle/15000/16474

[5] A. Stanford-Clark y H. L. Truong, «MQTT For Sensor Networks (MQTT-SN) Protocol Specification», nov. 2013.

[6] «ITU-T Study Group 10 - Languages for Telecommunication Systems». https://www.itu.int/ITU-T/studygroups/com10/languages/ (accedido 25 de mayo de 2022).

[7] K. J. Turner y J. Wiley, «Using formal description techniques, an introduction to ESTELLE, LOTOS and SDL». enero de 1993. Accedido: 25 de mayo de 2022. [En línea]. Disponible en: https://www.cs.stir.ac.uk/~kjt/using-fdts/using-fdts.pdf

[8] S. Ruehrup, «Network Protocol Design and Evaluation». 2009. [En línea]. Disponible en: http://hondo.informatik.uni-freiburg.de/teaching/vorlesung/protocol-design-s09/slides/04-Protocol_Specification_2.pdf

[9] S. H. Rodger, «JFLAP», julio de 2018. https://www.jflap.org/ (accedido 25 de mayo de 2022).

[10] N. Aakvaag y J.-E. Frey, «Redes de sensores inalámbricos», p. 4, feb. 2006.

[11] E. H. M. Ndoye, F. Jacquet, M. Misson, I. Niang, y C. Universite, «Evaluation of RTS/CTS with unslotted CSMA/CA algorithm in linear sensor networks». septiembre de 2013. [En línea]. Disponible en: https://www.researchgate.net/publication/272816013_Evaluation_of_RTSCTS_with_unslotted_CSMACA_algorithm_in_linear_sensor_networks

[12] «Estándar IEEE 802.15.4: Manual básico » Notas sobre electrónica», *electronics-notes*, 2006. https://www.electronics-notes.com/articles/*CONNECT*ivity/ieee-802-15-4-wireless/basics-tutorial-primer.php (accedido 27 de mayo de 2022).

[13] A. E. Aguaguiña Aconda, «IMPLEMENTACIÓN DE UN ALGORITMO PARA LA RECUPERACIÓN DE FALLOS EN UNA TOPOLOGÍA LINEAL UTILIZANDO EL ESTÁNDAR IEEE 802.15.4», ESCUELA POLITÉCNICA NACIONAL, Quito, 2019. [En línea]. Disponible en: http://bibdigital.epn.edu.ec/handle/15000/20292

[14] A. Koubaa, M. Alves, y E. Tovar, «IEEE 802.15.4 for Wireless Sensor Networks: A Technical Overview», jul. 2005, [En línea]. Disponible en: https://www.researchgate.net/publication/253418115_IEEE_802154_for_Wireless_Sensor_Networks_A_Technical_Overview

[15] «IEEE 802.15.4 Stack User Guide.pdf». Accedido: 1 de junio de 2022. [En línea]. Disponible en: https://www.nxp.com/docs/en/user-guide/JN-UG-3024.pdf

[16] F. D. Cali Reyes, «IMPLEMENTACIÓN DEL ALGORITMO DE PROTOCOLO DE DIRECCIONAMIENTO PARA REDES DE SENSORES INALÁMBRICOS CON EL ESTANDAR IEEE 802.15.4.», ESCUELA POLITÉCNICA NACIONAL, Quito, 2018. [En línea]. Disponible en: http://bibdigital.epn.edu.ec/handle/15000/19061

[17] C. Hervas Parra, «Análisis de rendimiento de protocolos de publicación/subscripción en comunicación con una red de sensores inalámbricos Zigbee», Magister en Redes de Datos, Universidad Nacional de La Plata, 2018. doi: 10.35537/10915/69435.

[18] «Uso de XMPP para el transporte de información». Accedido: 6 de junio de 2022. [En línea]. Disponible en: https://upcommons.upc.edu/bitstream/handle/2099.1/6907/memoria.pdf

[19] Team HiveMQ, «MQTT Protocol | Messaging & Data Exchange for the IoT», *The Messaging and Data Exchange Protocol of the IoT*, marzo de 2013. https://www.hivemq.com/mqtt/mqtt-protocol/ (accedido 7 de junio de 2022).

[20] P. Lea, *Internet of Things for architects: architecting IoT solutions by implementing sensors, communication infrastructure, edge computing, analytics, and security*, First *PUBLISH*ed. Birmingham Mumbai: Packt, 2018.

[21] A. Larmo, A. Ratilainen, y J. Saarinen, «Impact of CoAP and MQTT on NB-IoT System Performance», *Sensors*, vol. 19, n.º 1, p. 7, dic. 2018, doi: 10.3390/s19010007.

[22] M. Calabretta, R. Pecori, M. Vecchio, y L. Veltri, «MQTT-Auth: a Token-based Solution to Endow MQTT with Authentication and Authorization Capabilities», *J. Commun. Softw. Syst.*, vol. 14, n.º 4, 2018, doi: 10.24138/jcomss.v14i4.604.

[23] Openlabpro company, «MQTT Packet Format with examples», *OpenLabPro*, noviembre de 2019. https://openlabpro.com/guide/mqtt-packet-format/ (accedido 8 de junio de 2022).

[24] M. Obula y J. B. Seventline, «Real Time Sensor Data transmission to the IoT Applications using MQTT-SN and MQTT», *Int. J. Recent Technol. Eng.*, vol. 8, n.º 4, pp. 6371-6378, nov. 2019, doi: 10.35940/ijrte.D8834.118419.

[25] L. Llamas, «Qué son y cómo usar los Topics en MQTT correctamente», *Ingeniería, informática y diseño*, 2020. https://www.luisllamas.es/que-son-y-como-usar-los-topics-en-mqtt-correctamente/ (accedido 8 de junio de 2022).

[26] P. Egli, «MQTT - Message Queueing Telemetry Transport Introduction to MQTT, a protocol for M2M and IoT applications», enero de 2017. doi: 10.13140/RG.2.2.13210.54721.

[27] D. Bharatia, «Características fundamentales de MQTT», *Colección de tutoriales y referencias Acervo Lima*. https://es.acervolima.com/caracteristicas-fundamentales-de-mqtt-conjunto-3/ (accedido 11 de junio de 2022).

[28] A. Stanford-Clark y H. L. Truong, «MQTT For Sensor Networks (MQTT-SN) Protocol Specification». noviembre de 2013. [En línea]. Disponible en: https://www.oasisopen.org/committees/download.php/66091/MQTT-SN_spec_v1.2.pdf

[29] «MQTT-SN Description | IOT Broker», *IoTBroker.cloud*, abril de 2017. https://www.iotbroker.cloud/blog/MQTT%20sn/MQTT-SN%20Description (accedido 24 de junio de 2022).

[30] IONOS group, «UML, lenguaje de modelado gráfico», *IONOS Digitalguide*, 1988. https://www.ionos.es/digitalguide/paginas-web/desarrollo-web/uml-lenguaje-unificado-de-modelado-orientado-a-objetos/ (accedido 9 de junio de 2022).

[31] D. Bell, «Explore the UML sequence diagram», *IBM Developer*, febrero de 2004. https://developer.ibm.com/articles/the-sequence-diagram/developer.ibm.com/articles/the-sequence-diagram (accedido 10 de junio de 2022).

[32] Sparx Systems Pty Ltd, «Tutorial UML 2 - Diagrama de Secuencia», *Sparx Systems Enterprise Architect*. http://www.sparxsystems.com.ar/resources/tutorial/uml2_sequencediagram.php (accedido 10 de junio de 2022).

[33] J. A. G. Orozco, «Maquinas de Estados Finitos». Agosto2008. [En línea]. Disponible en: https://docplayer.es/23881038-Maquinas-de-estados-finitos.html

[34] C. Yañez, «Qué es la programación multihilo y qué ventajas tiene», *CEAC*, 21 de febrero de 2018. https://www.ceac.es/blog/que-es-la-programacion-multihilo-y-que-ventajas-tiene (accedido 17 de julio de 2022).

[35] B. Hogrefe, «SDL-88 Tutorial», mayo de 2013. http://www.sdl-forum.org/sdl88tutorial/index.html (accedido 17 de julio de 2022).

[36] S. Ruehrup, «Network Protocol Design and Evaluation». 2009. [En línea]. Disponible en: http://hondo.informatik.uni-freiburg.de/teaching/vorlesung/protocol-design-s09/slides/04-Protocol_Specification_2.pdf

ANEXOS

ANEXO A. Diagramas SDL de cada procedimiento MQTT-SN

ANEXO B. Simulaciones y Capturas

ANEXO C. Código implementado en los nodos RCB256RFR2

ANEXO D. Archivos de simulación JFLAP

ORDEN DE EMPASTADO