Journal of Network and Systems Management (2025) 33:37
https://doi.org/10.1007/s10922-025-09916-1

Integrating Real-Time Wireless Sensor Networks into IoT Using MQTT-SN

Valentin Stangaciu[1] · Cristina Stangaciu[1] · Bianca Gusita[1] · Daniel-Ioan Curiac[2]

Received: 9 August 2024 / Revised: 27 January 2025 / Accepted: 13 February 2025

Abstract
Wireless Sensor Networks gradually became a mature technology that enabled the implementation of a large number of critical applications. However, the integration of such networks in much more complex systems, such as IoT, still poses a major challenge. In this paper we present a viable solution to accomplish such a difficult task by using MQTT-SN to integrate critical real-time WSNs into an IoT platform. The core of our work is a platform independent real-time driver for MQTT-SN along with a communication architecture to enable the integration of low end devices into an MQTT network using MQTT-SN. We also describe a comprehensive practical demonstration using real hardware modules that validates our claimed solution.

Keywords Real-time communication · Wireless sensor networks · Internet of things · MQTT · MQTT-SN

✉ Valentin Stangaciu
valentin.stangaciu@cs.upt.ro

Cristina Stangaciu
cristina.stangaciu@cs.upt.ro

Bianca Gusita
bianca.gusita@cs.upt.ro

Daniel-Ioan Curiac
daniel.curiac@aut.upt.ro

[1] Department of Computer and Information Technology, Politehnica University Timisoara, 2, Piata Victoriei, 300006 Timisoara, Timis, Romania

[2] Department of Automation and Applied Informatics, Politehnica University Timisoara, 2, Piata Victoriei, 300006 Timisoara, Timis, Romania

Published online: 25 February 2025

1 Introduction

The Edge Layer of an Internet of Things (IoT) platform is represented by many small devices low on computational, power and communication resources that handle the basic sensing and control tasks [1, 2] and are monitored and managed by the Fog and Cloud Layers [3, 4]. Such devices have evolved in terms of both numbers and hardware resources [5] being able to perform more sophisticated tasks and use modern communication protocols such as MQTT or CoAp that rely mainly on the TCP/IP stack [6].

However, integrating already existing Wireless Sensor Networks (WSNs) into an IoT platform still poses a challenge mainly because of the lack of hardware resources of the nodes. Furthermore, these devices rely on battery power which limits their communication capabilities thus standard IoT protocols based on TCP/IP are not suitable which implies using other solutions such as MQTT-SN [7]. An even greater challenge is to integrate WSNs that have specific time constraints when implementing real-time applications [8].

MQTT (Message Queuing Telemetry Transport) is a communication protocol based on the publish/subscribe paradigm designed especially for IoT. The protocol is organized in a Client–Server manner where the client part is represented by the IoT nodes and the server part is represented by a so-called broker which governs the whole network [9].

The MQTT protocol is localized at the application layer according to the OSI reference model [10] having the TCP/IP protocol as a transport under-layer. A simplified architecture of the MQTT protocol is presented in Figure. 1

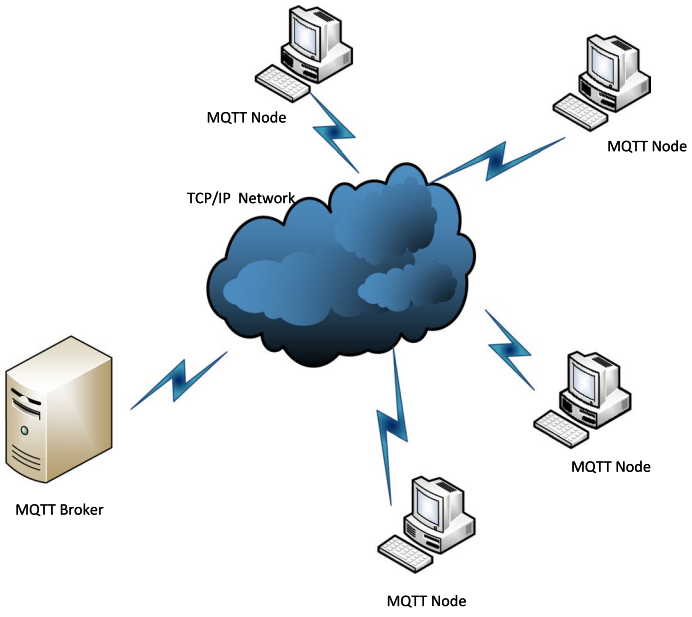


Fig. 1 MQTT general architecture

Being designed for IoT, the MQTT protocol is restrictive in requiring TCP/IP, which is unsuitable when applying this protocol for a sensor or wireless sensor network [11]. In such networks, the nodes have few resources in terms of memory or computational power and are usually battery-powered thus making the usage of TCP/IP almost impossible [12].

To solve some of these issues, a lighter version of the MQTT protocol was designed to allow MQTT usage on devices low on resource and battery-powered. The new version of the protocol designated as MQTT-SN (MQTT Sensor Networks) [13] allows the integration of sensor networks and wireless sensor networks into an MQTT network thus enabling access into the IoT paradigm.

The MQTT-SN may be transported by any communication protocol other than (but not excluding) TCP/IP such as 6LoWPAN [14], UDP [15] or ZigBee [16]. The messages of MQTT-SN are lighter and shorter than those of the classical MQTT but retain the same functionality. The most important differences between MQTT and MQTT-SN regarding lighter messages are the following [9, 17]:

- The connect message is split into three other messages where one is mandatory, the rest being optional
- The REGISTER-REGACK messages are introduced to replace topic names with numerical topic IDs, thus significantly reducing message size when addressing topics
- WILL messages and topics may be changed any time after a connection has been established
- PUBLISH-PUBACK will only use a topic ID or short topic name thus leaving more space for actual data
- SUBACK message returns the numerical topic ID of the topic name initially sent by the SUBSCRIBE message
- A DISCOVERY message is introduced for a gateway to announce its presence to the potential clients

Besides these changes, for a lightweight sensor network to be integrated into a MQTT network an additional element is needed. This additional element is represented by the MQTT-SN gateway which makes the interfacing and connection between the sensor network and the MQTT broker. The connection between the gateway and the sensor network is handled by the existing communication interface of the network which transports the MQTT-SN messages. The connection between the MQTT-SN gateway and the MQTT broker is implemented using the TCP/IP protocol stack.

Practically, the MQTT-SN protocol allows the integration of a classical wireless/wired sensor network with a modern MQTT network designed for the latest IoT paradigm. Such an integration takes advantage of the features of the wired/wireless sensor networks such as real-time operation, low-cost, and low power as well as the advantages of the modern IoT networks such as integrability, scalability, and security [18].

Another important aspect is that the nodes in such a hybrid network implementation are visible to the end user individually without any limitations.

Finally, with the help of MQTT-SN, the user may integrate an already existing WSN network with an IoT network.

The MQTT-SN solutions for the client nodes are not fully implemented even though they are available in many forms and many programming languages. Usually, for MQTT-SN, there are implementations available only for the main messages supported by the protocol but no actual working flow is defined. Many of the timings are also not implemented. Practically, without the main flow of the protocol the available implementations cannot be integrated into a working environment. The main flow must be implemented by the user in most of the available solutions.

In this paper, we propose an IoT architecture at Edge Layer that transparently integrates different wireless sensor networks using MQTT and MQTT-SN as application layer communication protocols. We also provide the design and implementation for the MQTT-SN communication protocol, including a platform-independent, real-time driver for MQTT-SN. With our solutionmode, different independent sensor networks that use different communication platforms will be able to interoperate and be integrated into a modern IoT network.

As we discuss in the Related Work section and from the best of our knowledge there is no actual solution for using MQTT-SN in critical, time-bounded real-time systems thus our main contribution is to provide such a functionality. Furthermore, we also present a novel IoT architecture which enables classical Real-Time Wireles Sensor Networks to be integrated into this paradigm.

The following is a summary of our main contributions:

- We provide a design for a hybrid IoT architecture at Edge Layer that transparently integrates different WSNs using MQTT and MQTT-SN communication protocols
- We present our design and implementation of the communication flow between clients and gateways.
- We offer a platform-independent, real-time driver for MQTT-SN
- We describe our implementation and demonstrate the effectiveness of our solution on real hardware platforms and present the experimental evaluation

The rest of the paper is organized as follows. A discussion regarding the related work is presented in Sect. 2 before describing our proposed network architecture in Sect. 3. The core of our work, the MQTT-SN real-time driver, is presented in Sect. 4 while a practical demonstration of the implementation on real hardware platforms is described in Sect. 5. The final Section concludes this paper.

2 Related Work

Regarding the implementation availability of MQTT, there are many available solutions for it. The implementations for MQTT are numerous supporting many programming languages and operating systems [19].

However, the situation is not the same for MQTT-SN. In this case, there are limited implementations for MQTT-SN. The gateway is usually found implemented

in many programming languages but the situation is not the same on the client side. A small step towards the integration of MQTT-SN was taken in [20] where authors explore this possibility but only using a simulation environment such as Cooja from ContikiOS. The same principle was adopted by the authors in [21] where MQTT-SN was evaluated along CoAP using the same simulation environment.

The MQTT-SN Protocol is not only integrated and evaluation in simulation environments, thus many implementations are currently available and documented properly. One such solution is available for the popular Arduino platform to be integrated in small sensing embedded projects [22]. The Zephyr Project also offers support in this direction providing their own solution for MQTT-SN for more complex embedded systems [23].

Another much more complex solution is available to be used in Linux based operating systems offering full suport along with a gateway to provide integration with an MQTT Broker [24]. Similar to this solution, the HiveMQ enterprise-ready MQTT platform [25] provides some limited support for MQTT-SN on both client and gateway side.

In many such solutions, the driver is usually implemented in a blocking manner which is not suitable for critical real-time systems. Furthermore, the transport protocol is usually hard-coded to UDP, thus limiting usage and portability to low-rate devices.

On the same principle as our proposed architecture, a hybrid solution of MQTT combined with ZigBee is already available and very popular in the home automation industry. Such a solution, represented by Zigbee2MQTT [26] acts similar to an MQTT-SN gateway and enables sensors and actuators implemented mainly using low rate and low-cost devices with ZigBee communication capabilities to be integrated into home automation solutions [27] where both power efficiency and network security are necessary [28, 29]. The drawback in this situation is that only supported Zigbee2MQTT devices may be integrated into the network. A list of such devices is provided, but without offering a universal solution, thus the addition of a new device implies a long process of integration into the system which is done by the developer team.

The relatively recent endeavours of further development of the MQTT-SN protocol are also tackling IoT-specific communication security problems. Security within MQTT-SN is recognized as an open issue [30, 31], prompting ongoing research aimed at enhancing its security frameworks without compromising its efficiency and low resource requirements. Currently, MQTT-SN is actively supported across various sectors such as smart city [12] or agriculture [32], reflecting its utility and confidence in its evolving security measures.

Different security solutions were proposed for enhancing MQTT-SN: like an intrusion detection system [30], several security mechanisms for authentication, access control, end-to-end security [33] or message encryption [34].

Other MQTT-SN extensions aim at improving the quality of communication in terms of latency: e.g. by controlling the Quality of Service in response to the status changes in the underlying network [15] or by providing real-time communication services, extending the message model by adding real-time attributes as priority and deadline to certain messages [35].

3 Hybrid IoT Real-Time Network

The main outcome of our work is a hybrid network of heterogeneous devices interconnected in order to achieve a single Internet of Things platform. We have interconnected four different types of networks as depicted in Fig. 2 using MQTT and MQTT-SN as application layer protocols.

The hybrid network is mainly based on MQTT under the coordination of an MQTT Broker. The classic MQTT network provides the base for the whole architecture being compatible with any device that can connect to an MQTT network using TCP/IP as a transport protocol. However, such a network is not suitable for low-rate devices, thus the TCP/IP protocol cannot be adapted to run on such devices.

In order to add the low-rate devices to the existing MQTT network we made use of the MQTT-SN protocol and adapted it accordingly. The interconnection was done by adding an MQTT-SN gateway by adapting the solution from Eclipse Paho [36, 37]. We modified the MQTT-SN gateway from Eclipse Paho to simultaneously coordinate multiple and different types of sensor networks such as:

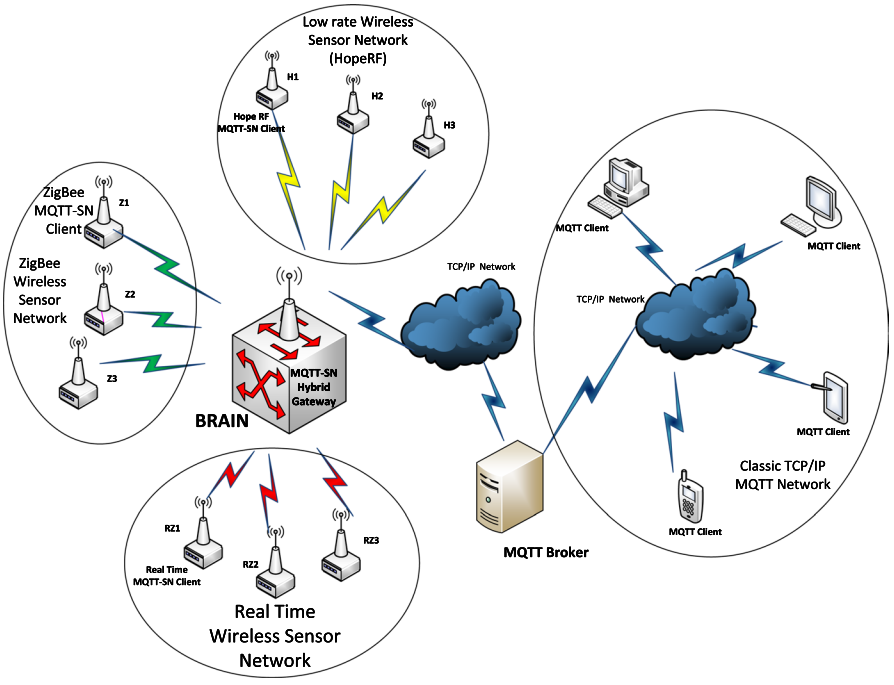


Fig. 2 Hybrid MQTT communication architecture

1. HopeRF Network - a low-rate, non-real-time wireless sensor network built using low-rate, low-power hardware with low computational and memory resources. Thus, such a network is usually very hard to integrate into an existing IoT network.
2. ZigBee Wireless Sensor Network - a soft real-time wireless sensor network using wireless modules along with the ZigBee [38] stack as communication interface. We consider this network to function in a soft-real time manner whe
3. Real-Time Wireless Sensor Networks - a hard real-time wireless sensor network, using also the ZigBee stack as communication interface on networks nodes having a hard real-time software component.

The real-time aspects of these networks is considerred at a node level where the communication driver along with the other tasks may or may not function in a real-time manner with strict time constraints. The firmware of the node of the HopeRF network (1) is written in a non-real-time manner with blocking and non-deterministic code. The system is not capable of real-time support from a hardware or software point of view.

The ZigBee Wireless Sensor Network (2) is capable of real-time constrants but many software modules are not designed in a deterministic manner which limits their real-time capability. In this case we consider that only soft real-time is supported where deadline misses of tasks in the node's firmware are permitted thus affecting only the performance of the system.

The nodes of the the Real-Time Wireless Sensor Network (3) were designed to fully support hard real-time constraints at a task level. The firmware is fully predictable without using any blocking statements having a Hard Real-Time Task Execution environment based on FreeRTOS [39] which guaranteed no deadline misses at a node level.

The three wireless sensor networks enumerated above were integrated with the help of our real-time, platform-independent driver for MQTT-SN based on the implementation provided by Paho Eclipse [22]. This part of our work is presented in section 4 of this paper.

4 MQTT-SN Real-Time Platform Independent Driver

This section aims at presenting the most important part of our work: our real-time solution of a real-time software module for MQTT-SN for the client devices, and the sensor nodes.

Our client implementation is based on the source code provided by Eclipse Paho [22] which offers only a base code for the message definitions and data encapsulations encoding and decoding. The documentation limits to the usage over network sockets in Linux-based operating systems [40].

Our implementation provides an MQTT-SN task to be executed in a real-time environment that can achieve strict time constraints. The task implementation is non-blocking with a predictable flow. Our solution may also be executed in a non-real-time environment.

The main flow of the driver is concentrated on the three main states of the MQTT-SN protocol that we defined in our solution:

- State **MQTT-SN DISCONNECTED** – the client is disconnected from the broker and no data exchange is possible except for the packets that may be received in a connection-less state such as ADVERTISE, SEARCHGW,... etc.
- State **MQTT-SN CONNECTING** – the client is in the connecting process. The MQTT CONNECT packets were sent by the client and it waits for a CONNACK confirmation from the broker.
- State **MQTT-SN CONNECTED** – the client is connected with the broker and a full message exchange is permitted from this point on.

The main flow of the protocol implementation is based on the state transitions presented in Fig. 3. The main state transitions are determined either by message exchange or by some strictly defined protocol timings.

The protocol timings are ensured using real-time software timers [41, 42]. The software timer's task needs to be executed by a real-time operating system with strict time parameters in order to provide accurate timing. However, in the case where such a real-time environment is not available, the software timer's main task may be executed periodically by using an interrupt servicing routine.

The software timers are used to implement three different timings required by the protocol:

- **Timeout timer** – this timer implements the timeout which is defined here as the maximum period of time between a request and a response. If this timer expires the timeout procedure is initiated.
- **Back-off timer** – the timer implements the back-off time, which is a period of time in which the client must not initiate any transmissions. Any transmission during the back-off time is rejected by the broker and eventually, the connection gets terminated.

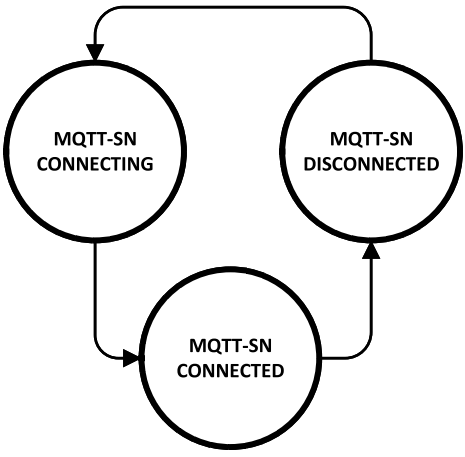


Fig. 3 MQTT-SN general state transitions

- **Ping timer** – this timer implements the period of time between the exchange of ping messages and has the role of keeping the session alive

The actual values of the timings implemented by the three timers are defined in accordance with the broker configuration.

The transitions defined in Fig. 3 that are determined either by the timings defined above or by event occuring upon MQTT-SN package exchange. A timeout event is generated when a response to a request is not received before the timeout timer expires. In such a situation the transaction is resumed but not before checking if a maximum number of consecutive timeouts were generated. Such a situation may suggest that the connection between the client and the gateway/broker is lost thus implying a transition to MQTT-SN DISCONNECTRED state. An internal variable is used to keep the consecutive number of timeouts. This variable is reset at each received packet. The transition to MQTT-SN CONNECTING is determined immediately after the back-off timer expires insuring the during this period no package exchange is permitted. In this state, the MQTT-SN client initiates the procedure responsible for establishing the connection with the MQTT broker through the MQTT-SN gateway. A transition to the MQTT-SN CONNECTED state will thus result after the connection was succesfully established.

Another important aspect is that when a timeout occurs, the current transaction is not immediately resumed. Before retrying a failed transaction, the client must first wait a *back-off time* during which it must be silent. Such behaviour is also defined by the broker and is mandatory.

The main flow of the protocol is not very complex but it needs to take some considerations into account:

- The timings and the protocol message exchange needs to be strictly fulfilled
- The time period of the protocol's task execution needs to be in accordance with the communication interface (at least two times faster than the maximum period of time between packet reception from the lower levels)

The main protocol flow will be presented in two phases: the *connecting* and *connected* phases. The connecting phase of our MQTT-SN implementation is designed according to Fig. 4

The flow begins with a back-off waiting time in order to ensure the requested silence in case of a possible previous error. After the back-off waiting state the flow goes into state MQTT-SN DISCONNECTED. In this state, the client must send an MQTT Connect request to the broker to initiate a connection. Successful transmission of a connect request will change the flow into state MQTT-SN CONNECTING. However, if the transmission is not successful, the flow will be restarted with the back-off waiting procedure.

In the MQTT-SN CONNECTING state, the client must only wait for a response from the broker containing a CONNACK message within the defined timeout period. If the timeout limit is reached then the MQTT flow will be reset with the back-off wait period. If a CONNACK message is received from the broker the client MQTT flow goes into MQTT-SN CONNECTED state. The client

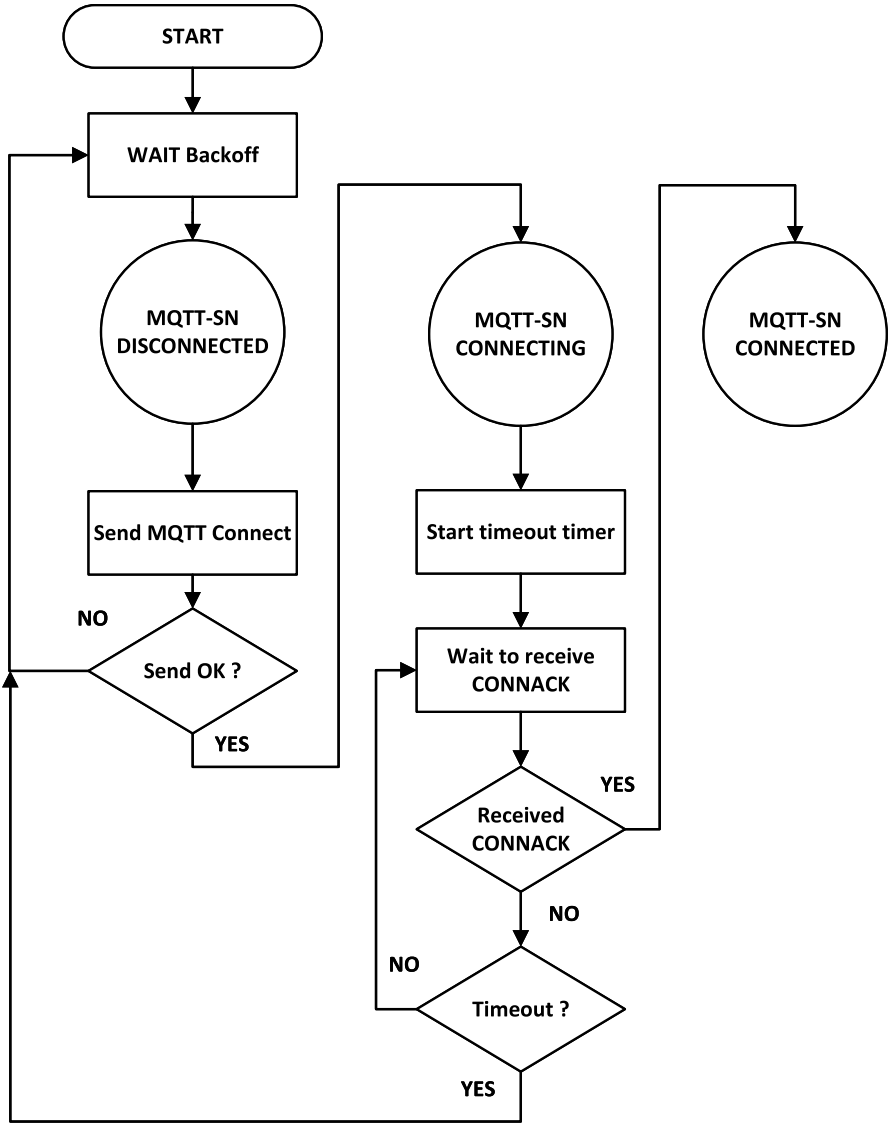


Fig. 4 MQTT-SN task main flow connecting phase

will remain in this state until either a DISCONNECT message is received from the gateway/broker, or the client wants to disconnect from the gateway/broker, or in case the maximum number of consecutive timeouts is reached.

In the MQTT-SN CONNECTED state, the flow is responsible for three operations: keeping the connected session alive by sending periodic ping requests (PINGREQ), processing incoming MQTT-SN messages from the broker, and

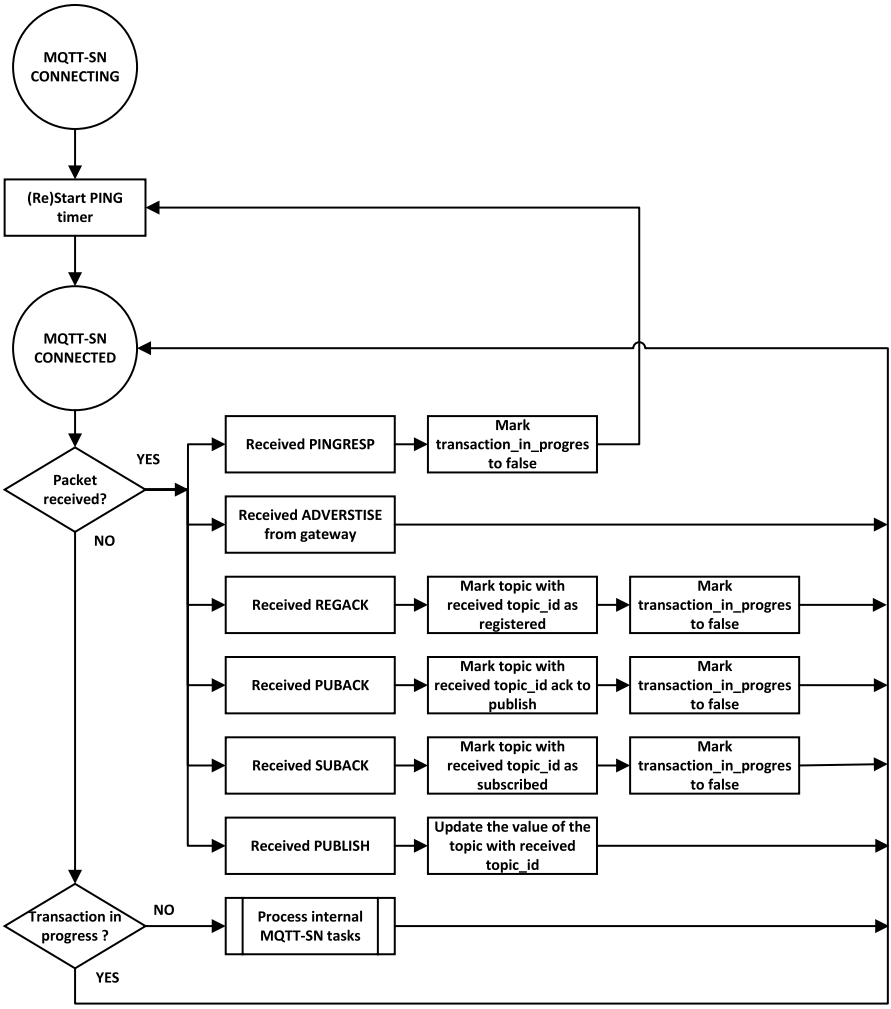


Fig. 5 MQTT-SN task main flow connected phase

processing local events for sending MQTT-SN messages to the broker. The flow is designed as presented in Fig. 5.

Immediately before commuting into the MQTT-SN CONNECTED state, the ping timer is started. This timer dictates when a ping request (PINGREQ) needs to be sent by the client to the gateway/broker. This ensures that the session is being kept alive. If the ping procedure is not implemented, the client is automatically disconnected by the broker, thus this procedure is vital for communication.

Our version of the MQTT-SN Client implementation manages the MQTT-SN topics by dividing them into local and remote topics. The local topics are the topics that are exported to the broker by issuing a SUBSCRIBE message followed by a PUBLISH message each time the payload changes. The remote topics are the topics that are

imported from the broker by issuing a REGISTER message and updating their value each time a PUBLISH message arrives from the broker. The data structure of a topic (remote or local) is described in Listing 1.

Code listing 1: MQTT-SN Topic data structure

```
typedef struct
{
  uint16_t topic_id;
  uint16_t register_packet_id;
  char *topic_name;
  uint8_t *payload;
  uint8_t payload_length;
  uint8_t flags_qos;
}MQTT_SN_TOPIC;
```

The structure members have the following meanings:

- **topic_id** – the numerical ID assigned to a topic, identified initially by its name, either in the REGISTER or SUBSCRIBE procedure.
- **register_packet_id** – the numerical ID of the packet that initiated the REGISTER or SUBSCRIBE procedure. This ID will later be used to confirm the procedure by REGACK or SUBACK packages.
- **payload** – the actual data payload transported usually by a PUBLISH message
- **payload_length** – the length of the payload in bytes
- **flag_qos** – a bitfield having the following structure:

The flags described in Table 1 have the following meaning:

- **valid** – states that the current record of the *MQTT_SN_TOPIC* structure is valid and that all the fields are correctly assigned.
- **registered** – states that the current topic was successfully registered or subscribed.
- **ackNeeded** – states that the current topic is the REGISTER or SUBSCRIBE process and a corresponding REGACK or SUBACK is needed

Table 1 Bitfield description of flag_qos

bit number	7	6	5	4	3 - 0
description	valid	registered	ackNeeded	newPayload	QOS

- **newPayload** – states that a new data payload was received for the current topic (which was successfully registered or subscribed)
- **QOS** – the QOS level for the current topic

The values of the flags in the bit-field described in Table 1 are limited to a certain set as described in Table 2.

Being designed for a real-time system, the local and remote topics are stored in statically allocated arrays and are accessed by the other tasks and application layers only through dedicated APIs.

The next part of the connected phase of the flow is to process any received MQTT-SN message from the gateway/broker. At each received message, the timeout timer is being reset. The implementation currently handles the following received MQTT-SN messages:

- **PINGRESP** – ping response - this packet states that a previously sent PINGREQ was acknowledged by the broker
- **ADVERTISE** – this packet is used by the gateway to announce its presence. Currently, the gateway is hard-coded thus this message is decoded but ignored
- **REGACK** – this message is received for a previously transmitted REGISTER request. In this case, the flow will mark the local topic identified by the received topic_id as registered. From this point on, the marked topic is registered by the broker and its value may be updated at any time by using a PUBLISH message
- **PUBACK** – this message is received as a response for previously sent PUBLISH request to update the value of a local topic. In this case, the local topic's new payload is marked as acknowledged
- **SUBACK** – this message is received as a response to a previously sent SUBSCRIBE request to subscribe to a remote topic. This message practically confirms the subscription to a remote topic and from this point on, the value for this topic will receive updates from the broker.
- **PUBLISH** – This message is sent by the broker for a subscribed remote topic to signal a value update. The flow will update the locally stored value of the remote topic with the value received with this message.

In the case of the reception of messages such as REGACK, PUBACK and SUBACK this marks the end of the transaction: REGISTER-REGACK, PUBLISH-PUBACK and SUBSCRIBE-SUBACK. This state is marked internally by using the variable *transaction_in_progress*. In the case of the reception of REGACK, PUBACK and SUBACK, the *transaction_in_progress* variable is set to false.

The internal tasks to be executed here, depicted in Fig. 6, are the following (in this order):

1. Find the first remote topic that has a pending PUBACK to be sent for a previously received PUBLISH message.
2. Find the first remote topic that needs to be subscribed by sending a SUBSCRIBE message.

Table 2 Flag value meaning

Valid	Registered	AckNeeded	NewPayload	Meaning
0	–	–	–	Current topic record is not valid and the driver will consider its field
1	0	–	–	Current topic is valid but not registered or subscribed the driver will send a REGISTER request for a local topic or a SUBSCRIBE request of a remote topic
1	1	0	–	The driver issued a REGISTER / SUBSCRIBE request and is currently waiting for a REGACK or SUBACK response from the gateway
1	1	1	–	The current topic (local or remote) is ready to be used for payload transfer using PUBLISH-PUBACK procedure
1	1	1	1	A new payload is available for the current topic either to be published (local topic) or was received via PUBACK (remote topic)

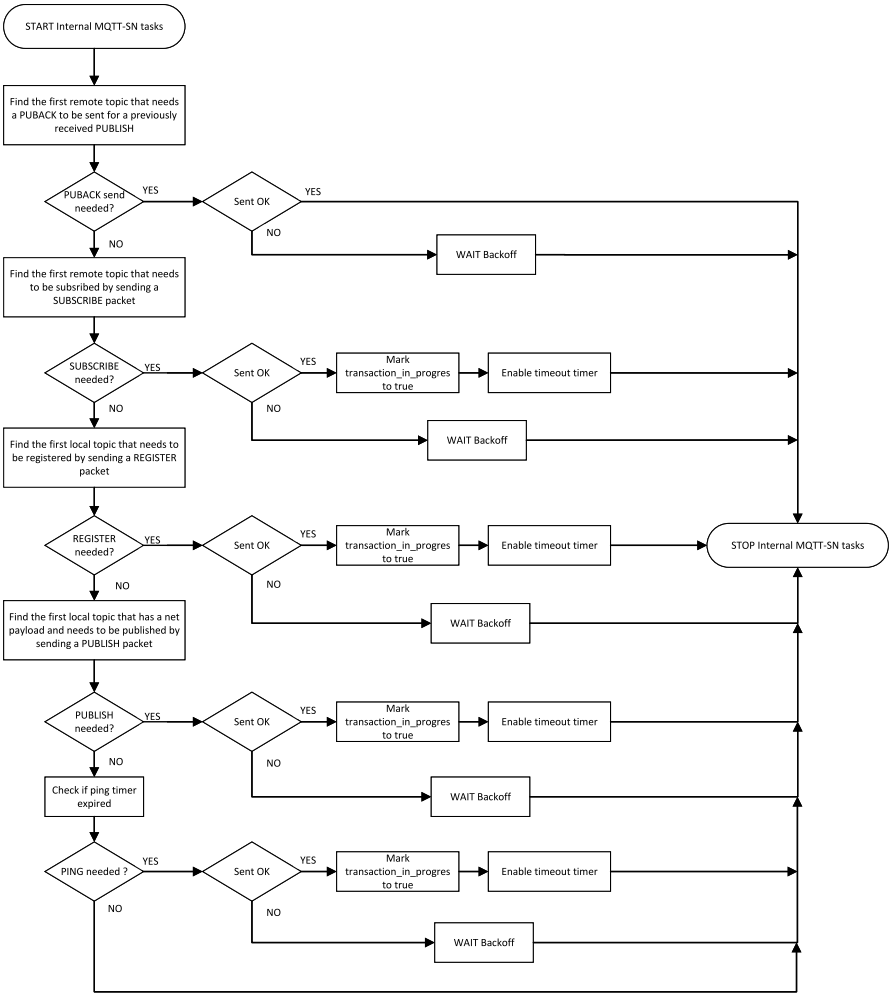
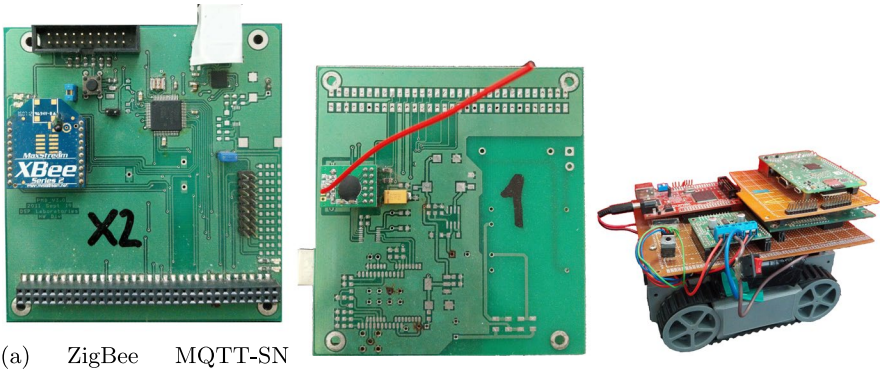


Fig. 6 MQTT-SN internal tasks flow

3. Find the first local topic that has not been registered in order to obtain a numerical topic ID and issue a REGISTER request if needed
4. Find the first local topic that has a new payload and the value has to be published by sending a PUBLISH message
5. Check if the ping timer has expired and a PINGREQ message needs to be sent

Each of these operations is executed only once within one execution of the main task. For each execution, the variable *transaction_in_progress* is marked as true and the timeout timer is restarted. Finally, the flow returns to the beginning of the MQTT-SN CONNECTED state.



(a) ZigBee MQTT-SN Wireless Sensor Network Node (b) HopeRF Low-rate Wireless Sensor Network Node (c) Real-Time Wireless Sensor Network Node

Fig. 7 Hybrid MQTT-SN nodes

Table 3 Low-Rate WSN test parameters

Parameter	Value	Parameter	Value
CPU Frequency	92.16 Mhz	Maximum RF packet Length	64 bytes
Back-off time	3000 ms	Transaction timeout	2 s
Keep alive transmit period	5 s	Keep alive timeout	100 s
Maximum consecutive timeouts	3	Data payload update period	1 s

5 Implementation and Experimental Results

We implemented our real-time MQTT-SN driver on various real hardware-software platforms. We built the architecture depicted in Fig. 2 and described in Sect. 3.

The Low-Rate Wireless Sensor Network is implemented using devices low on memory, power and computational resources. A HopeRF RFM12B Transceiver represents the wireless communication interface for these devices citehoperfrfm12b connected via SPI to the board's processor, an ARM7TDMI-S [43] microcontroller, an NXP LPC2103 chip [44]. Our custom implementation of such modules is presented in Fig. 7b. The software component is represented by a trivial non-real-time implementation, thus other solutions (i.e. a real-time operating system) are not suitable for such hardware. This limits the implementation of using a classical main loop using blocking operations along with the processor's interrupt system. The main characteristics of the test environment for this type of node are presented in Table 3.

The ZigBee Wireless Sensor Network is built using other custom-made modules (Fig. 7a). We implemented these modules using a similar ARM7TDMI-S processor, namely an NXP LPC2148 [45], which drives a Digi XBee Series 2

RF module [46] via a UART interface. The software component is based on our hybrid real-time solution [47] where we took advantage of both the FreeRTOS operating system [48] and the stability of jitterless execution provided by the HARETICK (Hard Real-Time Compact Kernel) [49]. The two operating systems manage a set of tasks that provide a predictable real-time by using specific time constraints as described in Table 4.

The XBee driver implementation [50] ensures a firm real-time communication with the ZigBee network. The driver is represented by 5 tasks XBEE1-4 and XAPP while the MQTT-SN driver is implemented by the MQTT task as specified in Table 4. The test characteristics for the ZigBee WSN nodes are described in Table 5.

Furthermore, we integrated an already existing real-time network into the hybrid solution we presented in this paper to adapt our environment to the IoT paradigm. Such a network implements a real-time robotic collaborative environment [51] meant for time-critical applications. The nodes of the networks are represented by the entity designated as WIT (Wireless Intelligent Terminal) which is built as a multi-board and multi-processor architecture with a hard real-time communication protocol stack (PARSECS_RT Stack) ensuring the intra-board communication [52].

The WIT, with a prototype presented in Fig. 7c, is a multifunctional node driven by a motherboard implemented with the help of a Raspberry PI3 module [53]. The communication is also implemented by an XBee Series 2 module attached to a dedicated communication board with a similar architecture as the previously described. However, the functionality of the whole node is dictated by the motherboard which implements the whole logical flow of the WIT, and thus hosts our MQTT-SN driver as one of the running tasks. The main operating system is ArchLinux [54] along with the Litmus^{RT} add-on meant to provide higher real-time flexibility for the Linux kernel. The real-time tasks that are running in the system using the Litmus^{RT} scheduling add-on along with their timing are described in Table 6. To ensure a strict hard-real time behavior, each task was assigned to a single different reservation pool [55].

The test characteristics for the Real-Time ZigBee WSN nodes are described in Table 7.

The values chosen in Tables 3,5 and7 are mostly dependent on the communication interface. The *Maximum RF Packet Length* parameters states the maximum value of a data packet supported by the communication interfaces which is 64 for the HopeRF modue and 96 for the ZigBee Series 2 modules. Regarding the HopeRF network, the values in Table 3 are a little more *aggressive* than the values for the other two networks mainly because the instability of the HopeRF network. This characteristic of the low-rate HopeRF network implies that the nodes are prone to more packet loss than the ZigBee modules which may lead to more disconnected events. In order to keep the nodes connected as much as possible, *the keep alive transmit period* was decreased keeping the nodes more active on the network. Furthermore, in order to provide a faster reconnection in case of a disconnect situation the maximum consecutive timeouts value is much more strict than the value used for the other network types. One can easily observe that these values are more *permissive* for the ZigBee networks because of the increased stability of such networks. However, the *data payload update period* was arbitrary taken.

Table 4 ZigBee MQTT-SN WSN node running tasks

Task name	Task short name	Task description	Task period	OS Context
TIMER_SOFTWARE	TS	The software timer engine that allows the MQTT-SN timings to be implemented	1 ms	FreeRTOS
Task_XBeeLayer1	XBEE1	Layer 1 of the XBee Series 2 real-time driver	1200 μs	HARETICK
Task_XBeeLayer2	XBEE2	Layer 2 of the XBee Series 2 real-time driver	6 ms	FreeRTOS
Task_XBeeLayer3	XBEE3	Layer 3 of the XBee Series 2 real-time driver	20 ms	FreeRTOS
Task_XBeeLayer4	XBEE4	Layer 4 of the XBee Series 2 real-time driver	50 ms	FreeRTOS
Task_XBeeAppTask	XAPP	Application layer of the XBee Series 2 real-time driver	50 ms	FreeRTOS
Task_MQTT	MQTT	Executes the real-time task responsible for implementing the MQTT-SN driver	50 ms	FreeRTOS

Table 5 ZigBee MQTT-SN WSN node test parameters

Parameter	Value	Parameter	Value
CPU frequency	58.9824 MHz	Maximum RF Packet Length	96 bytes
Back-off time	3000 ms	Transaction timeout	2 s
Keep alive transmit period	5 s	Keep alive timeout	50 s
Maximum consecutive timeouts	3	Data payload update period	500 ms

The rest of the network is a classical MQTT over TCP/IP implemented using commonly used smartphones and tablets running various versions of Android as the operating system.

The gateways and the MQTT broker are hosted on an Intel NUC [56]. The MQTT-SN gateway responsible for the HopeRF Low Rate Wireless Sensor Network interacts with the gateway using a dedicated adaptor having access to a HopeRF Transceiver as depicted in Fig. 8a. On the other hand, mainly for simplicity reasons, the two Zigbee networks are both handled by a single ZigBee coordinator connected to the gateway through an XBee Series 2 serial adapter as pictured in Fig. 8b.

The gateways and the MQTT broker are hosted on an Intel NUC [56]. The MQTT-SN gateway responsible for the HopeRF Low Rate Wireless Sensor Network interacts with the gateway using a dedicated adaptor having access to a HopeRF Transceiver as depicted in Fig. 8a. On the other hand, mainly for simplicity reasons, the two Zigbee networks are both handled by a single ZigBee coordinator connected to the gateway through an XBee Series 2 serial adapter as pictured in Fig. 8b.

The HopeRF Gateway Adapter is built around an NXP LPC2148 microcontroller [45] and interfaces with the gateway using the serial interface. The main role of the firmware running on the microcontroller is to manage the radio module and to provide a communication interface for data transfer with the gateway with minimal encoding as presented in Fig. 9.

The HopeRF Serial Interface Packet offers a minimum and necessary data encoding in order to provide simple data transfer for the gateway. When such a packet is received by the HopeRF Gateway adapter, the extracted data is sent using the HopeRF Radio module and vice-versa. The fields of the HopeRF Serial Interface Packet are described in Table 8.

Our solution integrates all of these nodes into a heterogeneous IoT network presenting it as a homogeneous network over MQTT.

In our test setup, in order to prove the functionality and effectiveness of our design each node publishes two topics:

- **upt/name/< name>** where < name> represents a string identifying the device
- **upt/< name>/data** where, under a topic containing the device's *name* each node exports a number that incremented internally, identified as *data*

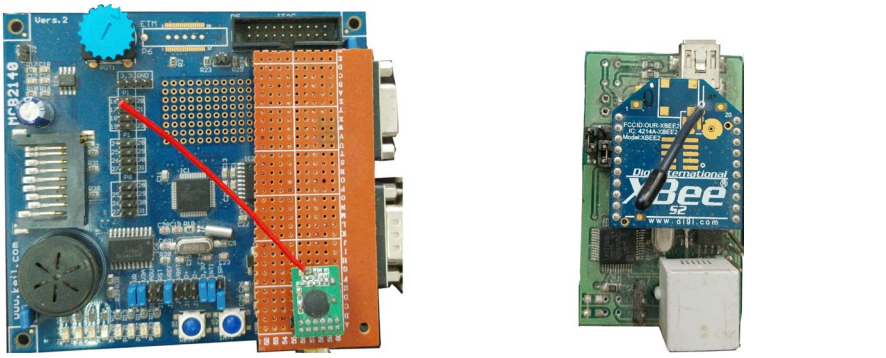
A much clearer view may be found in Table 9 where the whole test network is described more accurately.

Table 6 ZigBee real time MQTT-SN WSN node running tasks

Task name	Task short name	Task description	Task period
TIMER_SOFTWARE	TS	The software timer engine that allows the MQTT-SN timings to be implemented	1 ms
Task_PARSECS_LL	SPI_LL	The PARSECS_RT low level substack for intra-board SPI communication	600 us
Task_PARSECS_HL	SPI_HL	The PARSECS_RT high level substack for intra-board SPI communication	1 ms
Task_MQTT	MQTT	Executes the real-time task responsible for implementing the MQTT-SN driver	50 ms
Task_APP	TAPP	Executes the main demo application task	250 ms

Table 7 ZigBee Real-Time MQTT-SN WSN node test parameters

Parameter	Value	Parameter	Value
CPU frequency	1.2 GHz	Maximum RF packet length	96 bytes
Back-off time	5000 ms	Transaction timeout	5 s
Keep alive transmit period	30 s	Keep alive timeout	60 s
Maximum consecutive timeouts	10	Data payload update period	3000 ms



(a) HopeRF Gateway Adapter (b) XBee Series 2 Gateway Adapter

Fig. 8 Gateway adapters

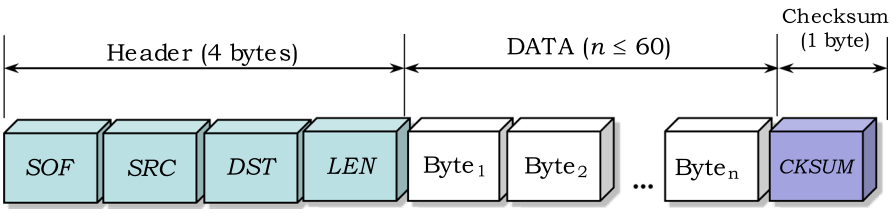


Fig. 9 HopeRF serial interface packet

The whole process was monitored either using the mosquitto_sub [57] and with a graphical MQTT visualization tool such as MQTT-Explorer [58]. As presented in Fig. 10, we can identify the topics exported by each node along with their associated values as described in Table 9.

We have concentrated on evaluating the whole network by measuring the packet loss in various scenarios with the parameters described in Tables 3, 5, 7. For each experiment, each node initiated 2500 PUBLISH-PUBACK transactions and the results are presented in Fig. 11. The main reason behind this decision is that we wanted to establish if and how different networks could influence each other.

Table 8 HopeRF serial packet field description

Field name	Field full name	Size (bytes)	Description
SOF	Start Of Frame	1	Marks the beginning of a frame the value is hard-coded as 0x7E
SRC	Source Address	1	The source address of the packet gateway has its address hard coded to 0x00
DST	Destination Address	1	The destination address of the node where the packet should be sent
LEN	Length	1	The length of the data field
DATA	Data field	LEN	The data payload
CKSUM	Checksum	1	The checksum of the data field

Table 9 Experimental network description

Network	Node short name	Node long name	Real-time	OS	Protocol	Published topics
Low Rate WSN	H1	hope-1	NO	NO OS	MQTT-SN	upt/name/hope-1 upt/hope-1/data
	H2	hope-2	NO	NO OS	MQTT-SN	upt/name/hope-2 upt/hope-2/data
ZigBee WSN	Z1	zigbee-freertos-1	FRT	FreeRTOS Haretick	MQTT-SN	upt/name/zigbee-freertos-1 upt/zigbee-freertos-1/data
	Z2	zigbee-freertos-2	FRT	FreeRTOS Haretick	MQTT-SN	upt/name/zigbee-freertos-2 upt/zigbee-freertos-2/data
Real-Time WSN	RZ1	wit-1	HRT	ArchLinux LitmusRT	MQTT-SN	upt/name/wit-1 upt/wit-1/data
Classic TCP/IP Network	mqtt-1	phone-1	NO	Android	MQTT	upt/name/phone-1 upt/phone-1/data
	mqtt-2	tablet-1	NO	Android	MQTT	upt/name/tablet-1 upt/tablet-1/data
	mqtt-3	tablet-2	NO	Android	MQTT	upt/name/tablet-2 upt/tablet-2/data

In order to establish a reference point, we conducted measurements to evaluate the packet loss of each device individually. As expected, the results in Fig. 11a emphasize the low rate and unreliability of the HopeRF network in contrast with a much more stable situation where ZigBee is used.

During the next step, we evaluated the packet loss at the network level for each type of network where all the devices of the same network were included in the experiment. Because the HopeRF network has an extremely poor MAC level with no mechanisms to prevent and treat collisions, when more than one device is used, the instability increases as shown in Fig. 11b. However, on the ZigBee side, even if the packet loss slightly increases, the obvious higher performance may be observed.

We evaluated the packet loss for the whole hybrid network where all the sensor networks were included in the experiment, thus all functioning in a hybrid manner together in parallel. As it can be observed in Fig. 11c few changes in packet loss occurred. A behaviour change may be noticed when merging all the networks into a hybrid communication platform but with little negative impact.

Even though these are only preliminary experimental results, they clearly demonstrate a small increase in packet loss when integrating all the networks. This result was indeed expected and the measurements prove that this increased packet loss is negligible and limited. Another important aspect may be observed in Fig. 11b and Fig. 11c where the two different networks communicating using the same ZigBee infrastructure are affected by each other. Although this is expected, in a much practical application such a situation should not occur mainly because the

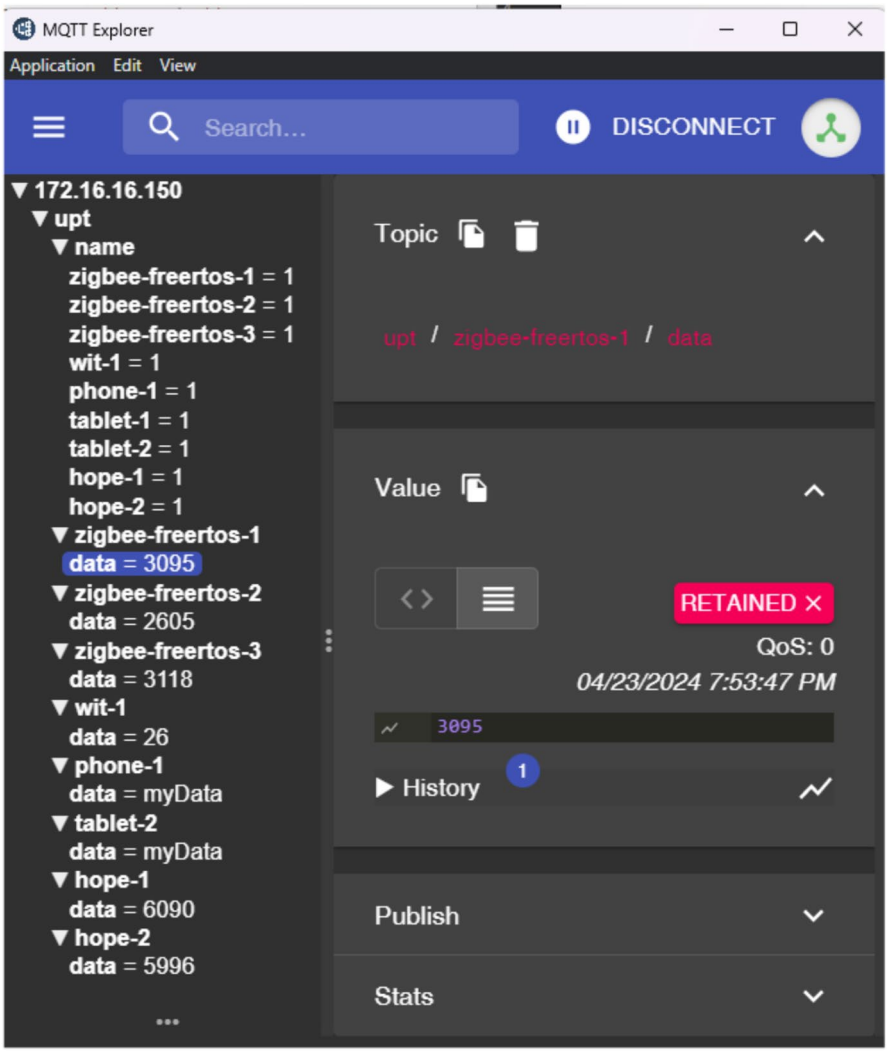
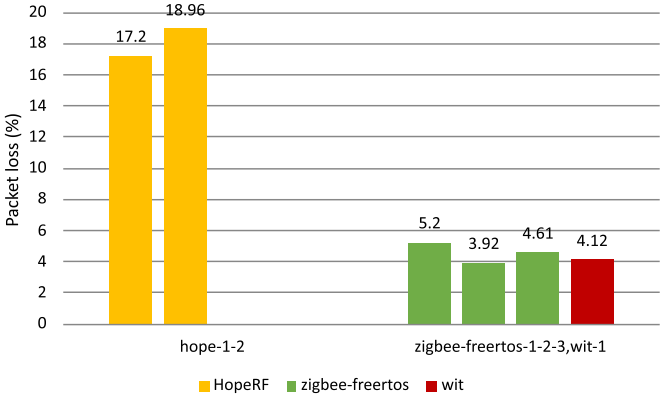


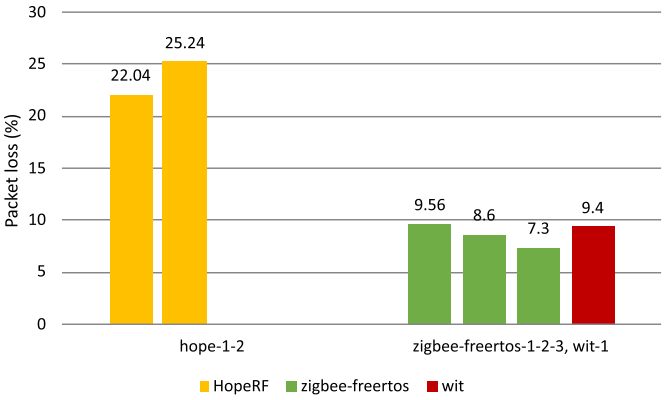
Fig. 10 Network visualization proof using MQTT-explorer

two networks could be completely separate having distinct networks coordinators communicating on different frequency channels.

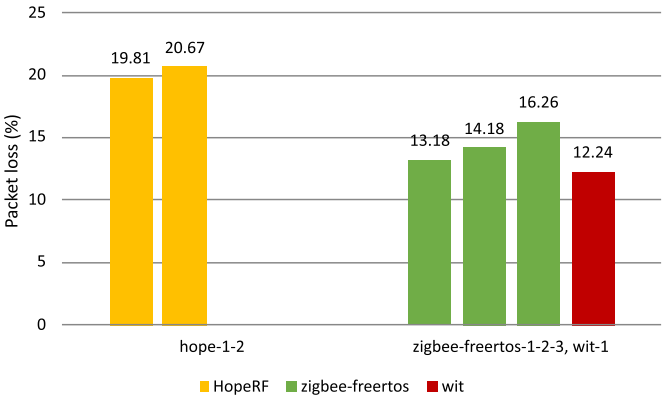
Also, it can be clearly seen in Fig. 11 that no packet loss increase can be observed in the HopeRF network. The reason behind this behaviour is that the HopeRF network communicate on a completely different frequency spectrum than the ZigBee networks. All of this preliminary concludes that the observed packet loss in the ZigBee networks are not necessarily caused by our hybrid network integration but because of the increased communication on the same frequency channels which in a real application may be easily avoided.



(a) At individual node level



(b) At sub-network level



(c) At full-network level

Fig. 11 Packet loss measurement

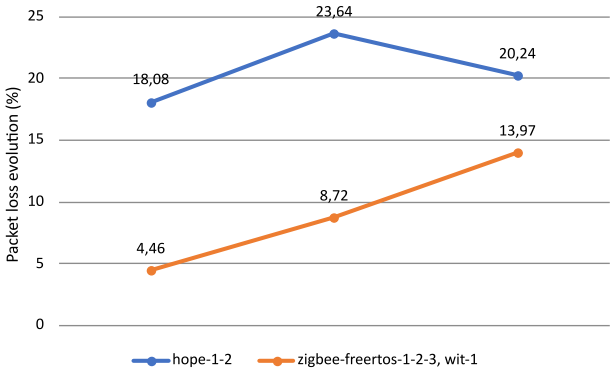


Fig. 12 Packet loss synthesis

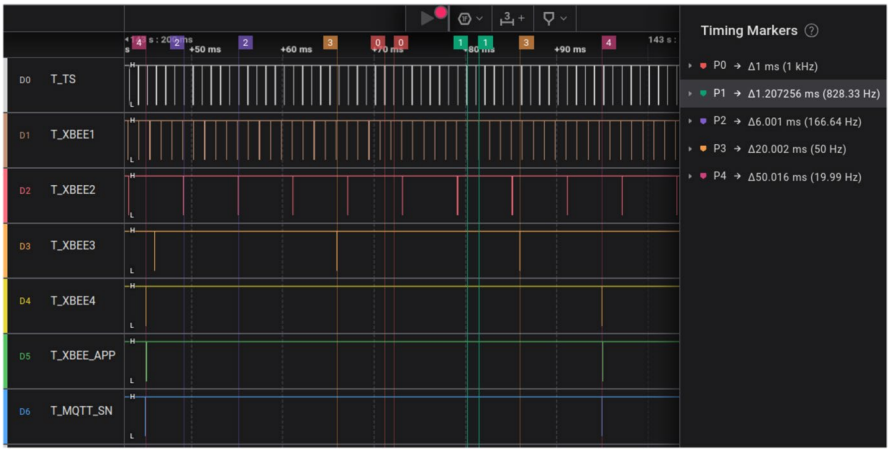


Fig. 13 Real-time task executing timings for ZigBee MQTT-SN WSN node

It is also very important to mention that the classical MQTT network cannot be affected in any way by our hybrid solution, thus MQTT is transported by a different TCP/IP network and is managed entirely by the MQTT broker which is technically unaware of the existence of the MQTT-SN network. Also, the MQTT network communicated on a completely different platform which is not affected by any communication from a WSN.

A packet loss synthesis as presented in Fig. 12 may be used to summarise the way packet loss was affected when applying our solution.

In order to further analyze our solution in terms of real-time requirements, we continue to present in Fig. 13 the task execution timings as captured using a Saleae Logic Pro16 Analyzer [59] for the ZigBee MQTT-SN WSN Node. In order to obtain these measurements a classical software method was used: a dedicated GPIO for each task is toggled to logic LOW at the beginning the the execution of the task and

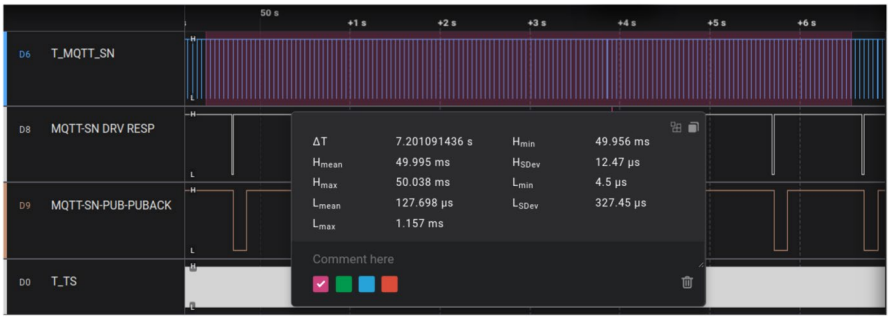


Fig. 14 MQTT-SN task execution time for ZigBee MQTT-SN WSN node

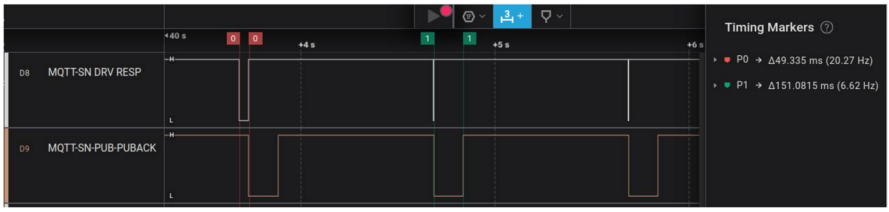


Fig. 15 MQTT-SN driver response time for ZigBee MQTT-SN WSN node

to logic HIGH at the end of execution. The timing markers on the right measure the execution period of the tasks: marker *P0* measures the period of the TIMER_SOFTWARE task (T_TS), marker *P1* the period of XBEE1 Task, *P2* the period of XBEE2 Task, *P3* the period of XBEE3 Task while *P4* measures the period of the MQTT-SN Task. As it can easily be observed the measured values demonstrate the theoretical data from Table 4.

An important parameter when dealing with real-time environments is the Worst Case Execution Time (WCET). This parameter is normally calculated using dedicated tools which are dependent on the CPU architecture. In Fig. 14 we determined the execution time experimentally and obtained its maximum measured value under the longest execution path of the driver. The obtained value may be found under L_{MAX} in Fig. 14.

Another crucial time parameter in this analysis is the response time of the driver. This parameter is experimentally determined in Fig. 15 at *MQTT-SN DRV RESP*. We considered the falling edge when the application updates the value of the topic *upt/zgbee-freertos-1/data* and the rising edge when the MQTT-SN driver sent the PUBLISH message to the radio interface. In the worst conditions, when the application task updates this value immediately after an execution of the MQTT-SN task this value is determined by the actual execution period of the MQTT-SN task. Such a condition is demonstrated by the *P0* timing marker in Fig. 15. Furthermore, as it can be observed the response time is not constant, being dependent on the conditions described before, but in a real-time environment the absolute maximum value is crucial.

The second waveform in Fig. 15 is represented by a PUBLISH-PUBACK transaction time and is designated as *MQTT-SN-PUB-PUBACK*. The falling edge represents the moment the PUBLISH message is sent to the radio interface and the rising edge determines the moment the PUBACK is received back by the MQTT-SN driver from the MQTT broker. Such a time parameter is highly dependent on the radio communication and it does not influence the timings of the MQTT-SN driver. The time period of this transaction is measured by *P1* timing markers.

A similar analysis is required for the Real-Time WIT Node as presented in Fig. 16 with similar execution parameters as for the ZigBee MQTT-SN WSN Node in order to have a proper reference point. The waveform presented here, designated as *T_MQTT_SN*, represents the execution of the MQTT-SN task on the Linux platform with the Litmus^{RT} extension as described earlier in this section. Having the same measuting methods as before, the longest execution time of the MQTT-SN task can be identified by the value L_{MAX}. This measured value of 302.5 μs for the WCET of the MQTT-SN task represents a much faster execution than in the previous case of 1.157 ms, mainly because this platform is a Raspberry PI3 with a much higher CPU frequency of 1.2 GHz.

Furthermore, the MQTT-SN Driver response time is described by the second waveform in Fig. 16 designated again as *MQTT-SN DRV RESP* and measured by the timing marker *P0*. We can observe the exact same behaviour and as in the previous case as expected. The full PUBLISH-PUBACK transaction time is represented here in the third waveform designated as *MQTT-SN-PUB-PUBACK* and measured by the *P1* timing markers. The notable time difference of aprox. 100 ms is easily explained by the fact that the data in this case has a longer transmission path thus the PARSECS_RT intra-board communication protocol transfers the MQTT-SN packet from the motherboard of the WIT to the communication module and vice-a-versa when the MQTT-SN response is received.

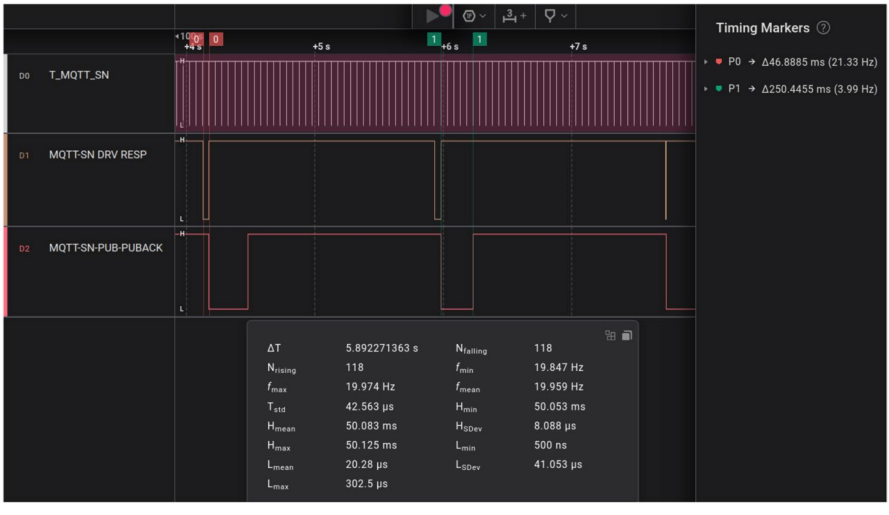


Fig. 16 MQTT-SN driver time analysis for real-time ZigBee MQTT-SN WSN WIT node

Table 10 Time measurement summary

MQTT-SN task Execution time parameters	ZigbBee MQTT-SN WSN Node	Real-time ZigBee MQTT-SN WIT node
Hardware platform	NXP LPC 2148	Raspberry PI3
CPU Frequency	58.9824 MHz	1.2 GHz
Execution Period	49.956 ms	50.083 ms
WCET	1.157 ms	302.5 μs
Response Time	49.335 ms	46.8885 ms
PUB-PUBACK Transaction Time	116.1081 ms	250.4455 ms

The time measurements regarding the MQTT-SN task presented in Figs. 13, 14, 15, 16 are being summarized in Table 10. The measured execution period values are consistent with the theoretical values as given in Tables 7, 5. Being mandatory in real-time systems, the predictability of the driver is demonstrated by the Response Time which must have a maximum value equal to the execution period as demonstrated by the measurements. Another essential parameter for real-time systems, the WCET is measured on the two different platforms which is clearly influenced by the performance of the CPUs executing the task.

6 Conclusion and Future Work

In this paper we propose a solution for integrating existing low-rate wireless sensor networks into an IoT network using MQTT-SN, an MQTT variant adapted for such networks. We concentrate our work in the real-time domain thus we designed a real-time, platform independent driver for MQTT-SN.

We managed to create a hybrid IoT network consisting of both MQTT capable devices and low-rate sensor networks running our MQTT-SN driver. We obtained a heterogeneous IoT network presented as a homogeneous MQTT network.

In terms of scalability, our solution practically relies on the scalability of the underlying transport protocols thus MQTT and MQTT-SN are proven not to affect this property.

To provide a much clearer concluding overview, in Table 11 we provide a comparison summary of our solution with other existing MQTT-SN implementations. As it can be observed our MQTT-SN Real Time driver not only that provides hard real-time support for critical application but also offers greater flexibility in terms of underlying transport protocols and implementation platforms.

We further continue our work and have already obtained preliminary promising results by adapting this proposed architecture for integration with the Conti Cooja Simulator in order to obtain an IoT network functioning with both simulated and real hardware devices.

Table 11 Solution comparison

Solution	Real-time aspects	Transport protocol	Hardware/software platform	Programming language
Arduino based MQTT-SN Client [22]	No real-time support implemented using busy-wait loops	UDP	Arduino based platform	C++
Eclipse Paho [37] MQTT-SN Client	Does not provide a full working implementation	UDP (for the exmple code only)	Platform independent	C
CommsChampion Ecosystem MQTT-SN Client [24]	Single-threaded non-blocking impementation	UDP	Linux based platform	C++ using STL
Zephyr MQTT-SN Client [23]	No real-time support	UDP, ZigBee, Bluetooth, UART	Zephyr OS	C/C++
HiveMQ Edge MQTT-SN Gateway [25]	No real-time support	UDP	HiveMQ Environment	C/C++
Our MQTT-SN driver	Hard real-time support	Any transport protocol	Platform and OS independent	C

Author Contributions Valentin Stangaciu conducted the main part of the research and written the most part of the article. Cristina Stangaciu conducted the literature study, was in charge of the real-time analysis and handled all the aspects regarding task scheduling in RTOS. Bianca Gusita conducted the main testing and validation of the project and verified the correct operation of the MQTT and MQTT-SN protocols. Daniel Ioan Curiac supervised the whole process related to the writing of the article and revised the manuscript.

Funding No funding was received in any way for conducting this study.

Data Availability Not applicable.

Materials Availability Not applicable.

Code Availability Not applicable.

Declarations

Conflict of interest The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Ethical Approval Not applicable.

Consent to Participate Not applicable.

Consent for Publication All authors have approved the submission of the manuscript.

References

1. Wang, Y., Wu, S., Lei, C., Jiao, J., Zhang, Q.: A review on wireless networked control system: the communication perspective. IEEE Internet Things J. **11**(5), 7499–7524 (2024). https://doi.org/10.1109/jiot.2023.3342032
2. Said, S., Hajlaoui, J.E., Omri, M.N.: A survey on the optimization of security components placement in internet of things. J. Netw. Syst. Manag. (2024). https://doi.org/10.1007/s10922-024-09852-6
3. Cruz, M.A.A., Rodrigues, J.J.P.C., Al-Muhtadi, J., Korotaev, V.V., Albuquerque, V.H.C.: A reference model for internet of things middleware. IEEE Internet Things J. **5**(2), 871–883 (2018). https://doi.org/10.1109/jiot.2018.2796561
4. Singh, P., Singh, R.: Energy-efficient delay-aware task offloading in fog-cloud computing system for iot sensor applications. J. Netw. Syst. Manag. (2021). https://doi.org/10.1007/s10922-021-09622-8
5. Portilla, J., Mujica, G., Lee, J.-S., Riesgo, T.: The extreme edge at the bottom of the internet of things: a review. IEEE Sens. J. **19**(9), 3179–3190 (2019). https://doi.org/10.1109/jsen.2019.2891911
6. Cruz, M.A.A., Rodrigues, J.J.P.C., Lorenz, P., Korotaev, V.V., Albuquerque, V.H.C.: In.iot-a new middleware for internet of things. IEEE Internet Things J. **8**(10), 7902–7911 (2021). https://doi.org/10.1109/jiot.2020.3041699

7. Tempel, S., Herdt, V., Drechsler, R.: Specification-based symbolic execution for stateful network protocol implementations in iot. IEEE Internet Things J. **10**(11), 9544–9555 (2023). https://doi.org/10.1109/jiot.2023.3236694
8. Behnke, I., Austad, H.: Real-time performance of industrial iot communication technologies: A review. IEEE Internet Things J. **11**(5), 7399–7410 (2024). https://doi.org/10.1109/jiot.2023.3332507
9. Standard, OASIS: Mqtt version 5.0. Retrieved June 22, 2020 (2019)
10. Zimmermann, H.: Osi reference model - the iso model of architecture for open systems interconnection. IEEE Trans. Commun. **28**(4), 425–432 (1980). https://doi.org/10.1109/TCOM.1980.1094702
11. Nast, M., Golatowski, F., Timmermann, D.: Design and performance evaluation of a standalone mqtt for sensor networks (mqtt-sn) broker. In: 2023 IEEE 19th international conference on factory communication systems (WFCS) (2023). https://doi.org/10.1109/wfcs57264.2023.10144241
12. Rasyid, M.U.H.A., Astika, F., Fikri, F.: Implementation mqtt-sn protocol on smart city application based wireless sensor network, pp. 7–12. IEEE, Yogyakarta, Indonesia (2019). https://doi.org/10.1109/ICSITech46713.2019.8987546
13. Quincozes, V.E., Quincozes, S.E., Kazienko, J.F., Gama, S., Cheikhrouhou, O., Koubaa, A.: A survey on iot application layer protocols, security challenges, and the role of explainable ai in iot (xaiot). Int. J. Inf. Secur. **23**(3), 1975–2002 (2024). https://doi.org/10.1007/s10207-024-00828-w
14. Herrero, R.: Mqtt-sn, coap, and RTP in wireless iot real-time communications. Multim. Syst. **26**(6), 643–654 (2020). https://doi.org/10.1007/S00530-020-00674-5
15. Palmese, F., Redondi, A.E.C., Cesana, M.: Adaptive quality of service control for mqtt-sn. Sensors (2022). https://doi.org/10.3390/s22228852
16. Guha Roy, D., Mahato, B., De, D., Buyya, R.: Application-aware end-to-end delay and message loss estimation in internet of things (iot) - mqtt-sn protocols. Future Gener. Comput. Syst. **89**, 300–316 (2018). https://doi.org/10.1016/j.future.2018.06.040
17. Stanford-Clark, A., Truong, H.: Mqtt-sn version 1.2. Protocol Specification, Oasis (2013)
18. Roldán-Gómez, J., Carrillo-Mondéjar, J., Castelo Gómez, J.M., Ruiz-Villafranca, S.: Security analysis of the mqtt-sn protocol for the internet of things. Appl. Sci. **12**(21), 10991 (2022). https://doi.org/10.3390/app122110991
19. Mishra, B., Kertesz, A.: The use of mqtt in m2m and iot systems: a survey. IEEE Access **8**, 201071–201086 (2020). https://doi.org/10.1109/access.2020.3035849
20. Silva, E.F., Dembogurski, B.J., Vieira, A.B., Ferreira, F.H.C.: Ieee p21451-1-7: Providing more efficient network services over mqtt-sn. In: 2019 IEEE sensors applications symposium (SAS) (2019). https://doi.org/10.1109/sas.2019.8706107
21. Martí, M., Garcia-Rubio, C., Campo, C.: Performance evaluation of coap and mqtt_sn in an iot environment. Proceedings (2019). https://doi.org/10.3390/proceedings2019031049
22. Nikol, G.: Arduino based MQTT-SN Client. GitHub (2018)
23. Project, Z.: Zephyr Project MQTT-SN Client. GitHub (2024)
24. Robenko, A.: CommsChampion Ecosystem MQTT-SN Client. GitHub (2024)
25. HiveMQ: HiveMQ Edge MQTT-SN Gateway. https://docs.hivemq.com/hivemq-edge/mqtt-sn-gateway.html. Accessed: 2024-04-23
26. Koenkk: Zigbee2MQTT. GitHub (2024)
27. Hussein, N., Nhlabatsi, A.: Living in the dark: Mqtt-based exploitation of iot security vulnerabilities in zigbee networks for smart lighting control. IoT **3**(4), 450–472 (2022). https://doi.org/10.3390/iot3040024
28. Ansari, A.M., Nazir, M., Mustafa, K.: Smart homes app vulnerabilities, threats, and solutions: a systematic literature review. J. Netw. Syst. Manag. (2024). https://doi.org/10.1007/s10922-024-09803-1
29. Zaheer, H., Shoaib, M., Iqbal, F., Arshad, S., Altaf, A., Villena, E.G., Torre Diez, I., Ashraf, I.: An energy-efficient technique to secure internet of things devices using blockchain. J. Netw. Syst. Manag. (2024). https://doi.org/10.1007/s10922-024-09870-4
30. Roldán-Gómez, J., Carrillo-Mondéjar, J., Castelo Gómez, J.M., Ruiz-Villafranca, S.: Security analysis of the mqtt-sn protocol for the internet of things. Appl. Sci. **12**(21), 10991 (2022)
31. Roldán-Gómez, J., Carrillo-Mondéjar, J., Gómez, J.M.C., Martínez, J.L.M.: Security assessment of the mqtt-sn protocol for the internet of things. J. Phys: Conf. Ser. **2224**(1), 012079 (2022). https://doi.org/10.1088/1742-6596/2224/1/012079
32. Dos Santos, R.P., Leithardt, V.R.Q., Beko, M.: Analysis of mqtt-sn and lwm2m communication protocols for precision agriculture iot devices. In: 2022 17th Iberian conference on information

systems and technologies (CISTI), pp. 1–6 (2022). https://doi.org/10.23919/CISTI54924.2022.9820048
33. Park, C.-S., Nam, H.-M.: Security architecture and protocols for secure mqtt-sn. IEEE Access **8**, 226422–226436 (2020). https://doi.org/10.1109/access.2020.3045441
34. Sadio, O., Ngom, I., Lishou, C.: Lightweight security scheme for mqtt/mqtt-sn protocol. In: 2019 Sixth international conference on internet of things: systems, management and security (IOTSMS). IEEE, pp. 119–123 (2019)
35. Fontes, F., Rocha, B., Mota, A., Pedreiras, P., Silva, V.: Extending mqtt-sn with real-time communication services. In: 2020 25th IEEE international conference on emerging technologies and factory automation (ETFA). IEEE, vol. 1, pp. 1–4 (2020)
36. Eclipse Foundation: Eclipse Paho. https://www.eclipse.org/paho Accessed 2023-05-29
37. Eclipse: Eclipse Paho MQTT-SN C/C++ client for Embedded platforms. GitHub (2024)
38. Zohourian, A., Dadkhah, S., Neto, E.C.P., Mahdikhani, H., Danso, P.K., Molyneaux, H., Ghorbani, A.A.: Iot zigbee device security: a comprehensive review. Internet Things **22**, 100791 (2023). https://doi.org/10.1016/j.iot.2023.100791
39. Stangaciu, C., Micea, M., Cretu, V.: An analysis of a hard real-time execution environment extension for freertos. Adv. Electric. Comput. Eng. **15**(3), 79–86 (2015). https://doi.org/10.4316/aece.2015.03011
40. Eclipse Foundation: Eclipse Paho Embedded MQTT-SN C/C++ Client. https://www.eclipse.org/paho/index.php?page=clients/c/embedded-sn/index.php Accessed 2023-05-29
41. Stangaciu, V., Stangaciu, C., Curiac, D.: Timer software: A software timer library for embedded real-time systems. Available at SSRN 4527250 (2023) https://doi.org/10.2139/ssrn.4527250
42. Stangaciu, V.: TIMER SOFTWARE. GitHub (2023)
43. Martin, T.: The insider's guide to the Philips ARM7-based microcontrollers. Coventry, Hitex (2005)
44. NXP: LPC2101/02/03 Singlechip 16-bit/32-bit microcontrollers. https://www.nxp.com/docs/en/data-sheet/LPC2101_02_03.pdf. Accessed: 2024-04-05 (2009)
45. NXP: LPC2141/42/44/46/48 SingleChip 16-bit/32-bit microcontrollers. https://www.nxp.com/docs/en/data-sheet/LPC2141_42_44_46_48.pdf. Accessed: 2024-04-10 (2011)
46. Digi International Inc.: XBee®/XBee-PRO S2C Zigbee RF Module. User guide. https://www.digi.com/resources/documentation/digidocs/pdfs/90002002.pdf. Accessed: 2024-04-10 (2022)
47. Stangaciu, C., Micea, M., Cretu, V.: An analysis of a hard real-time execution environment extension for freertos. Adv. Electric. Comput. Eng. **15**(3), 79–87 (2015)
48. Barry, R.: Freertos reference manual: Api functions and configuration options, real time engineers ltd. URL: http://www. freertos. org (2009)
49. Micea, M.V., Cretu, V., Groza, V.: Predictable signal generation with the hard real-time operating kernel Haretick. In: 2005 IEEE instrumentation and measurement technology conference proceedings. IEEE, vol. 3, pp. 2097–2102 (2005)
50. Micea, M.V., Stangaciu, V., Stangaciu, C., Filote, C.: Sensor-level real-time support for xbee-based wireless communication. In: Proceedings of the 2011 2nd International congress on computer applications and computational science. Springer, pp. 147–154 (2012)
51. Cioarga, R.-D., Micea, M.V., Ciubotaru, B., Chiuciudean, D., Stanescu, D.: CORE-TX: Collective robotic environment-The timisoara experiment. In: Proceedings of the Third Romanian-Hungarian Joint Symposium on Applied Computational Intelligence, SACI (2006)
52. Stangaciu, V., Stangaciu, C., Curiac, D.-I., Micea, M.V.: Parsecs_rt: a real-time parsecs-based communication protocol stack for critical sensing applications. Internet Things **25**, 101139 (2024). https://doi.org/10.1016/j.iot.2024.101139
53. Raspberry PI: Datasheet Raspberry PI 3 Model B Technical Specification. [Online], http://www.farnell.com/datasheets/2027912.pdf
54. Dieguez Castro, J.: Arch Linux, pp. 235–252. Apress, Berkeley, CA (2016)
55. Pautet, L., Robert, T., Tardieu, S.: Litmus-rt plugins for global static scheduling of mixed criticality systems. J. Syst. Arch. **118**, 102221 (2021). https://doi.org/10.1016/j.sysarc.2021.102221
56. Intel Corporation: Intel NUC Board NUC5CPYB and Intel NUC Board NUC5PPYB Technical Product Specification, rev. 10. [Online],https://www.intel.com/content/dam/support/us/en/documents/mini-pcs/nuc-kits/NUC5CPYB_NUC5PPYB_TechProdSpec.pdf (2017)
57. Roger Light: Eclipse mosquitto_sub Man Page. https://mosquitto.org/man/mosquitto_sub-1.html. Accessed: 2024-04-23
58. Thomas Nordquist: MQTT-Explorer. https://github.com/ thomasnordquist/MQTT-Explorer. Accessed: 2024-04-23 (2019)

59. Saleae Logic Analyzers. https://www.saleae.com. Accessed: 2025-01-08

Valentin Stangaciu is a lecturer at Politehnica University of Timisoara and a researcher at DSPLabs, part of the Computer and Information Technology Department. His research interests include IoT, communication protocols, and embedded systems, having a PhD thesis in real-time wireless communication protocols. He also introduced the "Data Transmission Coding and Compression" course for the Information Technology Master program. He was a member of different research grants teams, of which three were conducted in partnership with the industry.

Cristina Stangaciu (born Cristina Sorina Certejan) is a lecturer and a research engineer at the Department of Computer and Information Technology, Politehnica University of Timisoara. Her research areas and interests include embedded and real-time hardware-software systems, the Internet of Things, and power management in embedded devices, having a PhD thesis in energy efficient real-time scheduling on embedded systems. She received an award from ANIS România, in 2021 for introducing the "Operating System for IoT" course for the Cloud Computing and Internet of Things Master program. She was a member of five research and development grant teams, of which three represented an academic and industry collaboration.

Bianca Gusita is a PhD student in the Computer and Information Technology field at Politehnica University of Timisoara. The domain of her thesis is Cybersecurity for IoT.

Daniel-Ioan Curiac is a professor in the Department of Automation and Applied Informatics, Politehnica University of Timisoara, Romania and a Senior Member of IEEE. His current research interests include cyber physical systems, robotic systems, real-time adaptive systems, and information security. He is the author or co-author of more than 100 scientific papers.