energies

Article

Internet of Things (IoT) Platform for Multi-Topic Messaging

Mahmoud Hussein [1,2,†,‡], Ahmed I. Galal [2,‡], Emad Abd-Elrahman [1,*,†,‡] and Mohamed Zorkany [1,†,‡]

1 National Telecommunication Institute (NTI), Cairo 11768, Egypt; mah.hussein@nti.sci.eg (M.H.); m_zorkany@nti.sci.eg (M.Z.)
2 Faculty of Engineering, Minia University, Minia 61519, Egypt; galal@mu.edu.eg
* Correspondence: emad.abdelrahman@nti.sci.eg; Tel.: +20-1000-720-268
† Current address: 5 Mahmoud El Miligui Street, 6th district-Nasr City, Cairo 11768 , Egypt.
‡ These authors contributed equally to this work.

Received: 1 May 2020; Accepted: 26 June 2020; Published: 30 June 2020

Abstract: IoT-based applications operate in a client–server architecture, which requires a specific communication protocol. This protocol is used to establish the client–server communication model, allowing all clients of the system to perform specific tasks through internet communications. Many data communication protocols for the Internet of Things are used by IoT platforms, including message queuing telemetry transport (MQTT), advanced message queuing protocol (AMQP), MQTT for sensor networks (MQTT-SN), data distribution service (DDS), constrained application protocol (CoAP), and simple object access protocol (SOAP). These protocols only support single-topic messaging. Thus, in this work, an IoT message protocol that supports multi-topic messaging is proposed. This protocol will add a simple "brain" for IoT platforms in order to realize an intelligent IoT architecture. Moreover, it will enhance the traffic throughput by reducing the overheads of messages and the delay of multi-topic messaging. Most current IoT applications depend on real-time systems. Therefore, an RTOS (real-time operating system) as a famous OS (operating system) is used for the embedded systems to provide the constraints of real-time features, as required by these real-time systems. Using RTOS for IoT applications adds important features to the system, including reliability. Many of the undertaken research works into IoT platforms have only focused on specific applications; they did not deal with the real-time constraints under a real-time system umbrella. In this work, the design of the multi-topic IoT protocol and platform is implemented for real-time systems and also for general-purpose applications; this platform depends on the proposed multi-topic communication protocol, which is implemented here to show its functionality and effectiveness over similar protocols.

Keywords: internet of things (IoT); real-time system (RTS); real-time operating systems (RTOS); IoT protocols

1. Introduction

The proliferation of wireless connectivity in Internet of Things (IoT) devices is rapidly expanding. This is leading to the launch and integration of many IoT services and applications. The IoT is a technology that is widely used for interconnecting devices ("Things") through the Internet. It is used in many applications and fields, such as security, E-health, home automation, emergencies, logistics, smart metering, industrial control and smart cities [1]. Recently, a concept has been applied to IoT platforms that involves the perception of the conditions of the network, the analysis of the gathered knowledge, the making of smart decisions and the performance of actions adaptively [2]. This targets the maximization of the performance of the entire network.

Moreover, an IoT system can integrate cooperative algorithms and mechanisms that can ameliorate problems and promote performance by achieving intelligent actions [3]. This system can detect the network conditions, analyze the gathered knowledge, make intelligent decisions and perform automatic and adaptive actions that maximize the network performance. In this process, multi-domain integration can increase network capacity. Despite the limited research into the intelligent IoT field [4], there are many applications of the technology in different directions such as smart homes and cities [5], drone applications [6], agriculture and farming. Another IoT domain—called the cognitive domain—adds computing algorithms and mechanisms to IoT platforms so that the system devices can make decisions and actions [7]. These devices should have an IoT communication protocol that is responsible for establishing the connection between clients (devices) and the system's main broker server. The server (broker) executes the protocol algorithm with the connected devices, where the algorithm describes the sequences required for a successful communication process.

The common network architecture for IoT systems is centralized networks, as shown in Figure 1; the system may have a large number of nodes which require transmissions between multipoints (clients). In most IoT platforms, to achieve IoT systems requirements, a centralized network between the server and clients is established. In IoT systems, to connect between two devices (nodes, clients), devices should establish a connection with the server, and then the node can send and receive various messages which are described by IoT protocols.

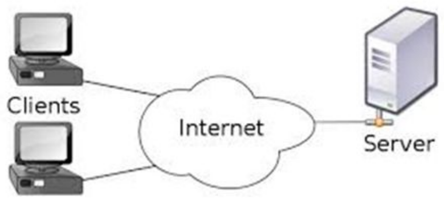

Figure 1. Architecture of the Internet of Things (IoT) network.

MQTT and CoAP message protocols are the most common IoT data protocols. MQTT messages have less of a delay than the CoAP protocol and a smaller message size compared with CoAP, and it is based on a transmission control protocol (TCP) connection, while CoAP uses user datagram protocol (UDP) connections; thus, MQTT has higher reliability than CoAP. MQTT is more suitable than CoAP for real-time systems as fewer overhead bytes are added to the messages being transferred [8].

This paper can be considered as an extension of our conference paper [9] which initiated the main framework. In this research, a new message protocol is proposed for IoT applications. The proposed protocol is designed to overcome an issue which has appeared with MQTT: multi-topic non-support messages, which require extra bytes and cause increased delays from the overhead side. On the other hand, the proposed data protocol handles the multi-topic messages which have become a feature of the system; this will be discussed and detailed in the following sub-sections.

In this work, the MQTT protocol is selected as the most famous IoT standard protocol for comparison with our proposed protocol. There are two ways to simulate the proposed protocol compared with standard MQTT: using ready-made solutions that depend on using open source programs (e.g., Mosquitto (https://mosquitto.org/)— Eclipse Mosquitto is an open-source message MQTT broker (Eclipse public license (EPL)/Eclipse distribution License (EDL)) that can be used to implement the MQTT protocol and support different versions 3.1, 3.1.1, and 5.0. Mosquitto supports Windows, Mac, Linux, Debian, Ubuntu, and Raspberry Pi that are commonly used, or building a new IoT platform from scratch. In this research, we decided to design an IoT platform from scratch to support real-time applications; thus, the simulation and real implementation results are discussed after implementing the IoT protocol based on the proposed architecture. Then, the proposed protocol is compared with the standard MQTT protocol in terms of message overheads and transmission delays.

Real-time systems (RTSs) are systems in which the output occurs in real time and must be correct; thus, most IoT systems should be RTSs. It is important to tailor to critical systems that have deadlines at a critical time, and these can be classified into two types: soft and hard RTSs. If a delayed time requirement is accepted, the RTS system is called a soft RTS; if not, then it is called a hard RTS. To meet the critical deadlines in an RTS, real-time operating systems (RTOSs) will be used [10]. An RTOS is an operating system (OS) that is used for embedded RTSs. This OS is used as it guarantees capabilities such as compactness, high performance, predictability, reliability and modularity [11]. RTOS supports many services such as time, memory and task management, as well as providing multi-tasking. Taking advantage of all of these features of RTOS, the proposed platform will be implemented based on this system. Research perspectives regarding RTOS for IoT investigating such themes as adjusting RTOS to work with IoT systems, the implementations of IoT platforms, IoT frameworks and IoT performance evaluation were discussed in [1].

IoT platforms make IoT development simpler, as all IoT clients (devices with Internet access) are connected to the broker (server). Using an IoT platform, we can publish sensor data from clients to other interested clients (subscribers) through the IoT and take further actions through actuator nodes. Much research has been undertaken into implementing IoT platforms [12–14], but most of them support neither the nature of RTSs, where time is critical, nor specific applications. Furthermore, most research has used ready-made protocols such as MQTT [15]. Thus, in our research, an IoT platform based on RTOS will be implemented using the proposed multi-topic communication protocol.

In the rest of this work is structured as follows: Section 2 presents an overview of the relevant IoT protocols. Section 3 introduces the proposed multi-topic IoT protocol. The proposed IoT platform based on RTOS is detailed in Section 4. The experimentation setup phases are highlighted in Section 5; then, experimental and simulation results are shown in Section 6. In Section 7, the main characteristics of the multi-topic protocol are discussed. Finally, the work is concluded with some prospective research directions in Section 8.

2. State of the Art

2.1. IoT Protocols

IoT systems are characterized by remote monitoring and control aspects. These aspects allow IoT components to communicate together through a remote service that is powered by Internet communications. System nodes are connected through a predefined communication protocol. The data protocol of IoT systems provides different numbers of message frames which enable remote messaging between IoT system nodes. There are many IoT protocols for IoT systems, including MQTT, MQTT-SN, AMQP, DDS, CoAP and SOAP, but MQTT and CoAP are the most common protocols [16].

MQTT is the most dominant IoT communication protocol; it is a pub/sub message system for limited-resources device and unreliable networks and was developed by IBM [17] and standardized by the Organization for the advancement of structured information standards (OASIS).

Another common IoT data protocol is CoAP; this is a recently developed protocol which must be used for communication by constrained devices [18]. It depends on the "representational state transfer" (REST) mechanism, which supports "request–response" models such as HyperText Transfer Protocol (HTTP).

Many IoT protocols support single-topic messaging. This type of messaging implies that one topic only per message can be sent through the network (a single topic such as publishing the temperature, pressure, humidity, etc.). The message topic is important information that is required to be published to subscribers. A subscriber node is a node that explicitly requests any published messages for a specific topic, as shown in Figure 2.

For example, if an MQTT client (publisher) has three sensors (such as temperature, pressure and humidity) and we need to send each sensor reading to a specific application instance client, (for example, sensor_1 sends to application instance_1, sensor_2 sends to application instance_2 and sensor_3 sends to application instance_3), we must send three different messages from the publisher, and each message has a unique topic.

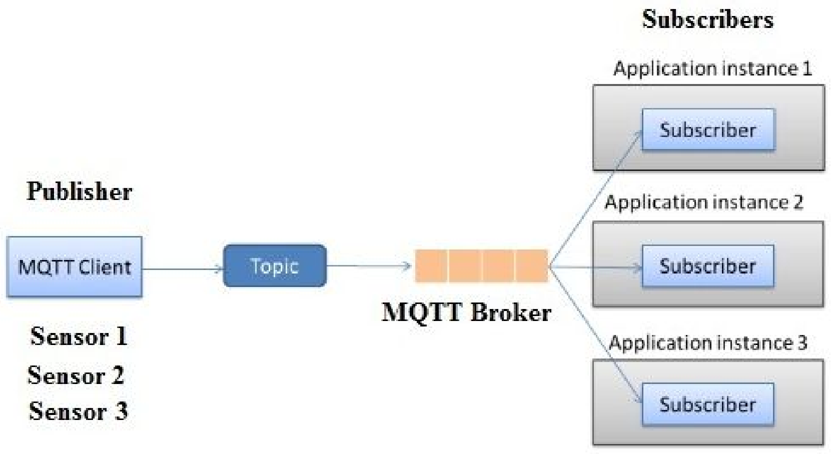

Figure 2. Publish/subscribe architecture based on single-topic messaging.

An IoT device should be able to send multiple messages for different topics. Therefore, the IoT protocols enable nodes to send many messages, but every message contains one topic only (i.e., the status of one sensor only). In our research, we propose the multi-topic feature in which a message can contain many topics for different subscribers without any waiting delays such as incurred by the message batching technique.

In recent years, the technique of batching multiple messages has been introduced in some Cloud system applications such as the Google Cloud Pub/Sub system which supports multiple message batching (https://cloud.google.com/pubsub/docs/publisher). However, batching multiple messages does not mean multi-topic messaging; batching messages puts messages into a queue until completion, which implies latency, and so batching is not suitable for real-time system applications. On the other hand, the proposed multi-topic messaging technique automatically sends any ready number of topics without waiting or latency making, it more suitable for supporting real-time system applications.

IoT systems could be adapted with promising technologies such as edge computing. This type of computing does not replace the MQTT protocol. Companies such as Cisco benefit from edge computing in the IoT field by adopting MQTT; this was highlighted by a senior manager at Cisco (https://blogs.cisco.com/internet-of-things/setting-a-simple-standard-using-mqtt-at-the-edge#comments).

Although MQTT is suitable for embedded real-time systems, the feature of multi-topic messaging is not supported by the protocol. Thus, in this work, a modified multi-topic messaging protocol that supports the multi-topic feature is implemented, as discussed in the next sections. This feature reduces network traffic and reduces the delay required to publish multi-topic messages.

2.2. *Multi-Message versus Multi-Topic Techniques*

In this part, we differentiate between two concepts: the multi-messages technique and our proposed multi-topics messaging technique.

2.2.1. Multi-Messages Technique (Batching)

The multi-messages techniques (i.e., batching), as shown in Figure 3, involves batching and buffering on senders to group multiple messages and send them as one batch to increase the throughput and cut down simple queue service (SQS) costs; however, this method has an impact on latency. In this method used, for example, in (https://codeahoy.com/2017/08/03/message-batching-to-increase-throughput-and-reduce-costs/), the customization of the batching algorithm can depend either on a specific number of messages (for example, 15 messages) or wait until threshold time value (for example, 50 ms). Thus, the batch will be formed either for a certain number or value and then sent out. This could cause applications to crash due to the queuing and processing sequence of messages. Therefore, we can lose some or all messages, and this method is thus not suitable for critical and real-time applications. Examples of these methods include message-batching, cloud.google (https://cloud.google.com/pubsub/docs/publisher), pulsar.apache (https://pulsar.apache.org/docs/ja/concepts-messaging/) and cloudkarafka (https://www.cloudkarafka.com/blog/2019-09-11-a-dive-into-multi-topic-subscriptions-with-apache-kafka.html).

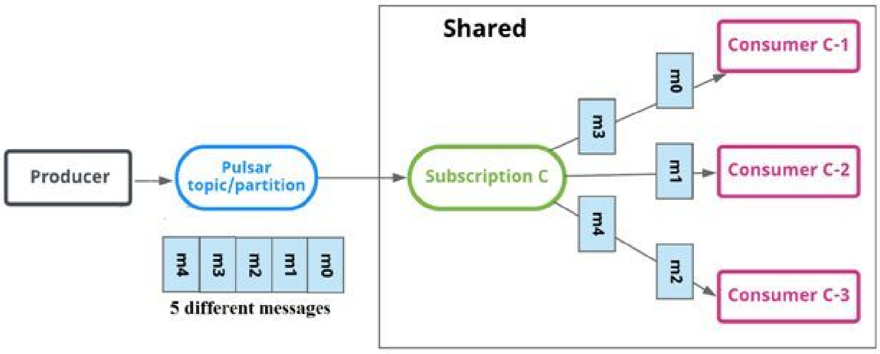

Figure 3. Multi-messages technique using batching; for example, pulsar.apache.

2.2.2. Multi-Topic Messaging Technique

The approach shown in Figure 3 is the multi-messages technique; however, in our proposal, we consider multi-topic messaging rather than multi-messaging. In our proposed multi-topic messaging protocol, we can send a single message with many topics; for example, as shown in Figure 4, three topics (i.e., temperature, humidity and pressure) are all sent in one message by the publisher to the broker. Then, the broker (proposed server) can detect and split this message into the number of new messages (three messages according to the three inherent topics in the received one) and resend each topic in a message to its specific client (each subscriber) without delay and loss in the data. The system acts intelligently as if it had received three separate messages from the publisher although it received one.

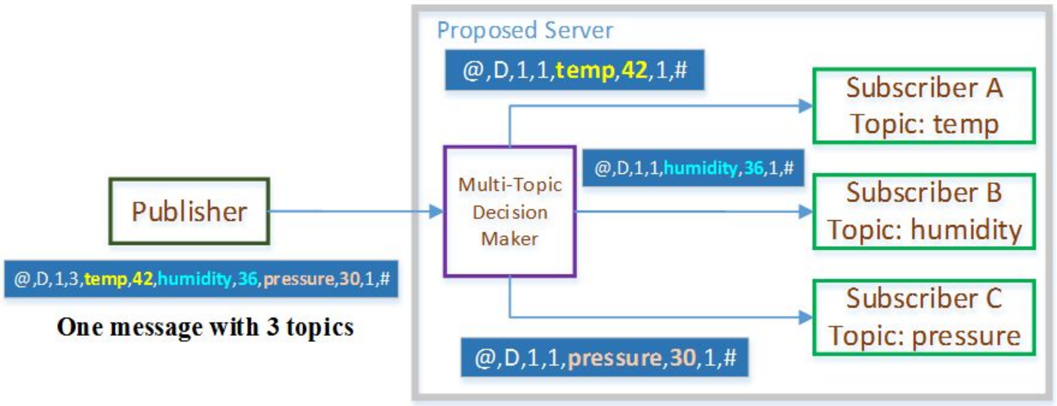

Figure 4. Proposed message with the three-topics technique.

Table 1 summarizes the main differences between the proposed protocol and four of the most common approaches in the IoT field; MQTT, Google Pub/Sub, Pulsar, and Karafka. This summary shows three different aspects: multi-topic subscriptions, multi-topic publications and multiple-messages queuing. The multi-topic subscription message means that a client can send one message that subscribes to many different topics. The clients at the beginning of the communication send this message only once to tell the broker that this client is interested in these topics. One publishing message contains many topics, but multiple-messages queuing is a batching-based technique that aggregates multiple different messages into a queue.

Table 1. Comparison against relevant protocols.

	MQTT [17]	Google Pub/Sub	Pulsar	Karafka	Proposed
Multi-topic subscriptions	Yes	No	Yes	Yes	Yes
Multi-topic publications	No	No	No	No	Yes
Multiple-messages queuing (batching)	No	Yes	Yes	Yes	No

2.2.3. Studied Use-Case Scenario

Let the IoT client (a patient with medical sensors: an electroencephalogram (EEG) signal and electrocardiography (ECG) signal and temperature, pressure and glucose level monitors) have five sensors and our aim be to send each sensor's status to a specific doctor (physician). The current solution in MQTT and batching or multi-messages methods, such as in the famous apache "pulsar.apache.org" method, is to send five different messages, with each message having frame headers; quality of service (QoS); for example, a QoS equal to 2 requires four acknowledgment messages between the client and server for one message) means more delays and more overhead bytes. The second solution is to let the client (a patient with five medical sensors) concatenate these five messages into one message (with a single topic), but in this case, the server will send this message to all destinations (five physicians). In this case, it would not be possible to separate the message to send each topic to a specific physician, as shown in Figure 3. On the other hand, in our proposal, we can send a single message with five topics (five patient sensor statuses) in a single message to the broker. Then, the broker can make the decision whether to separate this message into five new messages and resend each message to a specific physician, as shown in Figure 4 for the three-topic example.

3. Proposed IoT Multi-Topic Messaging Protocol

The proposed multi-topic data protocol is simply highlighted in this section. It is introduced either to establish the connection or to start the communication between IoT nodes. Moreover, it is designed to solve the single-topic messaging problem by supporting multi-topic messaging. This multi-topic feature acts as a brain for the messaging IoT broker. Acting as an intelligent IoT system, it converts normal IoT nodes to smart nodes that obtain the published messages and analyze the obtained messages to select the message destination. The resulting system can make the forwarding decisions implemented by the broker. While a client can send many types of data with different content in the same message without sending the data using many messages, the multi-topic feature provided by the proposed protocol can reduce the required traffic for the nodes, meaning that the protocol could be used in low-bandwidth networks and with limited hardware requirements.

Moreover, the IoT broker is modified from simply acting as a connecting point between clients to processing the received messages as it can separate one multi-topic message into many messages and display each message as a new message from a new client. Thus, it appears that the IoT broker has a "brain" and can make a simple decision when decomposing a multi-topic message.

We therefore propose new software (i.e., the broker IoT server) that can mimic human brain function (as it reads/inspects messages to check the number of topics and re-create a new number of messages according to the number of topics in the received message). Moreover, it makes decisions by sending each message from these newly generated messages to different destination clients. Thus, the single incoming message to the proposed broker will be split into many messages and each message re-transmitted to different destinations. Furthermore, for the content aspect of intelligent IoT platforms, our proposal adjusts the content of the message and re-creates new messages for a particular type of audience, as shown in the studied use case in Section 2.2.3.

3.1. Protocol Architecture

As shown in Figure 5, the proposed protocol architecture is similar to MQTT except for the added value of the multi-topic messaging feature in our proposal. The proposed data protocol depends on TCP/IP and consists of a centralized server (broker), IoT nodes (clients) and a multi-topics communication protocol:

Protocol Application Range: All applications that the IoT supports can use the proposed protocol as a communication IoT protocol. The application range includes enterprise, utilities, mobiles and home applications.

Communication Nodes: The server (broker), which is topic-based, is responsible for connecting nodes (client/devices). The different IoT nodes communicate with others through the server to accomplish the functionality of the system. A sequence diagram that describes each node type is shown in the next sections.

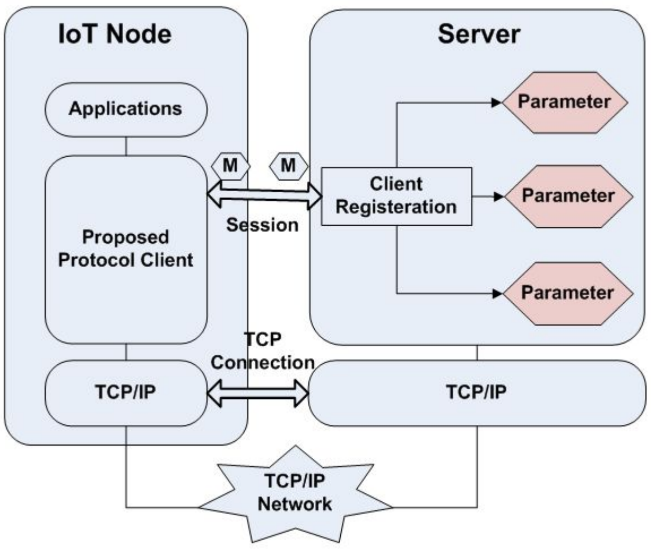

Figure 5. Architecture of the proposed multi-topic IoT protocol.

Nodes can be classified into four categories; sensor nodes, actuator nodes, normal or hybrid Nodes and finally, monitor nodes.

3.1.1. Sensor Nodes

The sensor node opens a TCP connection by sending a "connect" message to the broker server, as shown in Figure 6. This node must also send an identification number (ID) to the broker server to identify this node on the system. Then, it waits for an acknowledgment message (ACK) from the server; after that, it will be ready to publish the sensing data (either periodic or at interrupt times) or the information to the broker.

An IoT node can publish a message periodically or when an event occurs. Events may be generated from hardware interruptions (e.g., a push-button is pressed) or software triggers (e.g., a temperature exceeds a specific value).

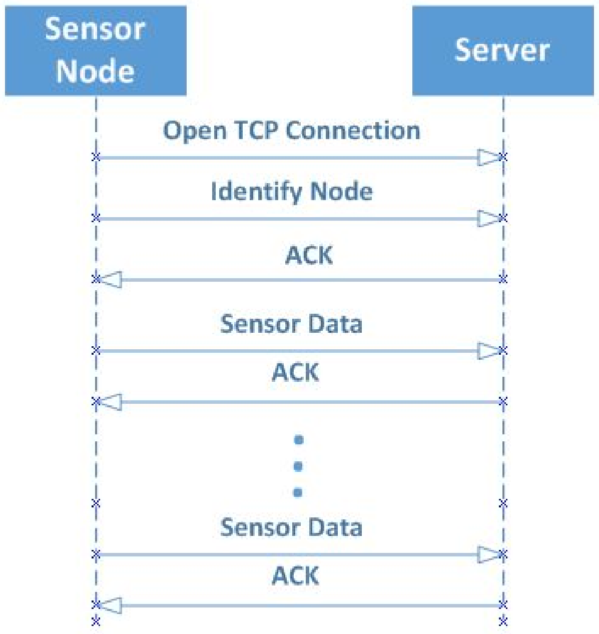

Figure 6. Sequence diagram for a sensor node. ACK: acknowledgement message.

3.1.2. Actuator Nodes

The actuator node and the main server first open a TCP connection. Then, the identification process is performed. Afterwards, this node will receive commands or messages which are transmitted by the monitor nodes (i.e., monitor clients) through the main server, as shown in Figure 7.

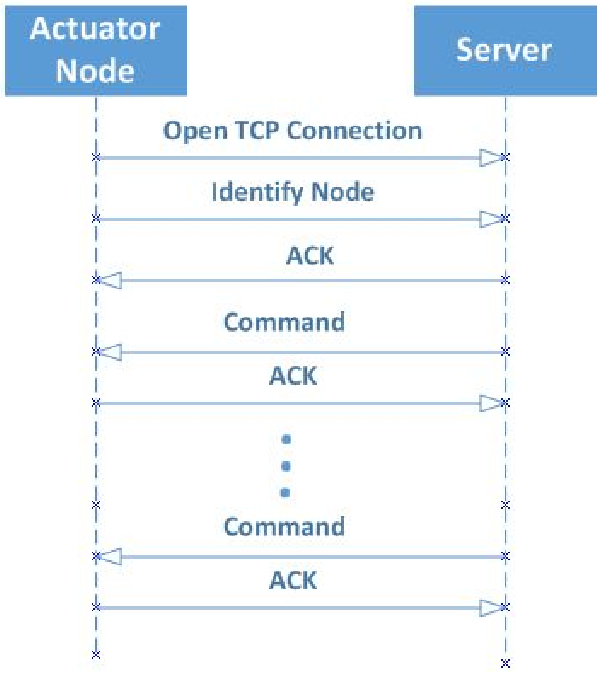

Figure 7. Sequence diagram for an actuator node.

3.1.3. Normal Nodes

A normal node performs the functionality of actuator and sensor nodes: it opens the TCP connection, then it identifies itself to the server with an identification number. Afterwards, it sends the sensed data to the server, and any actions or commands received from the server by the monitoring notes will be also executed through this node, as indicated in Figure 8.

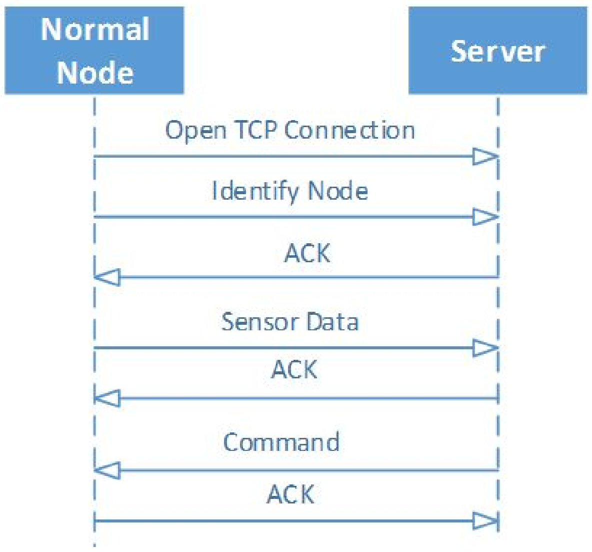

Figure 8. Sequence diagram of a normal node.

3.1.4. Monitor Nodes

The monitor node opens a TCP connection with the server after the identification process is performed; it registers to receive data from certain sensor nodes and then sends its messages to a specific actuator via the broker server, as shown in Figure 9.

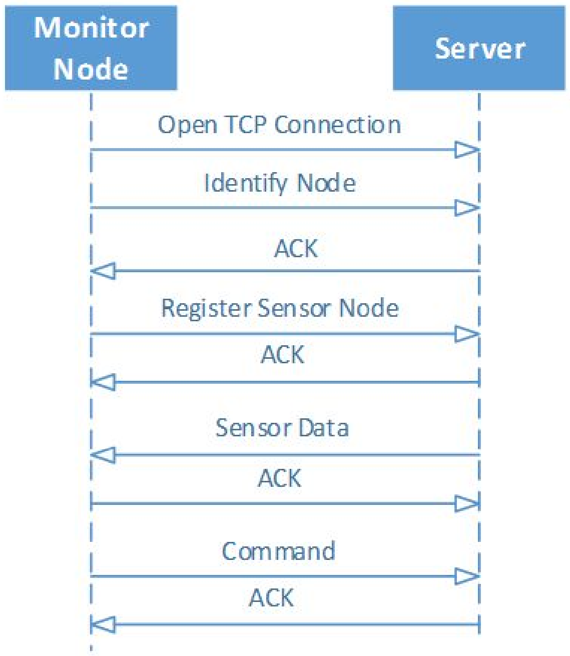

Figure 9. Sequence diagram of a monitor node.

3.2. Node to Node Communication

All node to node (client to client) communications must be done through the broker. Each client/node will try to establish a TCP connection and identify itself with the server. Then, it waits for an ACK message back from the server. After receiving the ACK message, successful communications between clients or nodes through the broker server can be done by sending and receiving different communication packets, as shown in Figure 10.

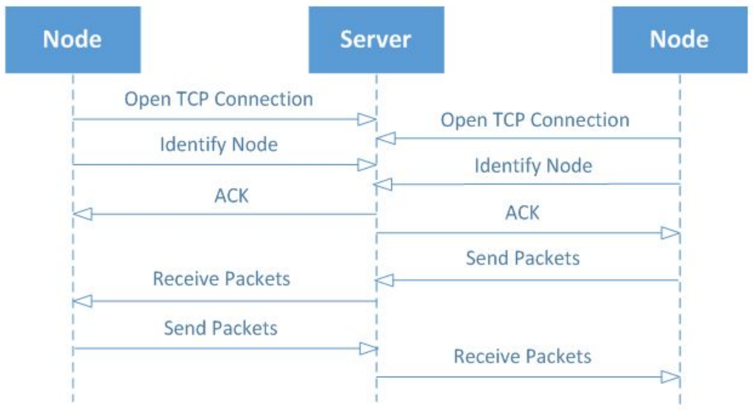

Figure 10. Sequence diagram for node to node communication.

3.3. *Protocol Frame Format*

Any kind of Internet connection, such as Wi-Fi, Ethernet, General packet radio services (GPRS), and 3G or 4G data connections, can be used to communicate between IoT nodes. Wi-Fi technology is selected in our simulation to communicate nodes with the server. The Wi-Fi hardware implementation module has a built-in TCP/IP stack as the proposed protocol depends on a TCP/IP protocol such as the MQTT protocol. The proposed multi-topics protocol has four type frames: identification, ACK, registration and finally, data frames. Table 2 indicates the common fields used in the proposed frame structure; all frames must have the same start (@) and end (#) of frame indicator. The frame types are shown in Table 3.

Table 2. Description of common frame fields.

Frame Field	Field Description
@	Start of Frame
ft	Frame Type
nid	Node ID
#	End of Frame
,	Frame delimiter
sn	Sequence Number
pc	Parameter Count
as	ACK State
pid	Parameter ID
pv	Parameter Value

Table 3. Frame Types.

Frame Field	Field Description
I	Identification frame
R	Registration frame
A	Acknowledgment frame
D	Data frame

3.3.1. Identification Frame

The identification frame is first transmitted by a node to declare and identify itself to the server. This phase is done by using a specific identification number (ID), and the system should wait until receiving the ACK message from the broker server. After that, each node can transmit (publish) or receive (by re-publishing from the server) frames. The frame structure needed for the identification node is shown in Figure 11. This frame is used in sensor nodes, actuator nodes, monitor nodes and hybrid or normal nodes. The function of this frame is similar to the "connect" message in the MQTT protocol.

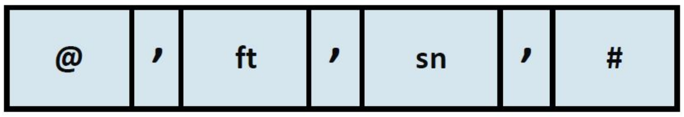

Figure 11. Identification frame structure.

3.3.2. Acknowledgment Frame

The acknowledgment (ACK) frame message is sent back from the broker server to the sender client to acknowledge the transmission process. This frame is transmitted from the server "broker" to the client "node". If the requested acknowledgement field is set, then an ID of the frame must be followed in the ACK frame. Figure 12 shows an ACK frame. This frame is used by the server as an acknowledgment to the sender node for all frame types. The function of this frame is similar to the "Connect ACK" message in the MQTT protocol.

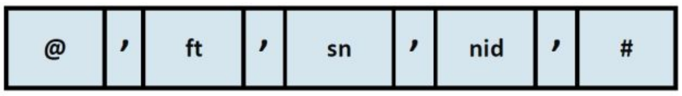

Figure 12. Acknowledgment frame structure.

3.3.3. Registration Frame

When the node listens to (receives a message) a specific parameter (topic), it must first send the registration frame with PIDs (parameter IDs) and PCs (parameter counts) as indicated in Figure 13. After the registration process, any message sent from other nodes with those parameters (topics), will be published to the node through the server; the ID frame should be sent first. This frame is used in actuator nodes, monitor nodes and normal or hybrid nodes. The function of this frame is equal to the "subscribe" message in the MQTT protocol.

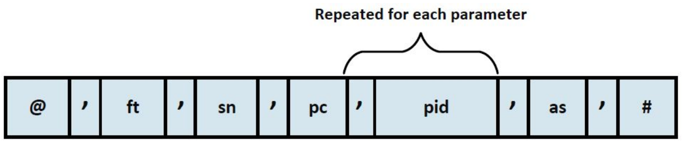

Figure 13. Registration frame structure.

3.3.4. Data Frame

With the data frame, the node can share and publish its data to other clients (nodes), as shown in Figure 14. The data frame has multiple parameters, such as a count field (PC), an ID (PID) and a value (PV). These parameters must be registered to use PID. Furthermore, the ID should be transmitted at the beginning of the process. This frame is used in sensor nodes, monitor nodes and normal or hybrid nodes. The function of this frame is equivalent to the "publish" message in the MQTT protocol, but it supports multi-topic messages.

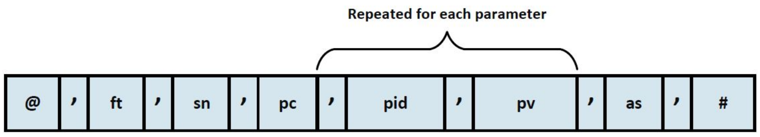

Figure 14. Data frame structure.

4. Proposed IoT Platform Based on RTOS

In this section, the design and implementation of the proposed IoT platform will be highlighted for critical and general applications and also for normal and real-time systems. The system uses FreeRTOS [19] as an RTOS to meet critical deadlines on time. Our platform depends on the proposed multi-topic protocol to support the "multi-topic" messaging feature. This will enhance the traffic throughput and decrease the delay required for multi-topic messages. The platform consists of subsystems such as IoT nodes and a broker server which communicates between nodes. Moreover, the proposed multi-topics protocol manages messages between IoT nodes and the broker server. This platform implementation uses the frame structure, which indicates how to exchange the data. The proposed platform can be easily used in a wide range of applications as it provides IoT connectivity and reliability and it uses the Wi-Fi module (we can use any type of Internet access) for the TCP/IP stack to decrease the hardware cost. The advanced RISC machine (ARM) Cortex-M4 is used to implement the proposed platform, which is powerful and has good power consumption. Furthermore, the IoT node can be a smartphone (i.e., a smartphone with the proposed IoT client application). The communication between system nodes through the Internet is based on the use of the broker as a backbone. This is proposed to be a topic-based broker, as per the MQTT message protocol, and it is designed based on RTOS.

4.1. Design of Real-Time System (RTS)

An RTS is a system that has a correct output that must be executed at the right time. Moreover, any RTS has many time requirements and critical deadlines that must be satisfied. These requirements must be handled by the RTOS used in system development. Therefore, the RTOS must have specific features that consider IoT challenges simultaneously, including scalability, connectivity, modularity, safety and security.

- Scalability refers to the IoT system's ability for future extension (i.e., system expandability).
- Connectivity refers to whether the RTOS is compatible with and supports many communication protocol standards.
- Modularity refers to the support of the implementation of the modules of the system. This will simplify the addition and integration of new features to smart devices.
- Safety refers to the prevention of any malfunction behavior that can lead to undesirable action.
- Security refers to the importance of countermeasures against either threats or malicious attacks.

Figure 15 shows the RTOS architecture in the proposed system. The RTOS has a great significance for the proposed IoT systems. This is because the use of the RTOS in development will add new advantages to the system, such as increased reliability, efficiency and predictability. Besides, it can simplify the management of the system [20]. As the platform design is done using RTOS, the system design could be listed in four system tasks, as shown in Figure 16.

Task "T_Connect" is the responsible task for connecting with the broker through the Wi-Fi communication module. Task "T_Comm" is responsible for receiving and transmitting the data from and to the remote main IoT server. Task "T_Sensor" gathers the information from the sensors in a special form and then sends the data to the main server. Finally, task "T_Actuator" executes the received commands by the "T_Comm" which are sent from the remote monitor node.

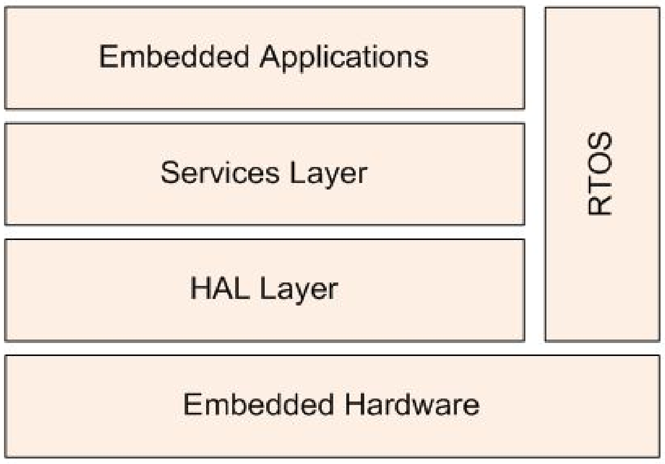

Figure 15. Real-time operating system (RTOS) architecture.

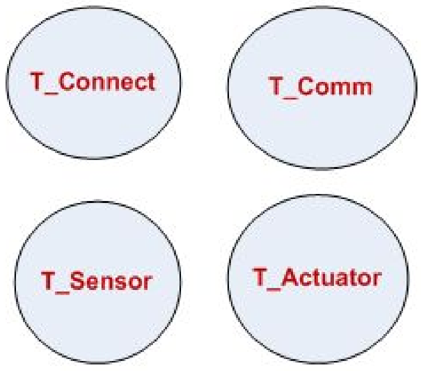

Figure 16. System tasks.

The system implementation scenario is as follows: the system peripheral initialization is done first, and then the operating system services used in the real-time design are created and initialized with the default state parameters. The system tasks should be created to allocate the required memory to become ready to operate. Finally, the operating system should be started to enable the scheduling of system tasks by the operating system scheduler.

The proposed IoT platform is implemented based on FreeRTOS (V10.3.1, Real Time Engineers Ltd., MIT License). It is possible for any task to block on a specific synchronization event with a specific time; it will exit from the blocked state even if the waiting event does not occur due to a time-out. The ready state tasks are able and ready to run but not in the running state currently as they have a lower priority than the already running task; all system tasks and the transitions between them are shown in Figure 17.

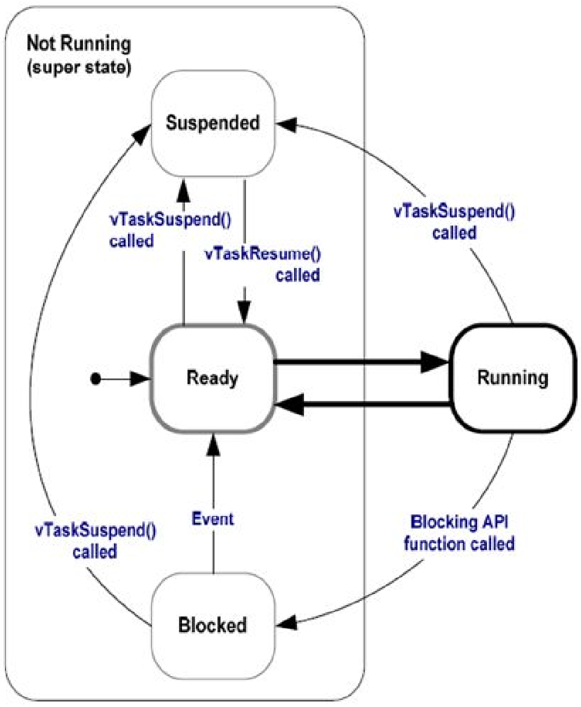

Figure 17. Task state diagram.

4.2. Proposed IoT Nodes

The proposed platform is an embedded system that could be implemented and designed for any type of micro-controller to meet system requirements. The proposed IoT node was designed and implemented in our laboratory at the National Telecommunication Institute (NTI) (http://www.nti.sci.eg/pcblab/) (as a printed circuit board (PCB) prototype) and passed all tests (PCB, unit, simulation and field tests) successfully. The proposed prototype of the IoT node was designed and implemented based on STMicroelectronics (STM) Nucleo Board. This board uses the ARM Cortex M4 processor. The ARM processor in the IoT node is developed for high-performance and to decrease the cost of devices as well as decreasing the power consumption.

Each node consists of different units, such as a micro-controller to manage node tasks and a Wi-Fi hardware stack to connect the client to the wireless network as an Internet access method. Furthermore, the proposed node has different sensor interfaces to sense a process or an environment and output interfaces (actuators) to control the environmental effects automatically.

All system nodes are connected together through the main broker server for specific designed tasks. For example, the proposed generic node that contains three sensors (i.e., a patient IoT unit with three medical sensors: an ECG signal and temperature and glucose level sensors), and we need to send each individual sensor status to the relevant physician. In addition, the proposed client has two actuators (i.e., turn-on alarm sound and automatic insulin dose) as shown in Figure 18.

In detail, we designed the IoT client unit to monitor the status of the patient and publish a message periodically (i.e., every 30 min) if the sensor readings correspond to normal conditions. However, if the status of any sensor is changed to an abnormal state (i.e., a patient temperature exceeds a threshold value, or the glucose level or ECG signal exceeds or drops below a certain value), the proposed IoT client understands the changes and makes the following automatic decisions;

- Publish this sensor status immediately to the physician.
- Increase the sending rate so that it is more suitable for the new changing rate (i.e., every 30 s).
- Run actuators, such as turning on the alarm sound or automatically injecting an insulin dose to save a patient's life.

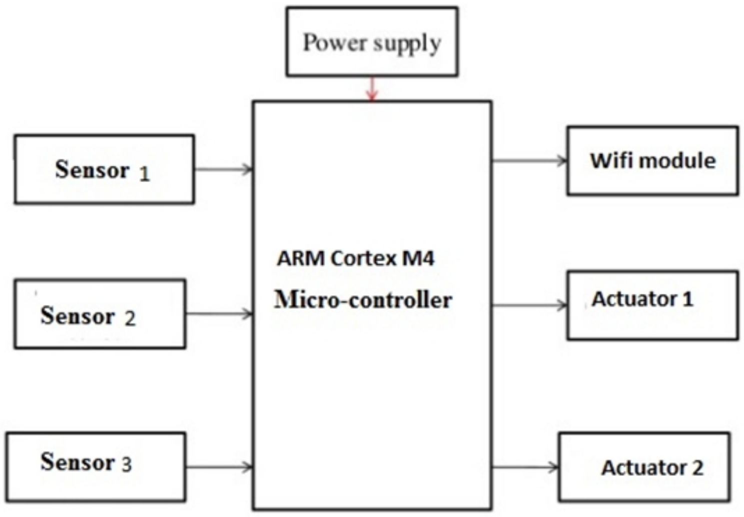

Figure 18. Block diagram of the proposed embedded IoT client.

This process is performed in an adaptive way for one sensor or many sensors, as clarified in our proposal for the IoT client.

- Understand the sensor status.
- Take an automatic decision to publish a message periodically (in normal status) or when an event occurs (patient abnormal).
- Send all sensor statuses in one message with multi-topics to the IoT server (to minimize delay), then send each sensor reading to its related subscriber (physician).
- Make the decision to run a specific actuator if required.

Based on the IoT node tasks, we can categorize nodes into four main types: sensor nodes, actuator nodes, normal nodes and monitor nodes.

The sensor nodes sense the environment and send the sensed data to the server periodically with a certain configuration period. These nodes include one sensor or more and do not include output interfaces (actuators). Figure 19 shows the sensor node flowchart.

Actuator nodes can affect or control the environment through messages (commands) from other monitor nodes via the broker server. These nodes include one or more actuators and do not include sensors. Figure 20 shows the actuator node flowchart.

Normal nodes have the functionality of the sensor and the actuator nodes; these nodes include actuators and sensors, as shown in Figure 21. A node's behavior is to communicate first with the IoT server and then send its sensor data to the server. It receives and executes the commands that come from monitor nodes through the broker server.

Monitor nodes could be proposed as hardware nodes or smartphones which monitor and control system nodes; they are the nodes which do not include sensors or actuators. These nodes can monitor sensors or control the actuators through publishing commands (sending messages), also receiving and processing the sensor data. The node communicates with the IoT broker server, and if the connection is established correctly, then it identifies itself to the server. It will register for topics and the parameters required from other nodes in the IoT system. Whenever it has new commands for the other nodes in the system, it will send them to the server directly for the command-specific topics or parameters.

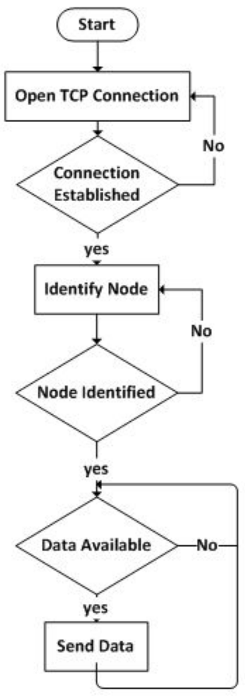

Figure 19. Sensor node flowchart.

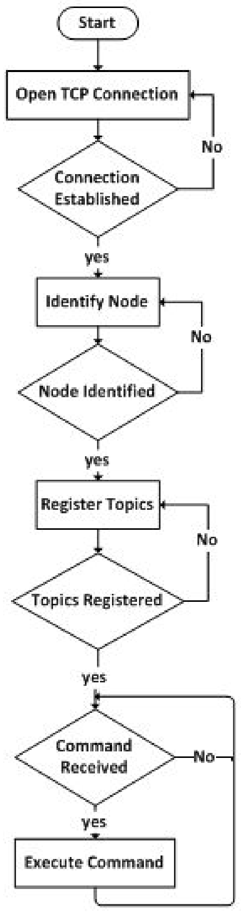

Figure 20. Actuator node flowchart.

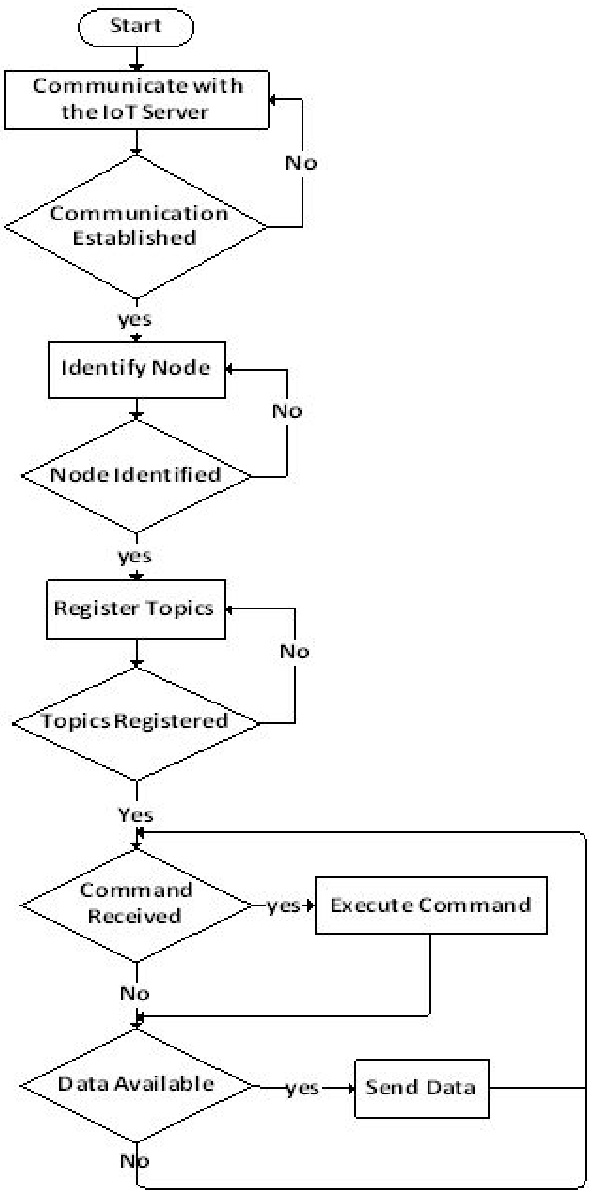

Figure 21. Normal node flowchart.

4.3. Android Device as a Client Node

As the monitor node could be an Android device; the Android application layouts are designed. The main layout is responsible for the communication with the IoT server, and the communication layout is responsible for the data exchange with the other nodes. This node application is implemented based on Java programming for Android development by the Eclipse IDE (release: 4.15, Eclipse Foundation Inc.) ; the main activity is responsible for handling the main application layout components, which are used to connect the broker with the IP address and the port number of the server.

4.4. IoT Server (Broker)

The IoT server is designed and implemented from scratch, independent of any ready-made server solutions; it is implemented based on the Java programming language. The basic block in the system is the IoT broker application, which communicates and connects all IoT Nodes. Thus, an IoT server is implemented for the proposed system, and the server is able to communicate with all types of nodes and is responsible for enabling monitor nodes to visualize the data of different sensors. The server's behavior, as shown in Figure 22, is to communicate the IoT Nodes.

If the sensor node is connected to the server, it forwards messages to the monitor node (registered node); if the connected node is an actuator node, the server will send the monitor message commands to it; if it is a normal node, the server will do the same thing for the actuator and sensor nodes; and if it is a monitoring node, it sends commands (messages) to the server and then forwards them to the actuator node.

The proposed broker server thread is implemented based on Java programming. It starts by listening to a specific server port to become ready for any node connection. If a new node tries to connect, then a new thread is created; the created thread is called the IoT device thread and is responsible for handling the IoT client node connection for the data exchange between this node and the other nodes in the system.

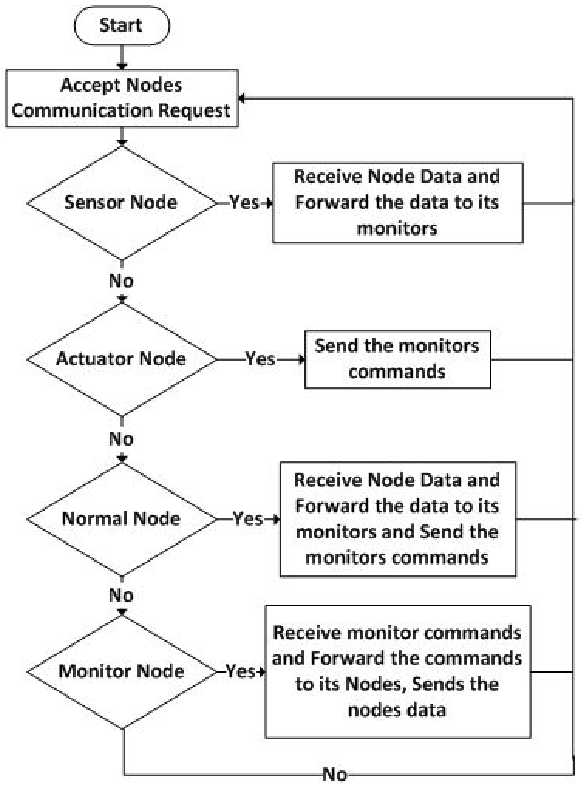
Start
Accept Nodes Communication Request
Sensor Node
Yes
Receive Node Data and Forward the data to its monitors
No
Actuator Node
Yes
Send the monitors commands
No
Normal Node
Yes
Receive Node Data and Forward the data to its monitors and Send the monitors commands
No
Monitor Node
Yes
Receive monitor commands and Forward the commands to its Nodes, Sends the nodes data
No

Figure 22. Server (broker) flowchart.

5. Experiment Setup

To study the performance of our proposed protocol, we carried out experiments. Additionally, the proposed protocol was compared with the standard MQTT protocol. This was done by setting different parameters of the network which affect the protocol's performance. The experimentation phase was executed inside our NTI premises on the real network infrastructure of the institute. We chose two performance metrics in the experiments: the delay time and the total transmitted bytes. The first metric concerns the delay of the transmitted message, defined by the interval time between sending frames (i.e., the publishing of a message) and the returned ACK message answered by the server, while the second metric concerns the total transmitted bytes per successfully sent message.

5.1. Hardware Setup

The hardware setup phase, as shown in Figure 1, consisted of three PCs/Laptops. The first one is used for the WANem (version 3.0) software (http://wanem.sourceforge.net/) to simulate the effects of channel losses to simulate transmission losses. Additionally, using the WANem server, we can simulate communication delays. The server (the computer which acts as a broker) for node communication and another laptop (client node) act as a node that publishes the message and waits for it to be acknowledged. Wireshark Software (https://www.wireshark.org/) is installed on the node (client) to monitor the traffic that is used later on for the analysis phase. The proposed multi-topic protocol and the MQTT protocol run on the same server. Each message is published from the IoT node to the broker server, going through the WANem node (the machine that simulates the network delay and packet drops). Finally, the server's ACK message will be returned to the transmitted node through the WANem server machine.

5.2. Software Setup

The Mosquitto (MQTT broker) as an open source MQTT broker is implemented based on the last updated version of the MQTT standard, which is v3.1, and the proposed multi-topic protocol broker-sides are implemented using Java coding. The proposed software of the broker is designed, implemented and tested with the same scenarios of the MQTT protocol. The software tools used in the practical experiment are the Wireshark software for packet analysis and the WANem software for network emulation.

The Wireshark tool can be used to measure and calculate metrics such as packet length and delay time; the software can monitor the network packets, the time elapsed for each packet and also the total traffic over a network. It can also generate reports after the monitoring of the network at a certain time; these reports are helpful for accurately studying the network traffic.

The WANem software is used for network delay and packet loss insertions. Network conditions do not have a fixed behavior, and so a network-related protocol evaluation and comparison cannot be fair because of the variant network conditions. For fixed network conditions, the WANem software is used; it can insert different network conditions for each test, including packet delays and packet loss percentage. These insertions help us to test under specific network conditions for a practical network test rather than simulation tests.

6. Simulation and Experimental Results

To test the proposed platform, we built an IoT platform consisting of one broker server,the WANem machine (PC), and a client, which were connected as a simple setup for the experiment. Both protocols (the proposed IoT communication protocol & the MQTT protocol) can achieve message delivery without applying a packet loss rate percentage. Successfully achieving this scenario would indicate that the two protocols have good communication and message delivery rates under different packet loss ratios. Thus, we can study the performance either in terms of the message delay or the successful amount of transmitted data, as indicated in the next sub-sections.

6.1. Message Delay

The message delay can be defined as the time interval between sending a message (i.e., publishing one) from the IoT node and receiving the ACK (returned message reply) from the broker node/server. This is calculated as shown in Equation (1). The message delay metric is a significant parameter for real-time systems where time is critical or sensitive. Different packet loss rates are applied to have ameliorate the message delay as a result of message re-transmission. Moreover, the MQTT protocol, which has a quality of service (QoS) equal to 1, is compared against our proposed multi-topic protocol with An ACK data state (QoS) equal to 1 for the same messages.

$$\overline{D} = \frac{1}{n}\sum_{i=1}^{n}(T_{ri} - T_{si}) \tag{1}$$

where $\overline{D}$ is the average message delay at a certain loss percentage, n is the number of messages, T_{ri} is the reception time of the message ACK and T_{si} is the sending time of the message.

MQTT has a small frame size compared to the multi-topic proposed frame, and so the protocol message delay in the case of a single-topic is less than our proposed one, as shown in Table 4.

In Figure 23, when the network losses are less than or equal to 15%, the average message delay of MQTT and the proposed protocol can be seen to be approximately the same. After 15% of network losses, the introduced message delay by the proposed protocol is greater than MQTT. The difference between the protocols increases with increasing network losses. Here, one single-topic message has a lower frame overhead than one multi-topic message. This is a normal case due to the new frame headers added to achieve the multi-topic functionality. However, the difference is not significant.

Table 4. MQTT versus proposed protocol average end-to-end delay in seconds for different rates of losses (%).

	0% Loss	**10% Loss**	**20% Loss**	**30% Loss**
MQTT	0.001081	0.065235	0.695212	2.357077
Proposed	0.001095	0.071645	0.721798	2.505362

6.2. Message Data Transfer

We found that the message data transferred per message should be considered as an important metric; thus, the traffic generated by the network should be as small as possible. The message data transferred can be calculated as the ratio of the total generated bytes to the number of successful messages delivered, as shown in Equations (2) and (3). We depend on the Wireshark tool output to calculate this metric at different percentages of packet loss rates.

$$\overline{L} = \frac{1}{n}\sum_{i=1}^{n}(\overline{L_i}) \tag{2}$$

where $\overline{L}$ is the number of bytes on average per successfully transmitted message, $\overline{L_i}$ is the number of bytes on average per successfully transmitted message for each trial and n is the number of trials.

$$\overline{L_i} = \frac{L_i}{M} \tag{3}$$

where $\overline{L_i}$ is the number of bytes on average per successfully transmitted message for each trial, Li is the total traffic or number of bytes per trial and M is the number of successful messages that replied with an ACK.

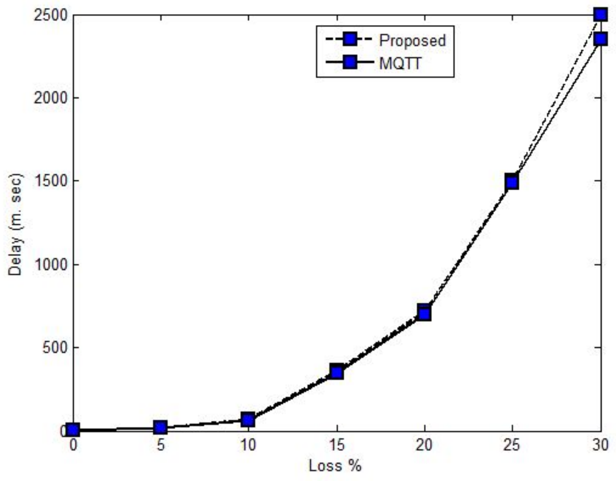

Figure 23. The average message delay (in seconds).

In Figure 24, the average message bytes of MQTT and the proposed protocol are close when the network losses are less than or equal to 10%. After that, the message bytes of the proposed protocol are greater than MQTT. When the network losses increase, the difference between the protocols increases. Here, a lower protocol overhead is introduced by a single-topic message, but a small increase is found for the multi-topic message. The new frame headers that are added to achieve the multi-topic functionality are the reason for this behavior.

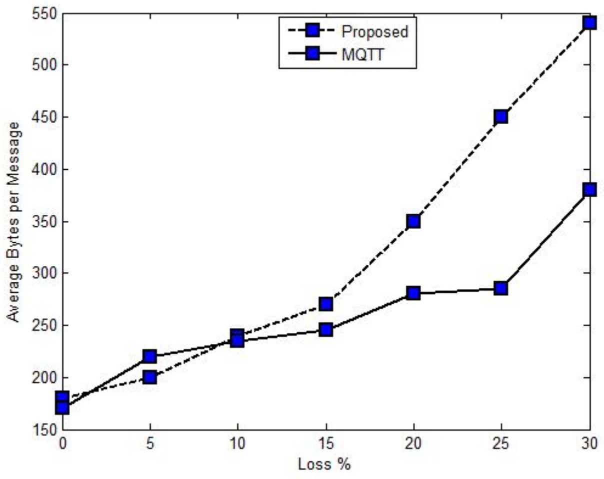

Figure 24. Average traffic per message (bytes).

6.3. Multi-Topic Messages

In many cases, we need to publish multi-topics to many clients or nodes at the same time . For this use case, the MQTT protocol cannot support these features. Thus, in MQTT, we need to publish each topic in a separate message; however, in our multi-topic protocol, which supports multi-topic features, we can publish these different topics in a single message. In this case, the proposed protocol delay is smaller than for MQTT, as shown in Figure 25, and the overhead bytes for each multi-topic message are shown in Figure 26 for no losses.

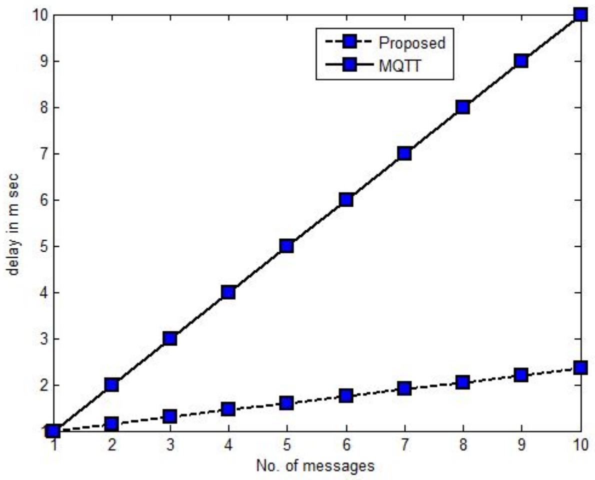

Figure 25. The multi-topic measured message delay (in milliseconds).

In Figures 25 and 26, when the number of the transmitted messages is equal to one, MQTT and the proposed protocol have approximately the same message delay and message size. However, when the number of transmitted messages is equal to two, the introduced message delay and message size from the MQTT are larger than the proposed approach. This difference increases as the number of transmitted messages increases. These observations and results arise because sending multiple single-topic messages adds more overhead than sending one multi-topic message, as MQTT requires a number of messages to be sent that equal the number of topics (one message per topic). On the other hand, the proposed protocol sends only one multi-topic message that has a lower overhead and achieves more throughput than the single-topic messages.

To conclude, our proposed multi-topic protocol has a lower delay and traffic compared to the standard MQTT protocol. This fact resulted from the addition of the multi-topic feature to the proposed protocol, adding some bytes for more topics as opposed to the approach in the MQTT protocol.

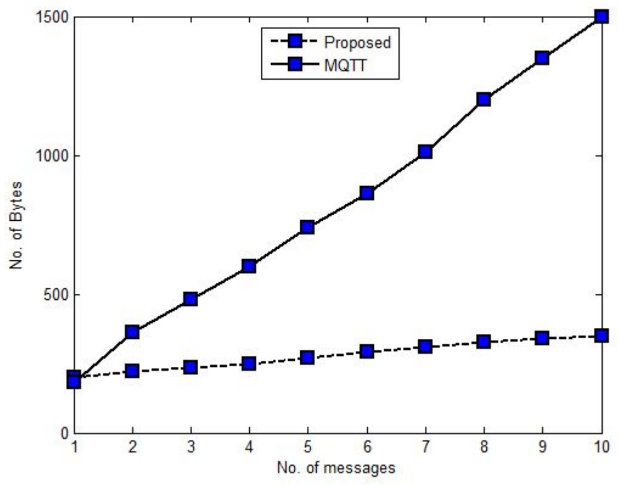

Figure 26. The total traffic measured per multi-topic message in bytes.

6.4. Worst-Case Scenario

Many current IoT applications depend on real-time systems. Real-time implies the enhancement of two parts of IoT communication systems: the first part describes the communication between the IoT node and the connected sensors or actuators (the proposed IoT client based on ARM Cortex-M4), and the second part involves the communication between the IoT client and the IoT broker. In our research, we addressed the two parts as follows.

- The proposed embedded IoT client is handled by using FreeRTOS (a common real-time operating system on the market). It provides multi-tasking to guarantee that many tasks work correctly in a semi-concurrent way and handle deadlines. Thus, the proposed IoT client is a real-time embedded system. Moreover, using RTOS in the proposed IoT embedded unit can decrease the processing time and minimize the delay.
- The second part (which involves the communication between the client and the server) is handled in our research by using the multi-topic feature to enhance the delay required for publishing many messages. Overall, the proposed system has a lower delay than similar systems due to the use of FreeRTOS and the multi-topic feature, as shown in Table 5.

Table 5 shows that the multi-topic messaging in the proposed protocol is better than MQTT for more than one topic (two topics or more), and the single-topic messaging in MQTT is near to the proposed protocol.

Table 5. Comparison for the worst-case scenario.

Worst-Case Scenario / Network Losses	0%	5%	10%	15%	20%	25%	30%
One message vs single-topic (MQTT)	1.14	16	85	375	1253	3153	5894
One message vs single-topic (Proposed)	1.21	22	108	627	1342	3245	6361
Two messages vs two-topic message (MQTT)	2.24	31	165	725	2452	6253	10,194
Two messages vs two-topic message (Proposed)	1.28	25	118	687	1388	3275	6450
Five messages vs five-topic message (MQTT)	5.4	75	405	1850	6253	12,153	21,894
Five messages vs five-topic message (Proposed)	1.31	26	122	714	1391	3315	6550

From this result, and according to the condition of meeting a 5 second deadline, we can state the following:

- In the case of a single topic, the proposed protocol, as well as standard MQTT, can transmit the data even if the loss percentage in the communication network reaches 25%; however, if loss reaches 30%, neither method can transmit the data within 5 s.
- In the case of two messages vs a single message with two topics, the proposed protocol can transmit the data even if the loss percentage in the communication network reaches 25%, while standard MQTT can only tolerate 20%.
- In the case of five messages vs a single message with five-topics, the proposed protocol still can transmit the data even if the loss percentage in the communication network reaches 25%, while standard MQTT can only tolerate 15%.

Thus, the proposed protocol is better and more able to work in worst-case conditions than standard MQTT.

7. Protocol Characteristics

This section will highlight the main implementation characteristics of our proposed protocol compared to the MQTT standard.

7.1. Main Characteristics

The proposed protocol implementation characteristics can be summarized as follows;

- Simplicity: This is an important feature for any protocol or system and can be summarized as follows: it must be easy to learn—and easy to use—the protocol or the system.
- Low protocol overhead: The protocol overhead is the number of bytes required for each designed frame. The frame structure shows that our protocol has a small overhead: i.e., the start of the frame byte and the frame type byte, where the frame elements are just comma-separated elements. The protocol has four main types of frames which have low overheads (few bytes): the identification frame has 6 bytes, the acknowledgement frame has 5 bytes, the registration frame has 8 bytes and the data frame has 9 bytes.
- Constrained network capability: The network challenges increase in the IoT systems, as billions of nodes require a network connection that maintains its data from corruption or being lost. In such networks, the network bandwidth should be limited for each node, as the network capacity might be extended; our protocol requires a small frame size for all the frame types.
- Constrained devices availability: The huge number of devices is a reason for minimizing the total cost for each device; choosing controllers with limited resources reduces the total cost of the IoT device. This has limited the use of such systems to date, although more powerful processors have been developed. The proposed protocol consists of four simple frames with a small number of bytes for each frame. The design simplicity and the low protocol overhead make the protocol able to work with the constrained devices which have controllers with an 8-bit processor architecture and limited resources.

- Asynchronous communication model: An IoT node receives its required data once the data are available; this system is called an event-oriented system, which is not polled in a periodic way to detect the data. The protocol enables the IoT nodes to use this feature, meaning that they are always up-to-date with data from the source.
- TCP-Oriented protocol: The TCP is the industry standard; it is considered to be the language of the Internet. TCP is a reliable protocol because the protocol itself checks everything that was transmitted if it was delivered at the receiving end correctly or not; if not it will re-transmit the data. Our protocol depends on a TCP connection.
- Publish/Subscribe model: The pub/sub messaging protocols allow messaging with a topic name, which is an array of characters; the node which sends the message is called a publisher and the node which receives the message is called a subscriber. This model or architecture has many features such as message decoupling and one-to-many messaging, which is discussed below. The model of the proposed protocol is similar to this model.
- Data decoupling and one-to-many messaging: The data decoupling feature is one of the features of pub/sub systems shared by the proposed protocol. This means that more than one node can publish or send messages on the same topic or parameter name. The senders will send data to the broker server with a certain topic or parameter name; the server will send these messages to the nodes which register the same parameter name, as in Figure 27.

  One-to-many messaging is the second feature of the pub/sub systems shared by the proposed protocol, which means that a node can send the same message to many nodes at once. The sender node will send a message with a specific parameter name to the main server, and the server will send the same message to all the registered nodes with the same parameter name, as shown in Figure 28.

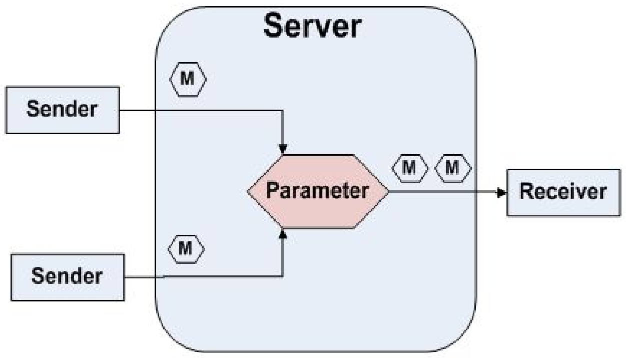

Figure 27. Data decoupling.

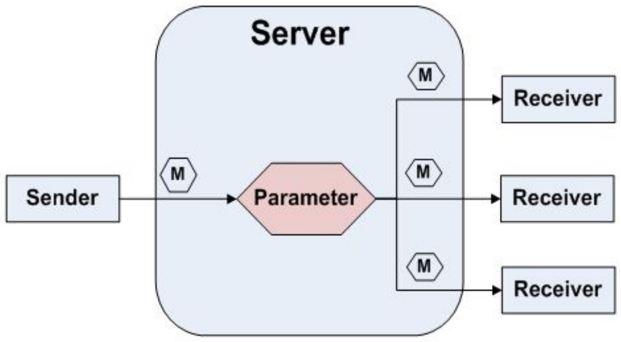

Figure 28. One-to-many messaging.

- Multi-functional communication: Any IoT node can perform both message decoupling and one-to-many messaging. It can send a message with any parameter to the server, and the server will automatically flood the message to all the registered nodes to that parameter. A node is capable of registering to any different parameters, and when the server receives any messages for these parameters, it will forward them to the node directly; both operations are shown in Figure 29.

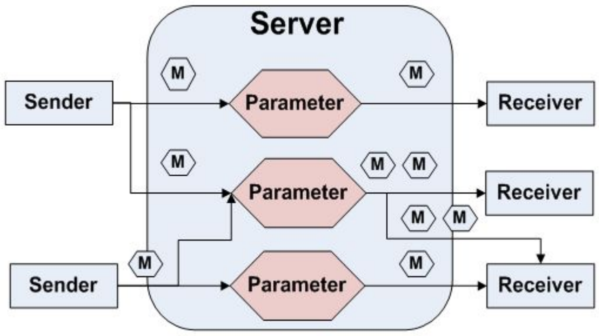

Figure 29. Multi-functional communication.

- Reliability: The proposed protocol is based on the TCP protocol, which adds a sort of physical layer of reliability. One of the protocol frames is called the acknowledgment frame; this frame is required by the receiving node to acknowledge the reception if it succeeded—the server also sends this frame to acknowledge the message received from any IoT node. Moreover, the identification frame should be replied with an ACK; the registration and data frames have the choice of enabling the ACK to be expected from the receiver or not.
- Scalability: The simplicity of the proposed multi-topic communication protocol tends to lead to a simpler server algorithm. The server main thread starts the main server algorithm and listens for the new IoT node connection when a new node tries to connect to the server. Thus, the scalability is not limited by the protocol itself as it consumes a few bytes and a small number of frames.

7.2. Comparison between Standard MQTT and Our Proposed Protocol

Our protocol has many features supported by MQTT protocol, such as a TCP-oriented protocol, message delivery mechanism and a publish/subscribe architecture. The protocols are different in some features, such as the multi-topic feature, which is supported only by the proposed protocol and not the MQTT, and the frame field-determining mechanism. A brief comparison between the standard MQTT and the proposed IoT multi-topics protocol is shown in Table 6.

Table 6. The proposed protocol and MQTT comparison. QoS: quality of service.

	The Proposed Protocol	**MQTT Protocol**
Transport layer	TCP-oriented	TCP-oriented
Message delivery mechanism	Simple ACK mechanism	QoS mechanism
Architecture	Publish–subscribe architecture	Publish–subscribe architecture
Multi-topic support	Yes	No
Frame field-determining mechanism	Field-delimiters mechanism	Field-size mechanism

In our evaluation, the average case was calculated for the end-to-end messaging for a fair comparison between the two messaging protocols (the MQTT protocol and the proposed one) with different network losses. It is worthy of note that an embedded system is not simply a real-time system that must take care of the worst-case deadlines within the device itself, but the end-to-end messaging is a network evaluation test that must be tested with many message transmissions. This evaluation is done for many trials that must be calculated on average for a successful comparison between the two messaging protocols.

8. Conclusions

In this research, the definition and architecture of the proposed IoT multi-topic protocol are presented. Additionally, the main protocol feature, which is called the multi-topic feature, and the application range that could be applicable for that protocol are discussed. The multi-topic IoT communication sequence diagrams are shown, and the protocol frame formats are designed in this research. The characteristics of the proposed protocol are also listed. Furthermore, the general IoT architecture based on RTOS is presented, and a new multi-topic IoT platform is proposed to implement the IoT nodes. The node–server communication is introduced and the experiment setup is shown; finally, the experimental results are discussed. Experiments were carried out to study and compare the performance of our IoT multi-topic protocol versus the standard MQTT using different parameters of the real network which affect the performance of protocols. Practical experiments were executed in a real environment based on our NTI network infrastructure. The obtained results showed that our protocol had less delay and lower traffic for multi-topics compared to the MQTT. Therefore, the proposed protocol outperformed the MQTT protocol in the case of multi-topic messaging. Moreover, our protocol was better than the batching of multiple messages for real-time system applications.

Author Contributions: All authors have equally participated in contributions to this research article, either in terms of the concept, methodology or validation phases of the proposed protocol. Moreover, all authors have read and agreed to the submitted version of the manuscript. All authors have read and agreed to the published version of the manuscript.

Funding: This research received no external funding.

Conflicts of Interest: The authors declare no conflict of interest.

References

1. Hussein, M.; Zorkany, M.; Abdelkader, N. Real Time Operating Systems for the Internet of Things, Vision, Architecture and Research Directions. In Proceedings of the IEEE Conference Publications, WSCAR, Cairo, Egypt, 12–14 March 2016; pp. 72–77.
2. Zhang, M.; Zhao, H.; Zheng, R.; Wu, Q.; Wei, W. Cognitive internet of things: Concepts and application example. *Int. J. Comput. Sci. Issues (IJCSI)* **2012**, *9*, 151.
3. Zhang, Y.; Ma, X.; Zhang, J.; Hossain, M.; Muhammad, G.; Amin, S. Edge Intelligence in the Cognitive Internet of Things: Improving Sensitivity and Interactivity. *IEEE Netw.* **2019**, *33*, 58–64.
4. Abdurahman, M. A Survey on the Concepts, Trends, Enabling Technologies, Architectures, Challenges and Open Issues in Cognitive IoT Based Smart Environments. *Int. J. Sci. Re. Sci. Eng. Technol. (IJSRSET)* **2018**, *4*, 512–522.
5. Park, J.; Salim, M.; Jo, J.; Sicato, J.; Rathore, S.; Park, J. CIoT-Net: A scalable cognitive IoT based smart city network architecture. *Hum. Cent. Comput. Inf. Sci.* **2019**, *9*, 1–20.
6. Ding, G.; Wu, Q.; Zhang, L.; Lin, Y.; Tsiftsis, T.; Yao, Y. An Amateur Drone Surveillance System Based on Cognitive Internet of Things. *IEEE Commun. Mag.* **2018**, *56*, 29–35.
7. Gad, R.; Talha, M.; Abd El-Latif, A.; Zorkany, M.; El-Sayed, A.; EL-Fishawy, N.; Muhammad, G. Iris Recognition Using Multi-Algorithmic Approaches for Cognitive Internet of things (CIoT) Framework. *Future Gener. Comput. Syst. (FGCS)* **2018**, *89*, 178–191.
8. Thangavel, D.; Ma, X.; Valera, A.; Tan, H.; Tan, C. Performance Evaluation of MQTT and CoAP via a Common Middleware. In Proceedings of the 2014 IEEE Ninth International Conference on Intelligent Sensors, Sensor Networks and Information Processing (ISSNIP), Singapore, 21–24 April 2014; pp. 1–6.
9. Hussein, M.; Zorkany, M.; Abdel Kader, N. Design and Implementation of IoT Platform for Real Time Systems. In *International Conference on Advanced Machine Learning Technologies and Applications*; Springer: Cham, Switzerland, 2018; pp. 171–180.
10. Hahm, O.; Baccelli, E.; Petersen, H.; Tsiftes, N. Operating Systems for Low-End Devices in the Internet of Things: A Survey. *IEEE Internet Things J.* **2016**, 3, 720–734

11. Labrosse, J. *MicoC/OS-II:The Real-Time Kernel*, CRC Press, 2nd ed.; pp. 1–648. Available online: https://www.amazon.com/MicroC-OS-II-Real-Time-Kernel-ebook/dp/B07CSRL7WY (accessed on 10 February 2020).
12. Zamfi, M.; Florian, V.; Stanciu, A. Towards a Platform for Prototyping IoT Health Monitoring Services. In *International Conference on Exploring Services Science*; Springer International Publishing: Cham, Switzerland, 2016; pp. 522–533.
13. Bellagente, P.; Ferrari, P.; Flammini, A.; Rinaldi, S. Adopting IoT Framework for Energy Management of Smart Building: A Real Test-Case. In Proceedings of the 2015 IEEE 1st International Forum on Research and Technologies for Society and Industry Leveraging a Better Tomorrow (RTSI), Turin, Italy, 16–18 September 2015; pp. 138–143.
14. Haghi, A.; Burney, K.; Kidd, F.; Valiente, L.; Peng, Y. Fast-paced development of a smart campus IoT platform, In Proceedings of the Global Internet of Things Summit (GIoTS), Geneva, Switzerland, 6–9 June 2017; pp. 1–6.
15. Happ, D.; Karowski, N.; Menzel, T.; Handziski, V.; Wolisz, A. Meeting IoT Platform Requirements with Open Pub/Sub Solutions. *Ann. Telecommun.* **2017**, 72, 41–52.
16. Karagiannis, V.; Chatzimisios, P.; Vazquez, F.; and Alonso, J. A Survey on Application Layer Protocols for the Internet of Things. *Trans. IoT Cloud Comput.* **2015**, 3, 11–17.
17. IBM. MQTT V3.1 Protocol Specification, Protocol Standard. pp. 1–81. Available online: http://docs.oasis-open.org/mqtt/mqtt/v3.1.1/os/mqtt-v3.1.1-os.pdf (accessed on 21 June 2020).
18. Shelby, Z.; Hartke, K.; Bormann, C. RFC 7252: The Constrained Application Protocol (CoAP), Internet Engineering Task Force, (IETF) PROPOSED STANDARD. 2014. pp. 1–112. Available online: https://tools.ietf.org/html/rfc7252 (accessed on 21 June 2020).
19. FreeRTOS. Available online: https://www.freertos.org/ (accessed on 21 June 2020).
20. Gaur, P.; and Tahiliani, M. Operating Systems for IoT Devices: A Critical Survey. In Proceedings of the 2015 IEEE Region 10 Symposium, Ahmedabad, India, 13–15 May 2015; pp. 33–36.