HYBRID APPLICATION LAYER PROTOCOL DESIGN FOR IOT ENVIRONMENTS

Erdal ÖZDOĞAN[1] and O.Ayhan ERDEM[2]

[1]Department of Information Systems, Gazi University, Ankara, Turkey
[2] Department of Computer Engineering, Gazi University, Ankara, Turkey

ABSTRACT

One of the important factors affecting communication performance in the Internet of Things is the messaging protocol. MQTT, XMPP and AMQP are centralized application protocols that communicate through the server. DDS and CoAP are application protocols that can communicate directly, especially in real-time applications. As the Internet of Things is becoming more widespread and usage scenarios have different requirements, new approaches to data communication are required. In this study, a UDP based hybrid application layer protocol has been designed which can communicate both directly and through central server. In addition, operating logic and packet structure of the developed hybrid protocol is examined.

KEYWORDS

Internet of Things, IoT Application Protocol, Hybrid IoT Protocol, Lightweight IoT Protocol, UDP Based IoT protocol

1. INTRODUCTION

The unpredictable development of the Internet, from its inception to the present day, shows that developments and innovations in Internet technology will have great repercussions worldwide in the coming years. In 2012, the number of devices connected to the Internet exceeded the number of people living on earth, and by 2020, between 26 and 50 billion objects are expected to be connected to the Internet and are expected to produce millions of gigabytes of data [1]. Organizing, interpreting and transforming this enormous mass of data to be produced in a heterogeneous structure is seen as one of the most important problems that the Internet of Things (IoT) will face [2]. The idea that every device or object in the near future will have an IP address, either directly or indirectly, reveals the need to develop protocols that support IP [3] [4]. The inadequacy of the existing Internet protocols to meet this aim leads to the development of new protocols. For this purpose, in order to provide data transfer on IoT, Message Queuing Telemetry Transport (MQTT), Constrained Application Protocol (CoAP), Advanced Message Queuing Protocol (AMQP), Data Distribution Service (DDS), and Extensible Messaging and Presence Protocol (XMPP) have been developed or adapted to IoT requirements [5]. On the other hand, the report "M2M service layer: APIs and protocols overview", published by ITU-T in 2014, states that new protocols should be developed in order to meet developing technologies and different usage scenarios [6].

MQTT, AMQP and XMPP protocols are central protocols that use servers for data transfer [7]. In this approach, there is no direct communication between the client (or user) and the IoT Device. Protocols such as CoAP and DDS are direct communication protocols that work in decentralized approach [8],[9]. Both centralized and the decentralized approaches have several advantages and disadvantages. One of the most important advantages of the decentralized approach is its ability to communicate directly [10]. However, administrative difficulties are themain disadvantages of this architecture [11]. One of the most important advantages of central communication is that the communication can be controlled by the server. However, the use of the server causes a single point of failure [8], [12] - [14] [15].

In this study, a hybrid application layer protocol (hIoT) is designed for data communication which provides both server-based and direct communication. The designed protocol has been developed in order to transfer low-dimensional data generated by sensors in devices with limited resources and in low bandwidth environments. In the developed protocol, both communication methods can be used for different purposes and needs. In addition, dynamic switching can be made between these two methods.

In the second part of the article, academic studies on IoT protocols are discussed. In the third section, the general working principle of the designed application protocol, packet structures, and protocol components are examined. In the last section of the article, a summary of the study and the advantages of the proposed protocol are given and suggestions are made to shed light on future studies.

2. RELATED WORK

Protocols play an important role in the realization of IoT. Research on existing IoT protocols shows that these protocols can be superior in different areas when compared with each other. In this section, studies on the performance comparisons in terms of bandwidth, latency, and interoperability of IoT application protocols are discussed.

In the study of Thangavel et al. [16], where the middleware layer was developed in order to enable MQTT and CoAP protocols to work together, MQTT and CoAP protocols were discussed in terms of end-to-end latency and bandwidth consumption. According to this study, MQTT is more successful in environments with packet losses of less than 20%; however, CoAP is more successful in higher packet losses. In a study [17] comparing protocols according to their payload, it was reported that the CoAP protocol experienced a loss of performance at payload capacities greater than 1024 bytes. In the study conducted in mobile and unstable networks [18], MQTT and AMQP protocols were compared in terms of bandwidth usage, delay, and jitter effect. There was no significant difference between these protocols. MQTT is more successful in terms of energy efficiency. In another study comparing CoAP and MQTT protocol in transporting the same data [19], CoAP protocol was stated to be more efficient. In a study conducted in an environment with high network traffic, the MQTT protocol was reported to be more successful than CoAP, with a higher bandwith and lower latency [20].

In a study addressing the problem of service discovery in the IoT, M. Kirsche et al. stated that MQTT and CoAP protocols could not provide an end-to-end communication, delays occurred due to message conversion, and the discovery process was complex. They have therefore developed a simplified structure using the combination of mDNS and DNS-SD [21] [22].

In a study that can be examined under the title of Developing IoT Protocols [23], it is mentioned that MQTT and CoAP protocols are widely used in IoT, but both protocols face scalability problems in large networks with multiple sensors. In the study, in order to ensure scalability and use of bandwidth efficiently, CoAP protocol was improved. Accordingly, it was ensured that the sensors were grouped to form a cluster and a representative sensor sent the data, which resulted in 18% less bandwidth consumption.

The complexity of the IoT system and the increase in the number and diversity of devices connected to the network lead to the use of hybrid methods as solutions. In the study of Bellavista et al., the study states that this architecture developed in high density networks provides scalability [8].

The study [24] conducted to ensure that multiple protocols work together using a central middleware layer reveals the interoperability of protocols without significant performance losses. In the design of an application protocol called "Custom UDP", the proposed protocol was compared with various IoT protocols and according to experimental results, it is stated that it offers relatively low energy and bandwidth consumption in environments with unpredictable packet losses [25].

As it is seen, it is very difficult to state that only one protocol is superior in every aspect of the IoT. Different protocols can be used according to the network topology used, security need, scalability, and bandwidth usage. In addition, the diversity of middleware software makes it difficult to connect to IoT devices and interpret the collected data [26]. The heterogeneous nature of the IoT ecosystem brings along the need for different protocols with an ever increasing trend of growth.

3. Developed Application Layer Protocol

Within the scope of the IoT, various research and academic studies have been carried out in recent years on problem situations such as security, resource discovery, interoperability, compatibility, and performance improvements. This study does not focus on all of these problem situations and some assumptions are taken into consideration. Accordingly, although the concept of security is critical for IoT, this study assumes that the environment is safe and data security is out of scope. The location and service information of the sensor was taken into account in order to comply with MQTT and CoAP protocols during the registration and resource discovery process. Parameters such as sensor manufacturer and sensor type have not been taken into consideration. The location of sensor, and the service provided by the sensor is designed as shown in Figure 1 to provide compatibility with the "topic" used in the MQTT.

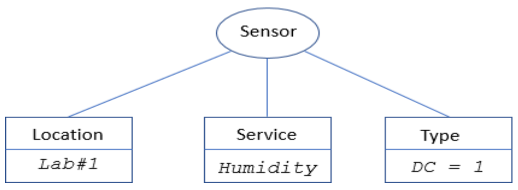

Figure 1. Sensor properties used in the developed protocol

In this study, it is assumed that the data obtained from the sensors can be carried in a single data packet and the IP packets are not fragmented.

Since the devices used in the IoT system have limited resources, applications requiring high processing power, memory and bandwidth consumption are inadequate and inefficient in meeting the need. Therefore, the developed protocol is designed to be flexible and simple to operate at low resources.

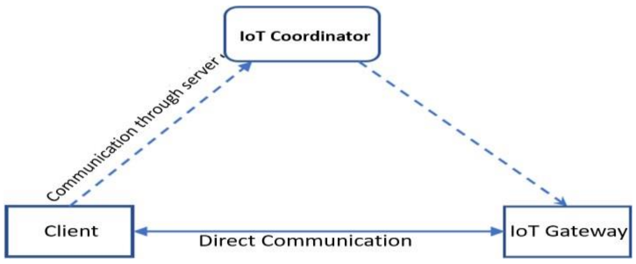

Figure 2. Hybrid Application Protocol supporting direct communication and server-based communication

Within the scope of the study, a Hybrid Application Layer Protocol (hIoT) was developed, as shown in Figure 2, which works with both the central server and supports direct access. Depending on the sensor type, either direct communication or communication through server can be selected. When the server is failed, a dynamic switching mechanism is designed that can be switched to direct communication. Thus, it is aimed to prevent a single point of failure in server-based communications.

3.1 General Operating Principle

In order for the designed system to function correctly, the roles of devices in the IoT ecosystem must be defined precisely.

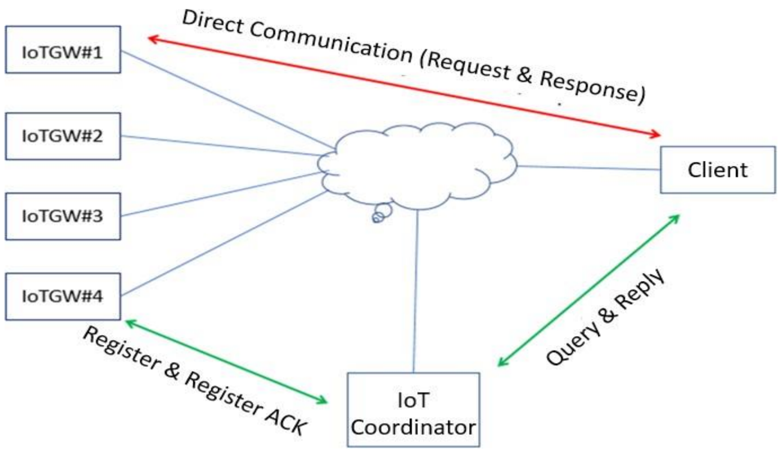

Figure 3. Devices and communication packets in the developed protocol

In the developed protocol, as shown in Figure 3, three different components are defined: IoT Coordinator, IoT Gateway (IoTGW), and Client.

The operation of the protocol is examined in three phases: Service Registration Phase, Service Query Phase, and Data Transfer Phase, as shown in Figure 4.

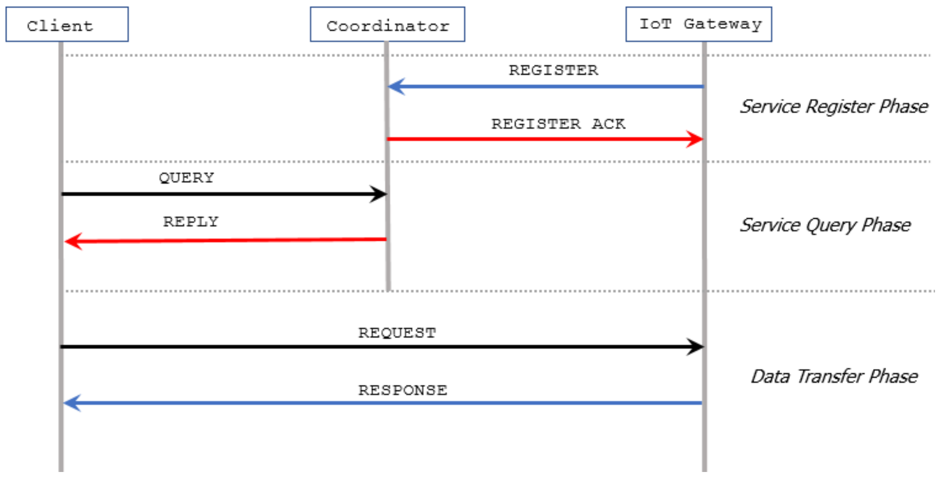

Figure 4. Basic phases of the developed protocol

The service register phase is the registration of the services running on the IoT Gateway to the database. During the service query phase, clients are asked how to access the services they are interested in. Information about how to access the service by the coordinator is sent to the client in response. In the data transfer phase, direct communication between the client and the gateway or communication through the server is performed, depending on the access information given by the coordinator.

3.2 Packet Types

The designed IoT protocol includes eight different packets. Packet types and their descriptions are shown in Table 1.

Table 1. Packet Types and Descriptions

Packet Type	**Description**
Control	Used for accessibility control.
Reset	Resets the access method.
Register	Enables the service provided by IoTGW to be registered to the server.
Error	The packet sent in case of error.
Query	The query packet that the client use makes for service discovery.
Reply	Sent in response to the service discovery packet.
Request	A request for information from the service provided by IoTGW.
Response	A packet containing data that is sent in response to the request packet.

3.3 Packet Structure and Headers

The header of the developed protocol is designed in a structure consisting of the areas shown in Figure 5 for ease of use.

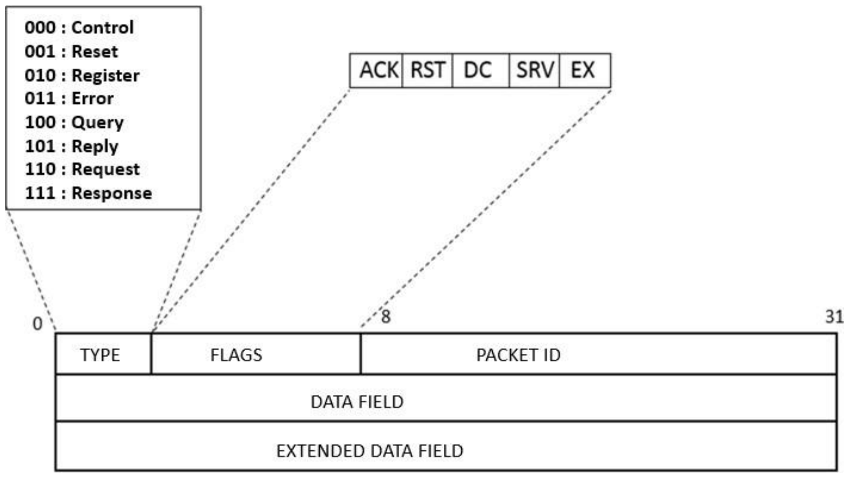

Figure 5. Packet header structure of Hybrid IoT Protocol

The 3-bit “Type” field in the header information shows the packet type. The 5-bit “Flags” field stores flags that are necessary for the proper functioning of communication. The 24-bit “Packet ID” field contains the unique ID of the packet.

The “Data Field” is a 32-byte long field that allows data from sensors to be transported in applications. “Extended Data Field” refers to an additional data transport area of 1024 bytes, which has been developed to support larger data transfer and operates by the “EX” flag.

The ACK flag is used for confirmation packets in communication. The RST flag is used to reset the direct access authorization through the IoT Gateway, which is used with the Reset packet. IoT Gateway, which receives the packet with the RST flag set, will lose direct communication. Thus, the coordinator server will be used as a tool to get information from the service. If the DC flag is “1”, direct communication to the relevant service can be provided. The SRV flag indicates server-generated traffic. The last flag, EX, indicates whether to use the Extended Data Field.

3.4 Working Phases of the Protocol

In this section, the phases of the developed protocol will be discussed in detail. Each step will be illustrated with packet structures and sample content.

3.4.1 Service Registration Phase

The purpose of this phase is to create a local database where services provided by IoT Gateways and access information to these services are kept. In cases where the number of sensors that need to be registered to the system is small, manual recording method or semi-automatic recording system can be used. However, in cases where this number is much higher, the manual registration method is inefficient, expensive and will require additional effort; in most cases it will be impossible to implement [27]. For this reason, a mechanism for semi-automatic recording of the services provided by sensors connected to IoT Gateway has been developed. Services such as temperature, light intensity, humidity provided by the sensor are combined with location information and recorded in the local database of IoT Coordinator.

The automatic registration of the sensor connected to the IoT Gateway to the local Coordinator device takes place at this phase. At this phase, the service provided by the IoT Gateway device, the IP address, whether it supports direct communication (DC) information, and the caching time is sent to the IoT Coordinator.

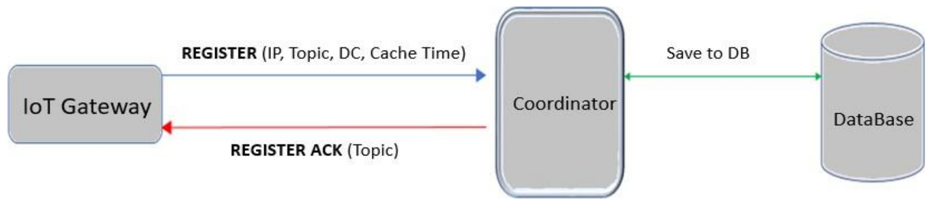

Figure 6. Service registration process of the developed protocol

For the “temperature” service provided in Lab1 location, the Register process is shown in Figure 6.

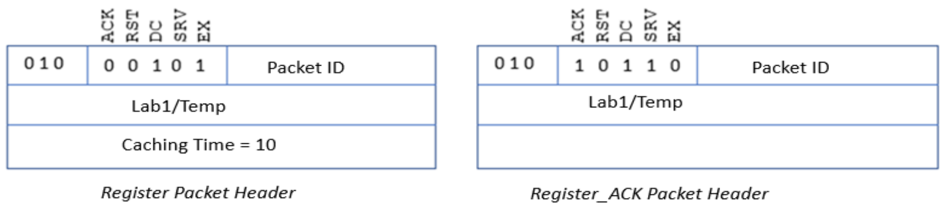

Figure 7. Packet headers of Register and Register_ACK

According to the example in Figure 7, the “Lab1/Temperature” service supports direct communication and the information sent by this service is kept in the cache of the coordinator server for 10 seconds. During this period, all data requests for the same service are retrieved directly from the coordinator cache without being interrogated to the sensor node again.

The server that receives the topic, cache time, DC information and the IP address of the IoT Gateway is saved to the local database. Following the registration, the registration confirmation packet (Register_ACK) is sent by the coordinator to the IoT Gateway.

During the service registration process of the developed protocol, all information is kept in the database of the Coordinator. Since this causes a single point of failure, service discovery queries will not be answered if the coordinator is failed. In order to prevent this problem, Automatic Registration feature has been developed for IoT Gateway devices. In the case of the server failure, the IoT Gateway sets itself to respond directly to service discovery requests if the register packet is not answered within a specified time period.

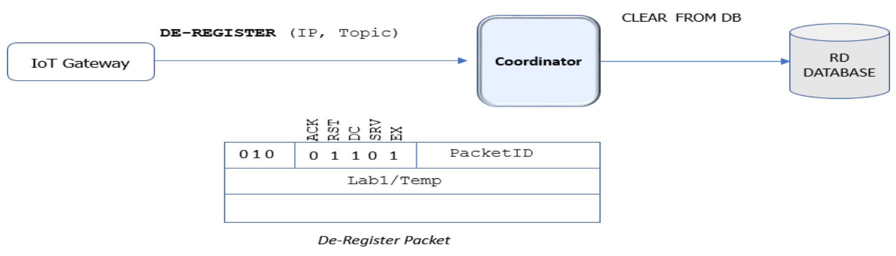

Figure 8. De-Register process and packet structure

If the service provided by IoT Gateway is disabled, registration must be canceled. In this case, the gateway sends a special Register packet with the RST bit set for the respective service. The Coordinator receiving the packet deletes the access information for this service from the database and responds to the IoT Gateway with the Register_Ack packet. The example in Figure 8 demonstrate the de-register packet for the “Lab1 / Temp” service.

3.4.2 Service Discovery Phase

To fully exploit the potential for successful implementation of IoT, there is a need for seamless and automatic discovery of available resources and dynamic access to the IoT device. MQTT does not have service discovery, but the CoAP protocol uses Uniform Resource Identifiers (URIs) for service discovery [28]. In the proposed protocol, the client queries the “topic” information for the service that it wants to access with “Query” packets. In the communication between the client and the server, the query regarding the location and service needed by the client is shown in Figure 9.

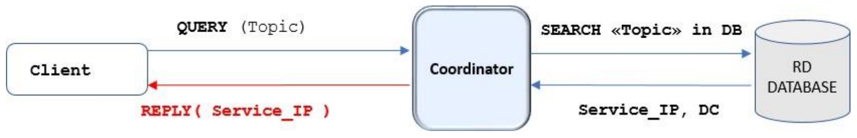

Figure 9. Service discovery process

The data related to the topic queried by the client is searched in the coordinator database, then the IP address of service notified to the client. If the relevant service supports direct communication, the IP address of the IoT Gateway is sent as a reply. Otherwise, the IP address of the Coordinator is sent.

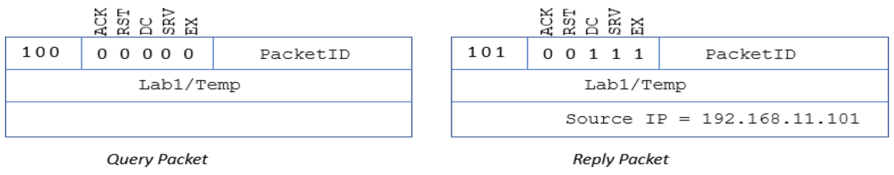

Figure 10. Query and reply packet headers

The example in Figure 10 shows the packet header of the “Query” sent by the client and the “Reply” packets given by the Coordinator in response.

3.4.3 Data Transfer Phase

At this phase, data request is made according to the “topic” information. Due to the hybrid nature of the proposed protocol, the data request can be provided directly from the Gateway, the sensor node (Figure 11), or via the Coordinator server (Figure 12).

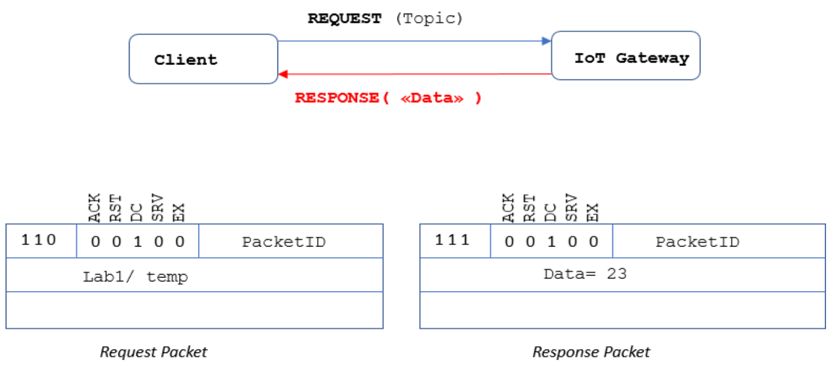

Figure 11. Process of direct access to the sensor node and packet headers.

During the Service Registration phase determines which methods of communication, either direct or through server, is supported. The packet structure and sample content for direct IoTGW access is given in Figure 11.

The Request message sent by the Client is given feedback via the Response packet that contains the data for the service being queried.

Modeling of server-based communication, another method supported by the hybrid protocol, is shown in Figure 12.

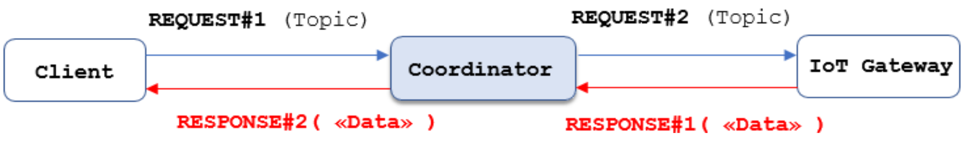

Figure 12. Data transfer process through server

In server-based communication, the client is not allowed to access the IoT Gateway directly.

Therefore, the data request from the service is made through the coordinator. The coordinator receives the Request packet from the client according to the "topic" and sends it to the IoT Gateway. The Response packet from IoT Gateway is sent to the client via the coordinator. In server-based communication, the sensor node only considers packets with the flag "SRV" marked "0" in the packet header.

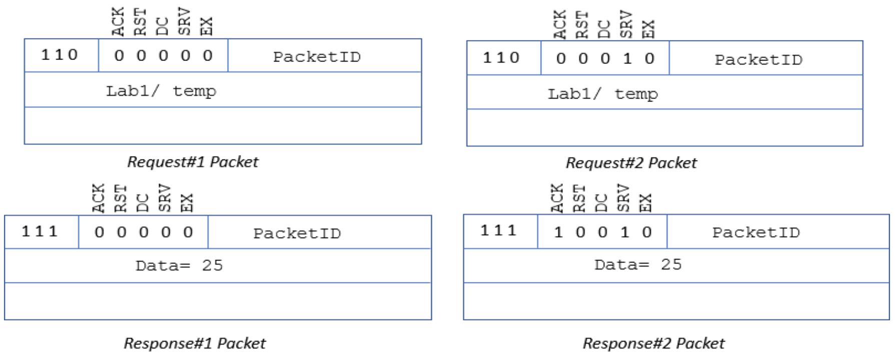

Figure 13. Request and response packets used in server-based communication

The header information and sample content of the packets used in server-based communications are given in Figure 13. When communicating through the coordinator server, the client has to connect to the server. All communication related to this service takes place through the coordinator. How long the data will be valid according to the cache time is determined during the service registration process. The coordinator stores data for the duration of the cache. During this period, other Request messages for the same service are answered directly from the cache without being sent to the sensor node.

3.5 Utility Packets

In addition to the five packets that offer the main functionality, Error, Reset and Control packet types are used to increase functionality.

Error Packet: It is sent to the client in case of any error in the local source database on the server, for instance, if there is no record of the subject queried, no access to the service, and no recordings are made to the database. The "error codes" for each error are also sent in the packet. An example of the error packet and the packet header structure is given in Figure 14.

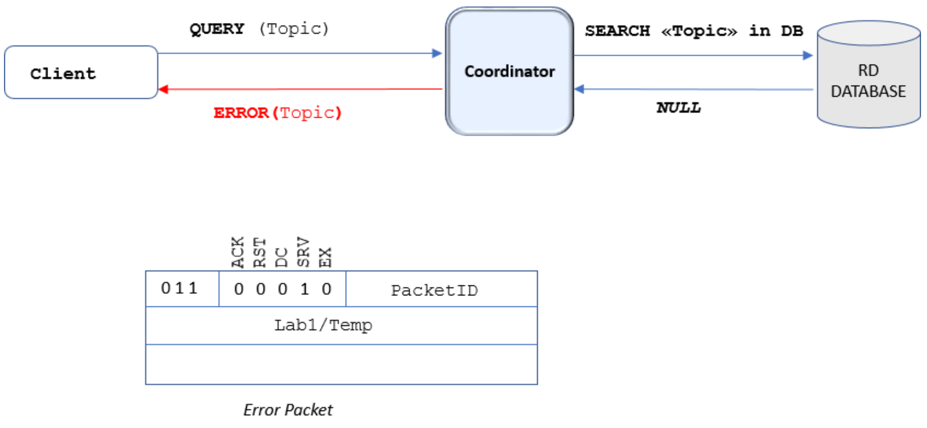

Figure 14. Error process and packet structure

Reset Packet : The type of access that the IoT Gateway device supports is determined during the Service Registration phase and this value remains constant. However, the direct access method may need to be revoked, depending on the state of the network. In this case, the Reset packet is sent by the coordinator to switch the communication method.

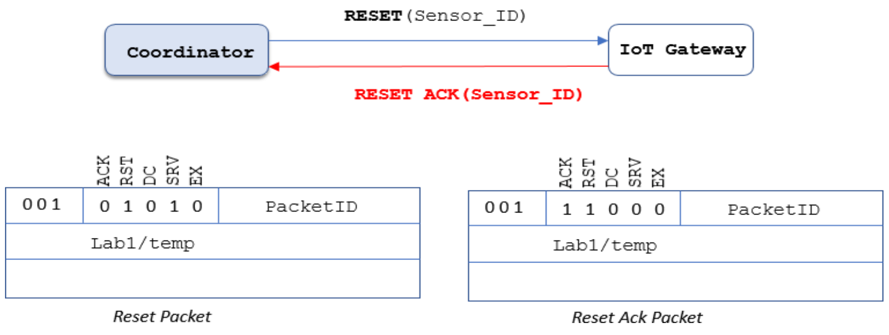

Figure 15. Process of changing (zeroing) the access method of the sensor node and packet structure

The communication method and packet structure of the reset packet is shown in Figure 15. The IoT Gateway that receives this packet loses its direct communication method.

Control Packet: Verification messages are used to check whether the service on the IoT Gateway is accessible. The control packets shown in Figure 16 are replied with control confirmation messages (Control_Ack).

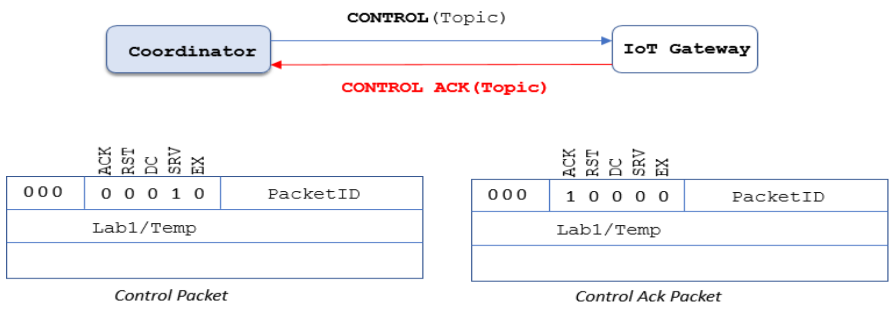

Figure 16. Control packet and sample header structure

Another advantage of the control messages is that the access time to the service can also be calculated by means of the confirmation messages. If no response to the control messages is received, the relevant service is assumed to be disabled and deleted from the coordinator's database. Likewise, when the Control messages do not reach the IoT Gateways, the coordinator is assumed to be failed and the sensor nodes (IoT Gateway) goes direct communication state.

3.6 Components in the Developed Protocol

As mentioned in the previous section, the proposed architecture of the protocol includes three components: Coordinator, Client software and IoT Gateway node. In this section, the working principles of these components are expressed in flow diagrams.

3.6.1 Client

An application that requests data about a location and service on the network and wants to interact with the sensor. It resembles the client that subscribes to any topic in the MQTT and AMQP structure. In the MQTT, the side that initiates the traffic is the publisher, and generally the data of the sensor is sent to the subscriber, regardless of whether the subscriber needs the information at the time. However, in the recommended architecture, the client side initiates traffic. As soon as data is needed, the data request is initiated by the client.

The client makes a query about how to access the service specified by “topic”. The query response from the coordinator contains the access information of the service source. The client matches the "topic" and the access information and stores it in the cache. Then, when the data request is made for this service, it goes directly to the data request stage without the need to query again and waits for the response from the sensor node. Figure 17 shows the flowchart of the client software.

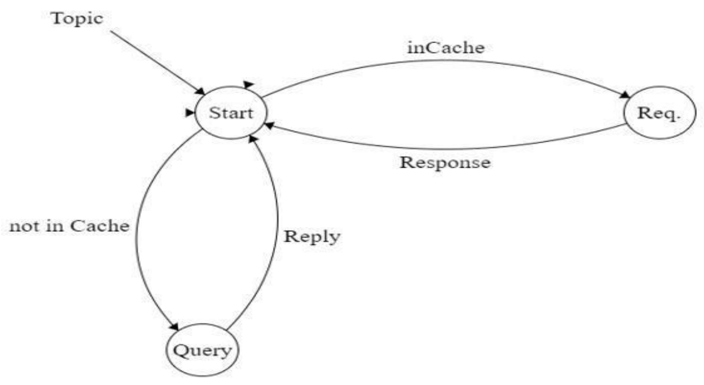

Figure 17. Client software flow diagram

The client software switches to query status or request status, depending on whether the “topic” information requested for data is in the cache and whether or not the Query response packet is received from the coordinator. The “topic” to be queried by the user is primarily searched in the cache. If there is a record of the subject information being queried in the cache, the IP address of the service is retrieved from the cache. Thus, the request status, which is the data transfer phase, is started. However, if there is no record for the requested topic in the cache, the service provider IoT Gateway must query the IP address and enter the query state.

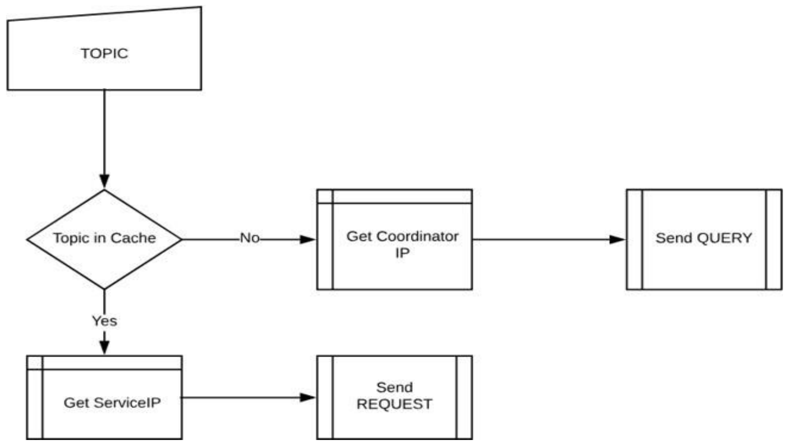

Figure 18. Client software initial state flow diagram

The flowchart for the initial state of the client software is given in Figure 18. The Send Query Packet and Send Request Packet processes in the diagram are presented as separate flow diagrams for modularity.

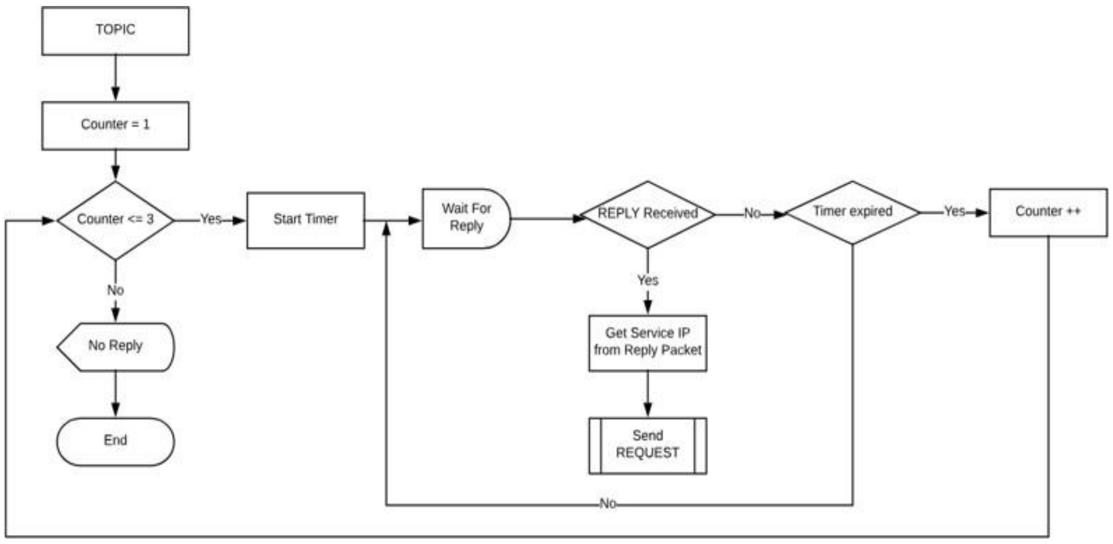

Figure 19. Flow diagram of query sending

The query process according to the "topic" information is performed by the Query packet sent from the client. The flow diagram of this process is shown in Figure 19. The query response message given to the query contains the service IP address. This information is stored in the client's cache for later reuse. As can be seen from the flow diagram, a certain time is expected for the query. If the Query Answer is not received within this period, the process is repeated 2 times. If the query is not received, the software is informed that the reply cannot be received. When the response is received, the client software switches to Request Send status.

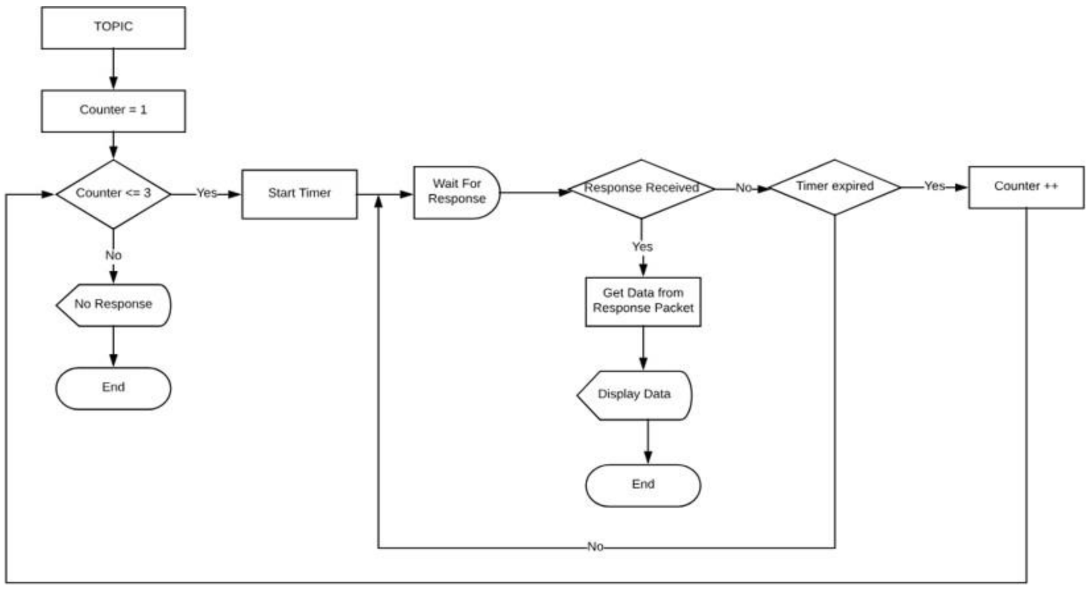

Figure 20. Flow diagram of request submission

On request, information from the IoT Gateway or Coordinator is included in the Response packets. As in the case of query, the request is also repeated 3 times. There is a 2 second dwell time for each operation. When the Response packet is received, data is extracted from the packet and sent to the client software. Otherwise, "no response" message is given to the software. This process is illustrated by the flow diagram in Figure 20.

3.6.2 IoT Coordinator

The most critical task in the proposed protocol is in the coordinator component. This component is responsible for listing the services available in the IoT ecosystem and for mediating users who want to access them.

One of the major problems encountered in the service discovery query on the IoT is the need to define a resource directory that is queried at the local level [22]. In the designed protocol, the coordinator keeps the service resources and how to reach these services in order to meet this need. It directs client traffic to the service source according to service discovery queries from clients. This device not only serves as a resource for service discovery, but also acts as an intermediary for certain types of services defined in the registration process. Accordingly, the flow diagram describing the functions of the coordinator device is shown in Figure 21.

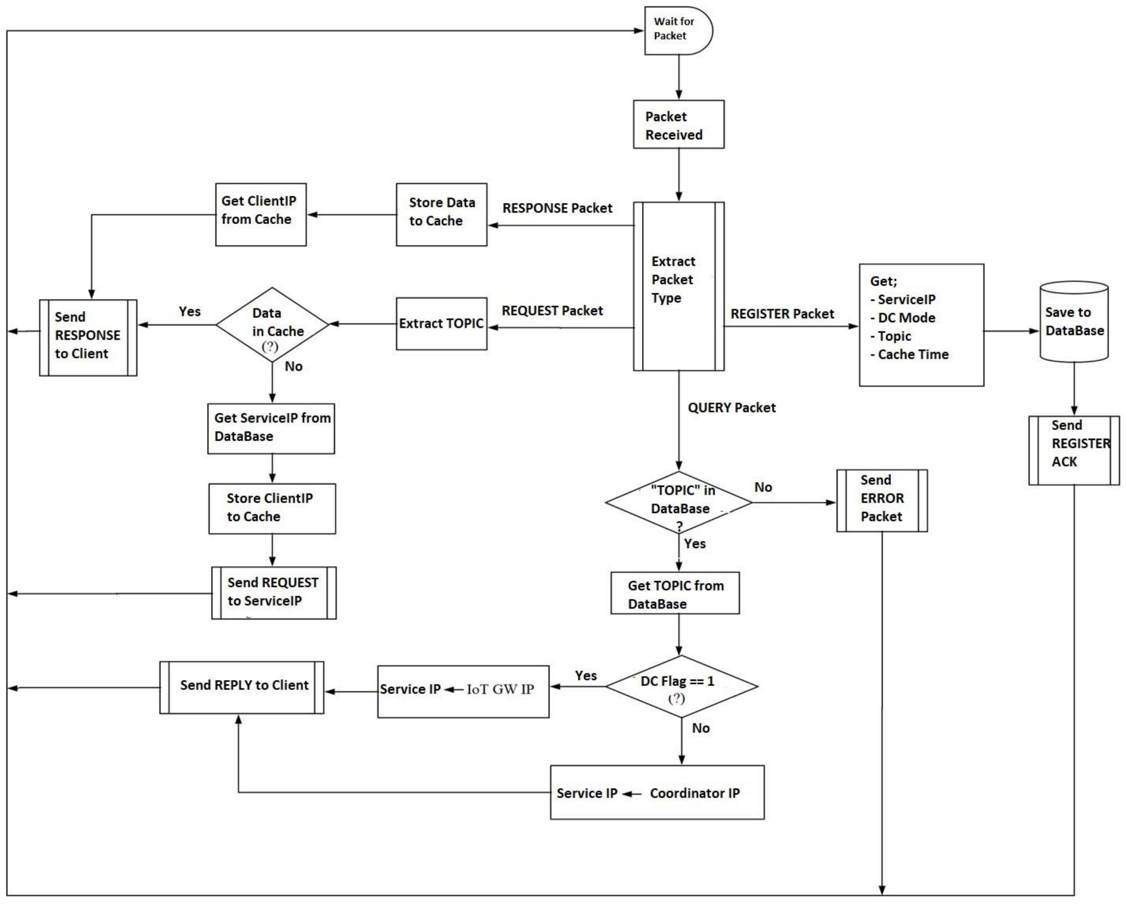

Figure 23. Flow diagram of IoT Coordinator

Identifying the incoming messages to the coordinator and the tasks to be performed according to the packet type are provided by the packet type analysis module. The tasks to be performed for each packet type are shown in the flow diagram in Figure 21.

3.6.3 IoT Gateway

Sensors, which are frequently used in the IoT, are devices with limited or no computational capabilities [21]. Therefore, in order to process the data obtained from the sensors, there is a need for devices called IoT Gateway which can communicate with these devices, send and receive signals, and present the received signals to the network environment. The IoT Gateway concept is one of the critical components in the IoT [29]. This component acts as a proxy between the sensor network and the application layer. Sensors, the digital interfaces of objects in the IoT ecosystem, need these components to communicate with the IP network. In summary, IoT Gateways acts like a sensor node concentrator.

In the developed protocol, IoT Gateway (IoTGW) is responsible for encapsulating the data obtained from its sensors into IP packets and sending them to the client software. IoT Gateway supports both direct communication (DC) and access through the coordinator. Access method supported for each service is provided with the Register packet. . After registration, the client is ready to respond to requests (Request Packet). Figure 22 shows the finite state flow diagram of IoT Gateway.

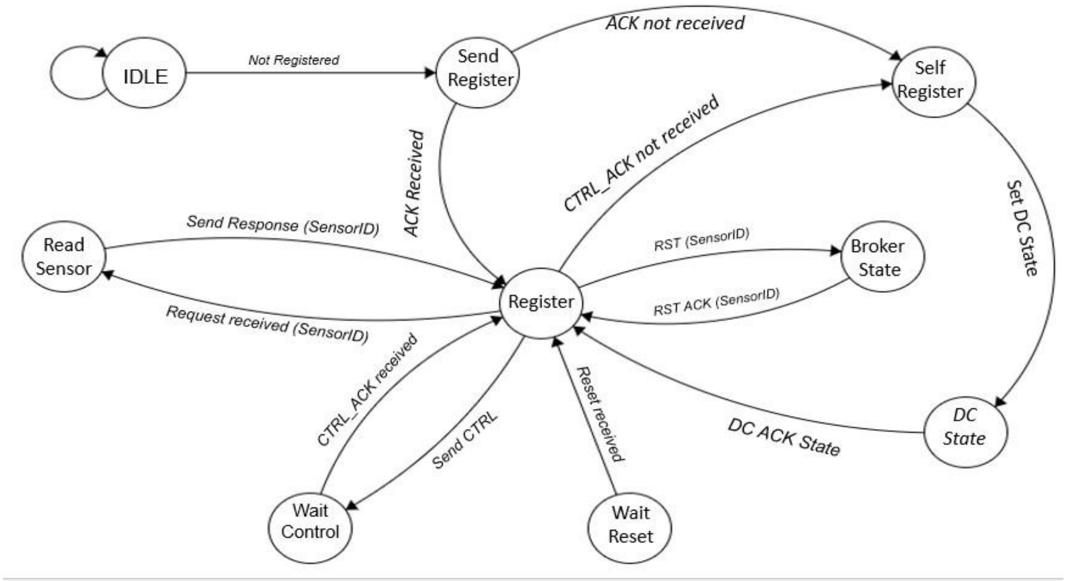

Figure 22. IoT Gateway flow diagram

As indicated in the diagram, it is confirmed by Control messages that the IoT Gateway is in constant communication with the Coordinator. However, the Coordinator creates a single point of failure in server-based communication. Therefore, it must be continuously verified whether the server remains disabled. Periodically sent Control messages check whether the server is accessible. If the server's accessibility is not provided, the IoT Gateway switches to Auto enrollment and supports direct communication (DC = 1). Thus, it can also respond to Query and Request packets that are queried by the client. According to the logic of the hybrid protocol, when the serving IoT Gateway is in direct communication state (DC), it loses direct access authorization from the coordinator and switches to server based operation mode (Server state).

3.7 Experimental Evaluation

For the performance evaluation of the developed protocol, a topology isolated from other network traffic was prepared in the laboratory and compared with the MQTT protocol which is frequently used in IoT applications. The simplified scheme of topology is shown in Figure 23.

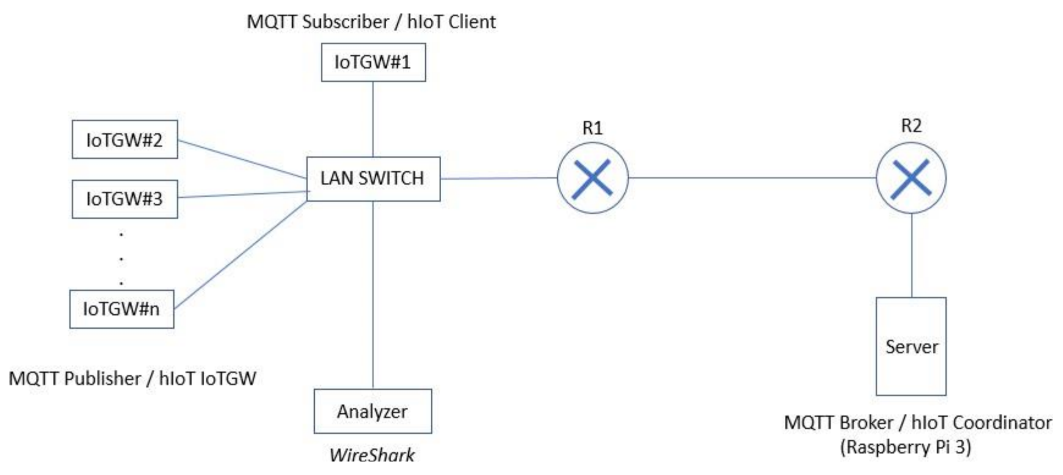
MQTT Subscriber / hIoT Client
IoTGW#1
IoTGW#2
IoTGW#3
IoTGW#n
LAN SWITCH
R1
R2
Server
MQTT Publisher / hIoT IoTGW
Analyzer
WireShark
MQTT Broker / hIoT Coordinator
(Raspberry Pi 3)

Figure 23. Experimental topology developed for performance testing

In the experimental topology, Arduino Unos were used as IoT Gateway devices and Raspberry Pi 3 as the server. The server was used as the "coordinator" in the hIoT protocol, and as the "MQTT Broker" in the MQTT architecture. Open source “Mosquitto” is used in MQTT broker. In the hIoT protocol, Python scripts are used on the server and SQLite is used as the database. In the topology, routers are used between the server and the IoT Gateway to measure the effect of different bandwidths. In the topology, a copy of all traffic sent and received by IoTGW was captured with WireShark to obtain average values. In the topology, 6 Arduino, 1 Raspberry Pi and 1 PC were used for analysis. Each communication was repeated 30 times and the mean values were calculated.

According to the information obtained from the captured packets, the average latency for six different bandwidths were calculated. As can be seen from Figure 24, at low bandwidths, the hIoT protocol has a lower latency. However, when the bandwidth increases, the gap between the latency of both protocols is closed.

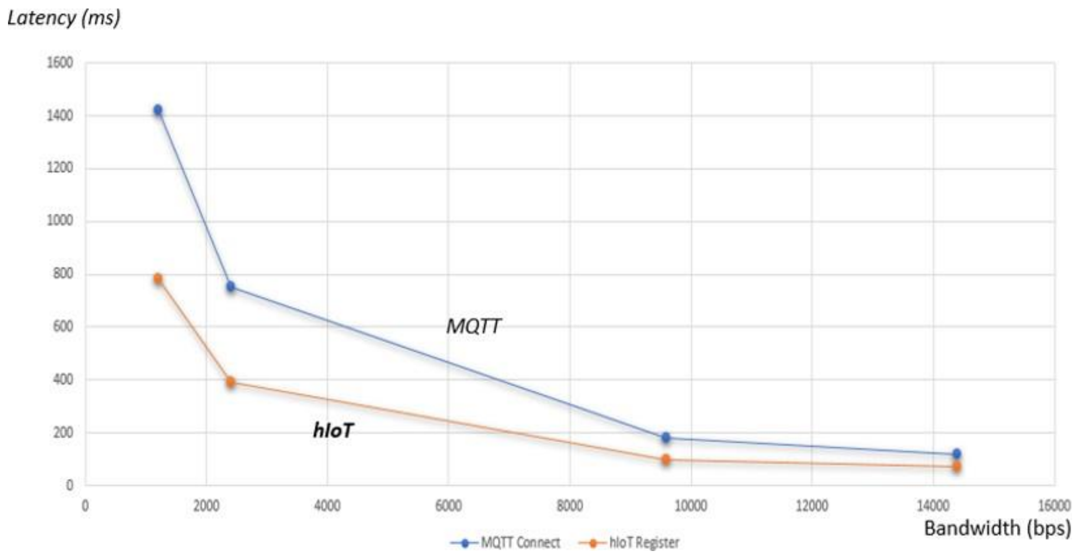

Figure 24. Latency in different bandwidths

In another analysis, latency based on payload were compared, during data transfer. In the end- to-end communication hIoT protocol has higher performance. However, MQTT showed higher performance in server based communication. The performance results are shown in Figure 25.

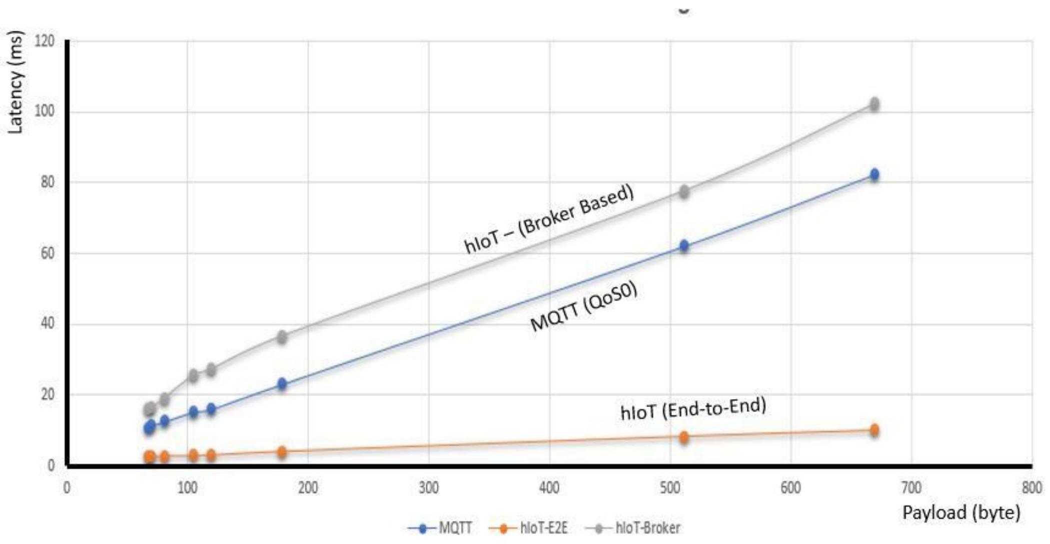

Figure 25. Latency in different payloads.

With the use of UDP in the proposed protocol and the simplicity of the packet header, it may be more suitable for devices with limited resources. Additionally, in applications where speed is important in the IoT ecosystem, the hIoT protocol can be used.

3.7.1 Limitations of Experiment

In this experimental testbed, evaluations were made on wired media. In order to evaluate the performance of the proposed protocol, tests should also be performed in wireless environments. In this study, the proposed protocol was compared only with MQTT protocol. Performance comparisons can be made with protocols used in the IoT ecosystem, such as CoAP. It will also be useful to make evaluations for multiple nodes using various simulation tools.

4. Conclusions

Data from objects in the IoT ecosystem differ according to applications. However, the data transferred in IoT applications are generally low and limited in size. The hIoT protocol, which was developed for the transmission of small data from sensors, is UDP-based and has a small packet header. Considering that the devices used in the Internet have low resources such as limited memory, processor and power, low overhead also contributes to efficient use of resources. Low latency times can be seen as an important criterion, especially in real-time applications. In the study of Jürgen et al., it is stated that UDP-based communications have lower RTT values in IoT environments than TCP-based communications.

In commonly used IoT protocols, information obtained from objects is sent periodically to the server via sensors. This information is again broadcast to the subscribers via the server. This means continuous use of the sensor and a relatively shorter life of the sensor and an increase in power consumption. In the architecture where the developed protocol will be used, optional communication is aimed. Thus, it is aimed to reduce the power consumption relatively.

In the hybrid protocol proposed within the scope of this study, there is no continuous dependence on the server in data communication and the devices in the network can communicate directly with each other. In this way, a single point of failure is prevented and traffic load is distributed and bottleneck formation is prevented. Again, in order to support the scenarios in which the central server is used, it is designed to be server-based. In cases where access to the data should be precise and instant, the sensors can be instantaneously communicated end-to-end via the sensors.

In heterogeneous IoT systems, access to each service is not equally important. Some services are critical, requiring direct communication, while others may compensate for this delay. Likewise, direct access to some services may involve security and performance risks, so it may be more appropriate to use middleware structures. This situation varies according to needs, usage scenarios, and security policies. In this study, a hybrid protocol has been designed to support various usage scenarios mentioned above in IoT systems.

In this study, whether the service supports direct access was determined during the service registration phase and it was seen to remain constant. However, instant decision making algorithms can be developed according to the determined traffic criteria. Again, according to various parameters, traffic estimation can be made and studies can be made to provide dynamic transitions between server-based or end-to-end communication.

Since most of the devices used in the IoT ecosystem have limited capacities, security features that require high memory and processing power, such as encryption, during the design of the protocol are excluded from this study. Considering that these limitations will gradually decrease with the development of technology, encryption, data integrity, and authentication studies can be performed in order to ensure secure communication in the protocol.

REFERENCES

[1] A. Anjum, M. U. Ilyas, D. Gorden, C. Gar, and Guo et al, *Internet of Things – From Research and Innovation to Market Deployment*, vol. 6, no. 1. River Publishers, 2014.

[2] L. Atzori, A. Iera, and G. Morabito, "The Internet of Things: A Survey," *Comput. Networks Int. J. Comput. Telecommun. Netw.*, vol. 54, no. 15, pp. 2787–2805, 2010.

[3] J. Höller, V. Tsiatsis, C. Mulligan, S. Karnouskos, S. Avesand, and D. Boyle, *M2M and IoT Technology Fundamentals*. 2014.

[4] J. Höller, V. Tsiatsis, C. Mulligan, S. Karnouskos, S. Avesand, and D. Boyle, "Part III. IoT Use Cases," *From Mach. to Internet Things,* pp. 233–235, 2014.

[5] J. Gubbi, R. Buyya, S. Marusic, and M. Palaniswami, "I*nternet of Things (IoT): A vision, architectural elements, and future directions*," *Futur. Gener. Comput. Syst.*, vol. 29, no. 7, pp. 1645–1660, 2013.

[6] ITU-T, "Focus Group on M2M service layer: APIs and protocols overview," *International Telecommunication Union*, 2014. [Online]. Available: https://www.itu.int/opb/publications.aspx?parent=T-FG&selection=6&sector. [Accessed: 31-Oct-2019].

[7] A. Talaminos-Barroso, M. A. Estudillo-Valderrama, L. M. Roa, J. Reina-Tosina, and F. Ortega- Ruiz, "A Machine-to-Machine protocol benchmark for eHealth applications - Use case: Respiratory rehabilitation," *Comput. Methods Programs Biomed.*, vol. 129, pp. 1–11, 2016.

[8] P. Bellavista and A. Zanni, "Towards better scalability for IoT-cloud interactions via combined exploitation of MQTT and CoAP," *2016 IEEE 2nd Int. Forum Res. Technol. Soc. Ind. Leveraging a Better Tomorrow, RTSI 2016*, 2016.

[9] J. Dizdarevic, F. Carpio, and A. Jukan, "S*urvey of Communication Protocols for Internet-of- Things and Related Challenges of Fog and Cloud Computing Integration*," *CoRR*, vol. 1, no. 1, pp. 1–27, 2018.

[10] M. Collina, M. Bartolucci, A. Vanelli-Coralli, and G. E. Corazza, "*Internet of Things application layer protocol analysis over error and delay prone links*," in *2014 7th Advanced Satellite Multimedia Systems Conference and the 13th Signal Processing for Space Communications Workshop, ASMS/SPSC 2014*, 2014, vol. 2014-Janua, pp. 398–404.

[11] G. Fortino, A. Guerrieri, W. Russo, and C. Savaglio, "Integration of agent-based and Cloud Computing for the smart objects-oriented IoT," *Proc. 2014 IEEE 18th Int. Conf. Comput. Support. Coop. Work Des. CSCWD 2014*, pp. 493–498, 2014.

[12] H. Kim, "Securing the Internet of Things via Locally Centralized , Globally Distributed Authentication and Authorization," 2017.

[13] P. Thota and Y. Kim, "Implementation and Comparison of M2M Protocols for Internet of Things," in *2016 4th Intl Conf on Applied Computing and Information Technology/3rd Intl Conf on Computational Science/Intelligence and Applied Informatics/1st Intl Conf on Big Data, Cloud Computing, Data Science Engineering (ACIT-CSII-BCD)*, 2016, pp. 43–48.

[14] P. H. Su, C. Shih, J. Y. Hsu, K. Lin, and Y. Wang, "Decentralized Fault Tolerance Mechanism for Intelligent IoT / M2M Middleware," in *2014 IEEE World Forum on Internet of Things (WF- IoT)*, 2014, pp. 45–50.

[15] N. Pathania, "Traffic Prioritization in an MQTT Gateway," in *International Journal of Computer Applications*, 2017, vol. 164, no. 2, pp. 32–38.

[16] D. Thangavel, X. Ma, A. Valera, H. X. Tan, and C. K. Y. Tan, "Performance evaluation of MQTT and CoAP via a common middleware," in *IEEE ISSNIP 2014 - 2014 IEEE 9th International Conference on Intelligent Sensors, Sensor Networks and Information Processing, Conference Proceedings*, 2014, no. November.

[17] D. Lars, "Performance Evaluation of M2M Protocols Over Cellular Networks in a Lab Environment," in *Conference: 18th International Conference on Intelligence in Next Generation Networks (ICIN), 2015, At Paris, France*, 2015, pp. 70–75.

[18] J. E. Luzuriaga, M. Perez, P. Boronat, J. C. Cano, C. Calafate, and P. Manzoni, "A comparative evaluation of AMQP and MQTT protocols over unstable and mobile networks," in *2015 12th Annual IEEE Consumer Communications and Networking Conference, CCNC 2015*, 2015, pp. 931–936.

[19] I.-J. Shin, B.-K. Song, and D.-S. Eom, "International Electronical Committee (IEC) 61850 Mapping with Constrained Application Protocol (CoAP) in Smart Grids Based European Telecommunications Standard Institute Machine-to-Machine (M2M) Environment," *Energies*, vol. 10, no. 3, p. 393, 2017.

[20] M. Collina, M. Bartolucci, A. Vanelli-coralli, and G. E. Corazza, "Internet of Things Application Layer Protocol Analysis over Error and Delay prone Links," in *2014 7th Advanced Satellite Multimedia Systems Conference and the 13th Signal Processing for Space Communications Workshop (ASMS/SPSC)*, 2014.

[21] R. Klauck and M. Kirsche, "Bonjour Contiki: A Case Study of a DNS-Based Discovery Service for the Internet of Things," in *Ad-hoc, Mobile, and Wireless Networks*, 2012, pp. 316–329.

[22] A. J. Jara, P. Martinez-Julia, and A. Skarmeta, "Light-weight multicast DNS and DNS-SD (lmDNS-SD): IPv6-based resource and service discovery for the web of things," *Proc. - 6th Int. Conf. Innov. Mob. Internet Serv. Ubiquitous Comput. IMIS 2012*, pp. 731–738, 2012.

[23] D. Choi, J. Jung, H. Kang, and S. Koh, "Cluster-based CoAP for Message Queueing in Internet- of-Things Networks," pp. 584–588, 2017.

[24] C. Huo, "A Centralized IoT Middleware System for DevicesWorking Across Application Domains Using Self-descriptive Capability Profile," University Of California, 2014.

[25] Y. Chen and T. Kunz, "Performance evaluation of IoT protocols under a constrained wireless access network," in *2016 International Conference on Selected Topics in Mobile and Wireless Networking, MoWNeT 2016*, 2016.

[26] A. H. Ngu, M. Gutierrez, V. Metsis, and Q. Z. Sheng, "IoT Middleware : A Survey on Issues and Enabling Technologies," *IEEE Internet Things J.*, vol. 4, no. 1, pp. 1–20, 2017.

[27] B. Kang, D. Kim, and H. Choo, "Internet of Everything: A Large-Scale Autonomic IoT Gateway," *IEEE Trans. Multi-Scale Comput. Syst.*, vol. 3, no. 3, pp. 206–214, 2017.

[28] S. M. Kim, H. S. Choi, and W. S. Rhee, "IoT home gateway for auto-configuration and management of MQTT devices," *2015 IEEE Conf. Wirel. Sensors, ICWiSE 2015*, pp. 12–17, 2016.

[29] A. Grygoruk and J. Legierski, "IoT gateway – implementation proposal based on Arduino board," in *Proceedings of the Federated Conference on Computer Science*, 2016, vol. 8, pp. 1011–1014.

AUTHORS

Erdal ÖZDOĞAN

Received B.Sc. degree from Ankara University Astronomy and Space Science, and second B.Sc. degree, from Anadolu University Management Information Systems, Turkey. M.Sc. degree from Gazi University, Computer Education, Turkey. He is a PhD candidate in Information Systems at Gazi University. He gives trainings to public institutions and organizations on Computer Networks, Cyber Security and IoT. He is still a mathematics teacher at Ministry of National Education.

O.Ayhan ERDEM

Received B.Sc., M.Sc. and PhD. degrees from Gazi University Institute of Science and Technology, Turkey. In 1990 he attended to English Language Education Program of Indiana University, USA. He finished Technology of Computing Education at Purdue University. He has books and papers about, also he is doing research in the fields of Computer Networks, Programming Languages, IT, Computer Systems. He still is a Professor at Department of Computer Engineering, Faculty of Technology, Gazi University, Ankara, Turkey.