JST: Smart Systems and Devices
Volume 31, Issue 2, September 2021, 010-018

Design and Implementation of a LoRa Communication System Supporting Edge Computing on the Smart Multi-Platform IoT Gateway

Vinh Tran-Quang[1*], Huy Dao Nguyen, Dat Tran Tien
Hanoi University of Science and Technology, Hanoi, Vietnam
[*]Email: vinh.tranquang1@hust.edu.vn

Abstract

LoRa technology was developed over 10 years ago, with many communication protocols optimized for LoRaWAN. However, in the protocols, all data from the end devices are sent directly or forwarded through a gateway to the LoRaWAN server and processed centrally there. Accordingly, the gateway only acts as a forwarder. This mechanism increases the processing load on the server, increases latency, and is not suitable for applications with a large number of end devices or that require real-time applications. In this paper, we design and develop a new LoRa communication protocol that supports edge computing at the gateway. At the same time, the authors design and manufacture a Smart Multiplatform IoT Gateway (SMGW) and LoRa nodes that allow the implementation and evaluation of the proposed protocol in practice. The test results on a system of 50 LoRa nodes and the SMGW show that the proposed protocol works well when evaluating its performance in terms of reliability, latency, and power consumption. This proposed system is suitable for applications that require edge computing and is easily extendable to other IoT applications.

Keywords: LoRa protocol, edge computing, multiplatform gateway, IoT system.

1. Introduction

In the past 10 years, the 4.0 industry and the Internet of Things (IoT) have become more popular than ever. The advantages of IoT applications are changing the way we live and work day by day. As a forecast shown recently, the number of IoT devices will almost triple from 8.74 billion in 2020 to more than 25.4 billion in 2030 [1]. The essential difference in requirements between "the Internet" and "the Internet of Things" [2] is that in IoT devices, there are fewer things available, i.e., less memory, less power of processing, less bandwidth and less available energy. One reason for these requirements is to optimize the energy consumed since maximizing the device lifetime is our priority. Furthermore, when the number of devices is colossal, we need to share bandwidth for each device. It seems that the devices in the IoT system need to "do more with less", and traditional wireless devices, such as Wi-Fi or cellular networks, are obviously not suitable for IoT devices. The rapid growth of IoT devices has generated a need for suitable low-power wide-area (LPWA) technologies that are long-ranged and consume ultralow power to substitute traditional wireless technologies [3-5]. To date, many LPWAs have been invented, such as ZigBee [6], which consumes low power but is limited in distance, and BLE [7], which consumes ultralow power and works over a very short distance. Each technology has outstanding advantages and is suitable for a special application. LoRa technology [8] was invented to satisfy IoT requirements as much as possible. LoRa can transmit data within 3 km in urban environments and 10-20 km in suburban environments [9]. The longest range recorded for LoRa to transmit data was 766 km [10]. LoRa devices also consume power at low levels, ranging from 10 to 15 mA in the receiving process and varying from 20 to 120 mA in the transmission process [11]. While patented LoRa wireless radio frequency technology stands for the physical layer protocol, LoRaWAN [12], which was developed by the LoRa Alliance, stands for the media access control layer protocol. In the LoRaWAN architecture, there are three fundamental components: the LoRa node, LoRa gateway and LoRa server. The LoRa server in a LoRa network system needs to process all packets sent by LoRa nodes (end devices). It is not a problem if our IoT system is on a small scale. However, once the number of end devices is enormous, the traditional LoRa system will not only have a large latency of message transmission but will also overload the LoRa server because it must process a huge number of messages, reducing system performance significantly. LoRa gateways, which are now only responsible for transparent forwarding messages between end devices and LoRa servers, have a good ability for processing data, and we do not need to be concerned with their power consumption.

To exploit LoRa gateway resources that traditional systems are wasting, this paper will propose and implement a new improved protocol that is easy to deploy and allows us to parse, store packets from end devices for additional purposes such as edge

ISSN: 2734-9373
https://doi.org/10.51316/jst.152.ssad.2021.31.2.2
Received: May 20, 2021; accepted: June 30, 2021

computing and still ensure all functions as LoRaWAN does.

The goal of this research is to build a complete IoT system, including designing and manufacturing hardware devices, designing communication protocols and data processing algorithms, and embedding these proposals on the devices. The rest of this paper is organized as follows. Section 2 presents the overall architecture of the Lora communication system, flow of activity between end devices, gateway, and server. Section 3 and 4 present the implementation of the proposals in both end device and the gateway. The tests are conducted in Section 5. Finally, Section 6 is the conclusion of the paper.

2. LoRa Communication System Architecture

2.1. System Architecture and Components

The overall architecture of the proposed LoRa system is illustrated in Fig. 1. It still maintains the architecture of a traditional system based on LoRaWAN consisting of 4 component layers that include end devices, LoRa gateways, servers, and applications. The functions of each component are described below:

- End device (LoRa node): The end devices with integrated sensors that measure the parameters of the environment in agriculture (i.e., humidity, temperature, pressure) or the parameters of the human body (i.e., blood pressure, pulse heart, weight). These parameters are processed by the central processor, usually a microcontroller, and then packaged with standard formatted data packets to be sent to the LoRa gateway by its LoRa module using our proposed LoRa communication protocol.

- Smart multiplatform IoT gateway (SMGW): data packets sent by end devices are received by an SMGW. The SMGW plays an important role in this proposed system since it performs many processing functions rather than simply forwards data to servers as do traditional LoRa systems. It will parse packets to obtain needed data for edge computing purposes, send acknowledgments, send commands from server to end devices if available and transmit data to server.

- Server: The server receives data from the SMGW via back-haul transport, for example, over Wi-Fi, Ethernet, or 4G/LTE. Servers store and process data and provide services to applications.

- Application: This layer interacts directly with the user, provides interfaces that allow the user to monitor the measured parameters online and allows the user to manage and control the end devices by sending control commands directly to the end devices over LTE or through the SMGW.

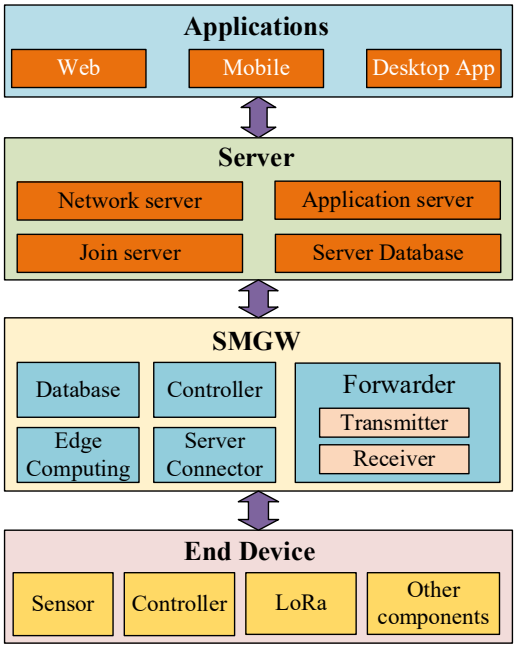

Fig. 1. LoRa communications system architecture and components.

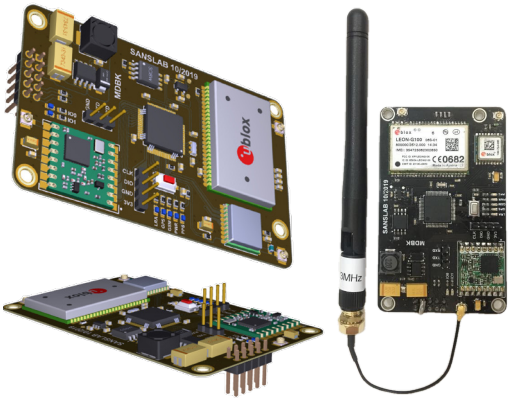

Fig. 2. Layout and a prototype of the LoRa node

2.2. End Device (LoRa Node)

We designed and manufactured a LoRa node prototype that uses STM32L072 as a microcontroller, an RFM95 module for the LoRa communication module, and MAX7Q for the GPS receiver module. Fig. 2 shows the LoRa node device after designing, manufacturing, assembling all components, firmware loading, and testing.

2.3. Smart Multi-Platform IoT Gateway (SMGW)

We propose and develop a smart and multicommunication IoT platform gateway that can provide multiple radio interfaces to allow connection to different IoT end devices using different radio communication technologies, such as LoRa, ZigBee, V2X, or Wi-Fi. The SMGW receives data from sensor nodes (the sensor nodes in this paper are the LoRa nodes that are integrated sensors), processes the data and then forwards it to the server; and receives control commands

from the server and forwards them to sensor nodes. In particular, the SMGW has the ability to perform edge computing or machine learning on data received from sensor nodes, which is scalable for many different IoT systems. With such requirements, the SMGW is designed with a functional block diagram architecture, as shown in Fig. 3. Fig. 4 is a SMGW prototype. Given the limited scope of this paper, we do not present in detail the design and assembly process of the LoRa node and the SMGW devices.

2.4. Messages Flow and LoRa Components Activity

In this system, end devices (LoRa node), the SMGW and the server are the three main components that interact with each other in a flow of activity, as described in Fig. 5.

First, LoRa node packages sensed data (step 1) and then sent them to the SMGW (step 2). Then, a receive window will be opened to receive the corresponding acknowledgement (ACK) message (step 5). The ACK message may contain a control command (CMD) from the server. If so, the node executes the control command and responds with an ACK to notify the server if the CMD command was successfully executed by the node or not (step 4). If the gateway does not receive any ACK message corresponding to the command, a message containing that command is retransmitted in the next receive window.

At the SMGW, after receiving the data packet from the nodes, the SMGW conducts data processing (step 3), for example, combining data, performing data aggregation, running edge computing algorithms or AI algorithms and then sending the processed data to the server (step 6). More details about the processing flow at the SMGW will be presented in Section 4.

2.5. Data Packet Format

The data format and message structure used in this proposed LoRa communication system are defined and described as shown in Fig. 6. There are 4 fields in the physical layer, namely, Preamble, Phy Header, Phy Header CRC, and CRC, which are preserved as in LoRaWAN [12]. These fields are added automatically when data are processed at the LoRa module's physical layer. This format is applied to both uplink and downlink messages.

As we can observe from Fig. 6, a Phy Payload starts with a MAC Header (MHDR) field of 1 byte in size. This is the MAC layer header field containing information that defines the purpose of the message, which includes the MType, RFU, and major fields. The MType field (3 bits) identifies the type of message that will be handled on the corresponding handler. Details of the message types are shown in Table 1. The RFU field (3 bits) is reserved for future usage. The major field (2 bits) contains the implemented protocol version.

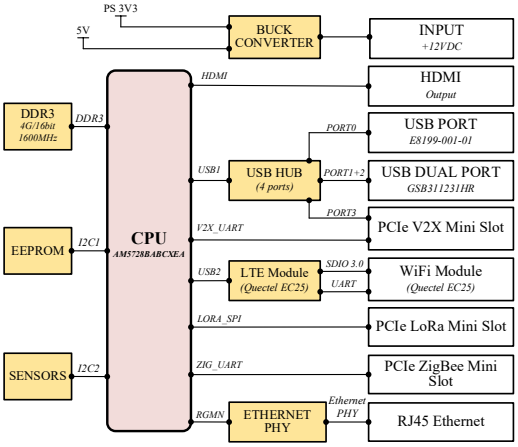

Fig. 3. Function block diagram of the Smart Multiplatform IoT Gateway (SMGW).

Fig. 4. A SMGW prototype.

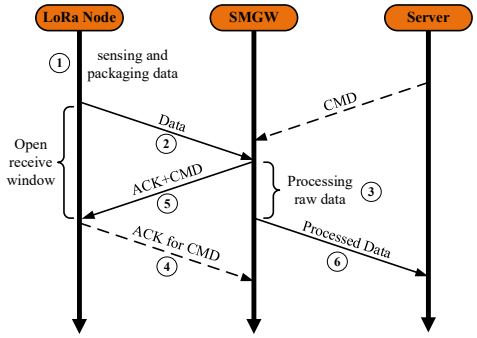

Fig. 5. Message flow and activities among the main LoRa components.

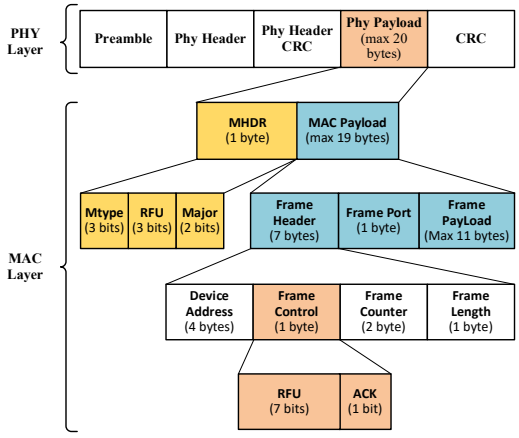

Fig. 6. Data format and message structure for the proposed LoRa communication protocol.

Table 1. MType classification

Mtype	Type of message
000	Join-request message, sent from node
001	Join-accept message, sent from gateway to notify that node has joined successfully
010	Unconfirmed data message, sent by node, no ACK
011	Confirmed data message, sent by node, ACK requirement
100	Unconfirmed command message, sent by gateway, no ACK requirement
101	Confirm command message, sent by gateway, ACK requirement
110	Reserved for future usage
111	Reserved for future usage

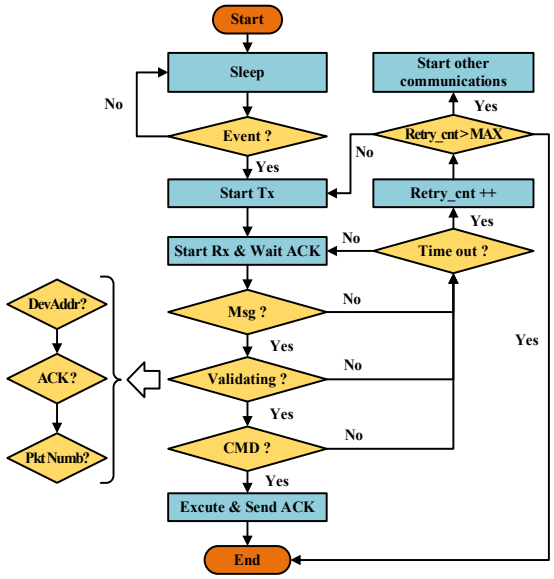

Fig. 7. Flowchart of the LoRa communication protocol implemented on the end device.

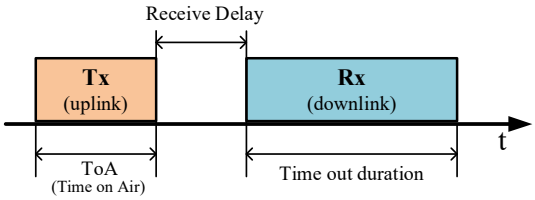

Fig. 8. Receiving time slot on the end device.

The MAC Payload, next to the MAC header field, up to 19 bytes in size, contains MAC layer data. This field is preceded by a frame header of 7 bytes in size containing subfields for control purposes. Next is a 1-byte frame port field, which is designed for future use. Finally, the frame payload field with a maximum size of 11 bytes contains the frame data, which are the data the user needs to send. The frame header field is 7 bytes in size divided into 4 subfields and used for control purposes. The device address subfields have a size of 4 bytes, where the first 2 bytes are the gateway address, and the last 2 bytes are the address of the node. The frame control subfields (1 byte) contain protocol control information. In the proposed protocol, the first 7 bits are reserved, and the last bit is the ACK bit, which enables the acknowledgment of the ACK message. The frame counter subfields (2 bytes) contain the sequence number of the packet. The frame length subfields (1 byte) represent the payload length.

3. LoRa Protocol Implementation on End Devices

3.1. LoRa Communications Protocol

After the end device (sensor node) starts activating, the node is in sleep mode to save as much power as possible. If there is an event (the event here is triggered when data from the sensor is transmitted to the processor), the node will wake up, take data from the sensor, pack data in the format presented in Section 2, and start data transmission. When the transmission finishes, the node will open a receive window in a certain timeout duration. If the receive window time expires and the node does not receive any ACK message from the gateway, it will retransmit the current packet in a certain time. If the transmission fails, the node will discard the current packet and start waiting for another event from the sensor. If a node receives a packet in timeout duration, it starts processing the packet. First, the data fields are parsed. After that, we check if the address of the devices is as expected, whether the message is an ACK or not and what the ordinal number of the message is. If these conditions are satisfied, the node keeps checking whether the packet has commanded or not. If there is a command in the packet, the node will execute this command and send a command ACK to the gateway. A flowchart of the LoRa communication protocol implemented on the end device is shown in Fig. 7.

The communication process, which is designed based on class A of LoRaWAN, is revised to make the process as simple as possible. Fig. 8 illustrates the process of transmission and reception of the proposed protocol on the end devices.

3.2. Firmware Structure

Based on the LoRa communication protocol proposed above, we develop the source code and deploy it on our end device. The firmware on the device is developed in Keil C IDE, which supports various microcontrollers, and compiled by the ARM GCC compiler, which is for microcontrollers using the ARM architecture. The firmware structure on the end device is divided into 6 layers, as shown in Fig. 9.

The application layer is the highest layer that is responsible for deploying and implementing the proposed communication protocol. The application layer consists of services such as LoRa service, GPS service, GSM/4G service and other services. The

midware layer is responsible for controlling and scheduling tasks of each service so that each can work independently and is easy to maintain if one of them has problems in the future. In this layer, we use Free-RTOS, a real-time operating system that is compatible with the ARM architecture. The abstraction layer is developed with additional tools, e.g., debugging tools and diagnostic tools. The component driver layer contains libraries for developing drivers for LoRa, GPS, or GSM modules. The peripheral driver layer consists of driver libraries for users to use peripheral modules in microcontrollers, i.e., UART, SPI, TIMER, and GPIO. Finally, the hardware layer is the physical part in the end device. This lowest layer receives data from a higher layer, performs modulation or demodulation of the received data, and passes it on to the higher layer.

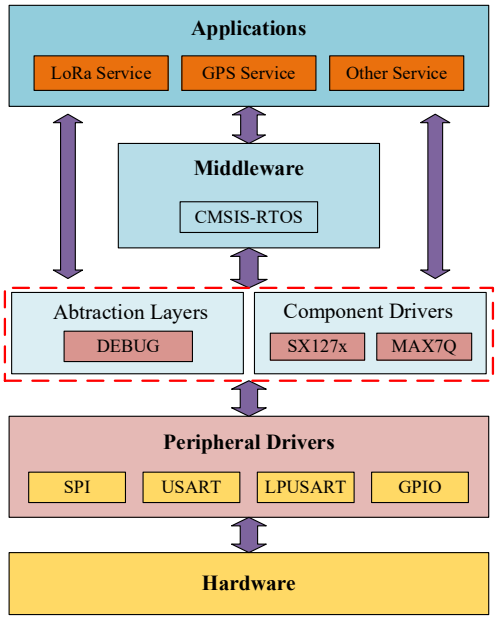

Fig. 9. Firmware structure developed and deployed on the end device LoRa node.

4. LoRa Protocol Implementation on SMGW

4.1. SMGW Components and Communication Flow

In a traditional LoRaWAN network, the gateway only acts as an intermediary device to forward data from the end devices to the server. In the proposed system, the gateway plays an important role; it allows us to receive raw data from end devices, perform functions of data aggregation, edge computation, etc., to reduce the amount of raw data from many end devices that need to be sent to the server. This is also the reason for the research team to design and manufacture the Smart Multiplatform IoT Gateway (SMGW) device and to develop and deploy the new LoRa communication protocol on this device.

The SMGW components, functions, and flow of communication among them are illustrated in Fig. 10. The gateway controller is the main part and central processor in the SMGW. All messages transmitted and received will be parsed and packed here. The Gateway Forwarder takes responsibility for communicating with end devices and the Gateway Controller. This is a gate of LoRa communication. It sends the uplink formatted message which is received via a receiver module to the Gateway Controller and sends downlink formatted messages to the end devices. The gateway-server connector is responsible for communicating with the server via back-haul transport, i.e., Wi-Fi, Ethernet, or 4G/LTE. The message processed by the Gateway Controller or computed by the Edge Computing module is moved to the Gateway-Server Connector for encryption and then sent to the server. If the server sends a command, this module decrypts it and moves the command to the gateway controller for handling. The gateway database is responsible for storing and saving parsed and processed data. Edge Computing: This module takes data saved in the Gateway database to compute and send to the Gateway-Server Connector.

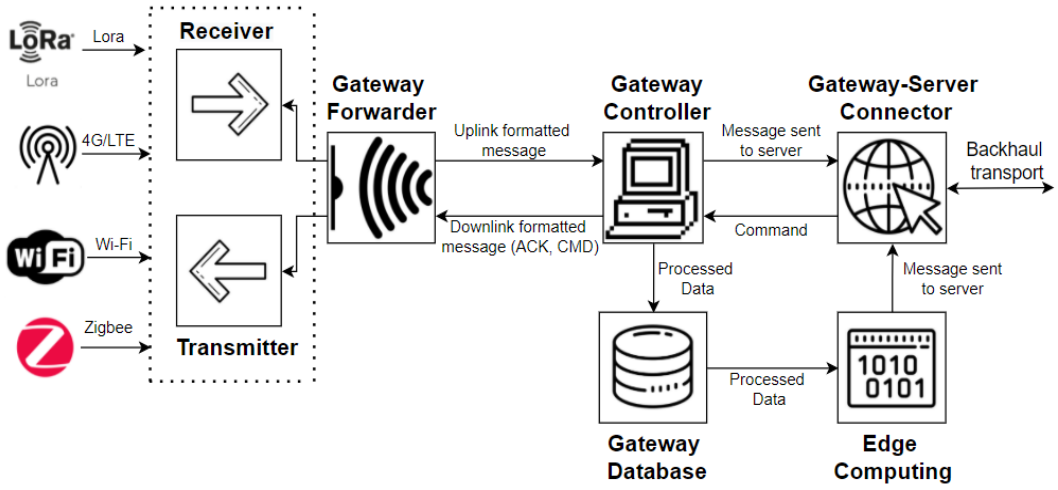

Fig. 10. The SMGW components' functions and communication flow.

4.2. Flow of Activity on Gateway Controller

Messages received from nodes are pushed into a FIFO buffer, and the Gateway Controller fetches data from this buffer continuously. These data packets are parsed, and their type is identified via its MAC header. Each message will be handled differently depending on its type. There are 2 types of messages we need to focus on: if the type of message is a data message, it is processed as the flowchart in Fig. 11, and if the type of message is an ACK message, it is processed as the flowchart in Fig. 14.

4.2.1. Data Message Processing

The data messages are validated by their gateway address (GWAddr) and packet number (PktNumb). If these conditions are not satisfied, these data messages will be discarded. After checking the conditions, data messages will be parsed and divided into fields in format in Section 3. Needed fields and related information will be stored in the Gateway Database so that the applications can take these data and apply algorithm models for edge computing purposes.

After successfully processing and storing the received data message, the gateway controller generates an ACK message and sends it to the corresponding node in its receiving window. To save power, the node's LoRa communication module is configured to always stay idle until an event occurs. Therefore, the CMD command that the gateway wants to send to the node will be sent with an ACK message. This integration will save node power, but it will also delay command execution as a trade-off. All the CMD commands from the server are pushed in a table whose header is presented in Fig.12 (a), and we call it the configuration and control table. When the gateway generates an ACK message, it must check the device address and command in this table to verify that there is a control command to send to this node. The DevAddr column contains addresses of the nodes to which we need to send the control command, and the CMD column is the CMD code of the command. These codes are designed to follow the preconventional syntax between the gateway and the nodes. The CMD code in the table is encapsulated into an ACK message and sent to the corresponding node. To ensure that the node has received the control command, an ACK message from the receiving node is sent to the SMGW for confirmation. Therefore, we need a table to manage the sent commands. This is the In-Progress-Command (IPC) table with its header shown in Fig. 12 (b).

The IPC table contains commands that wait for an ACK from the node. DevAddr is the address of the node that must be sent the ACK. The CMD field contains a list of waiting ACK commands. The start time field is the time when the command started waiting for ACK. The timeout field is the ACK timeout. If that duration exceeds timeout, that command will be moved and posted in the first row of the command table. Those commands that have received ACK will be deleted from the IPC table. These processes are illustrated by the flowchart in Fig. 13. If there are no commands left in the command table, the SMGW will send ACK messages only.

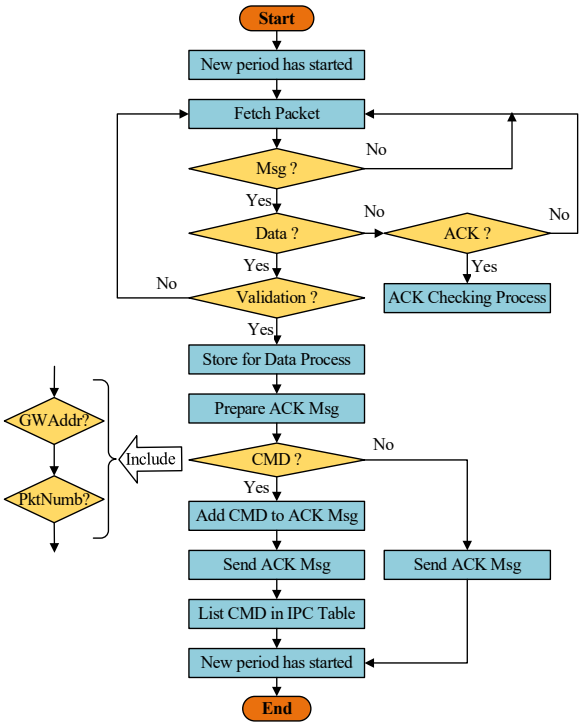

Fig. 11. Flowchart of LoRa communication implemented on the SMGW.

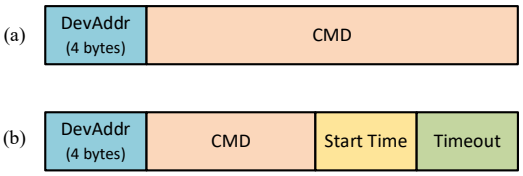

Fig. 12. The commands header table (a) and the IPC header table (b).

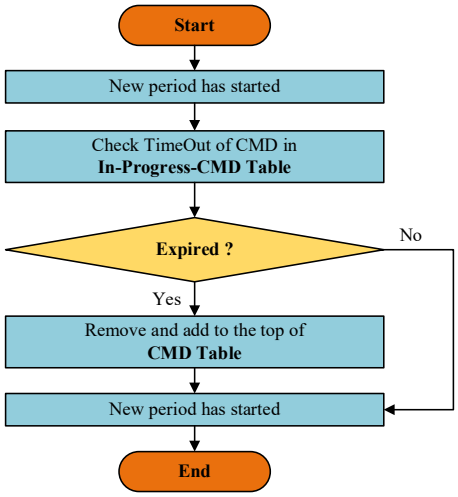

Fig. 13. The flowchart of IPC processing.

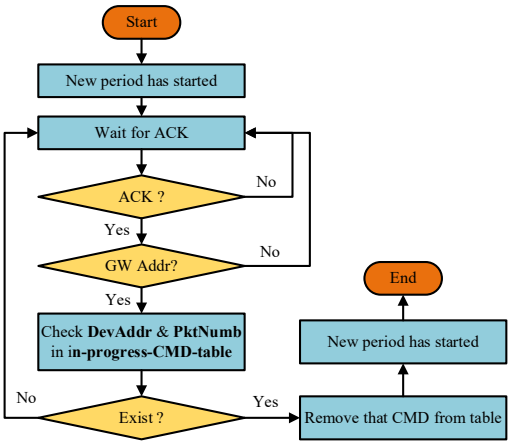

Fig. 14. Flowchart of ACK message processing.

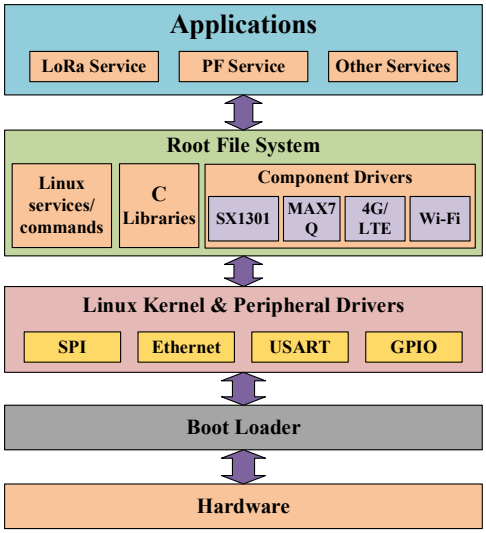

Fig. 15. Firmware structure on the SMGW.

4.2.2. ACK Message Processing

If the message sent to the SMGW is an ACK message, the SMGW will trigger a process to handle the ACK message. This process is performed as shown in the flowchart presented in Fig. 14. Following this procedure, the SMGW compares the GWAddr in the received ACK message with its own address. After validating the received ACK message, which means that the CMD has been sent successfully, it will discard the command code in the IPC table and start a new period.

4.3. Firmware Structure on the SMGW

Based on the proposed LoRa communication protocol and the process of handling data messages and control messages on the SMGW as described above, we develop the source code and deploy it on our SMGW. The firmware is developed and embedded in the Linux operating system with the structure divided into 4 layers, as presented in Fig. 15.

Applications Layer: This layer is designed similar to the application layer on the Node. It includes source code to run necessary services such as Lora service, PF service, GPS service, and 4G/LTE service.

Root File System Layer: This layer is designed with libraries to control operations of communication modules, including LoRa-SX1301, 4G/LTE, Wi-Fi, GPS MAX7Q, and Ethernet modules. All the functions that communicate with the Linux operating system are also implemented in this layer.

Linux Kernel and Peripheral Drivers: This layer includes the Linux kernel and drivers for peripheral interfaces such as SPI, UART, GPIO, USB (for mouse, keyboard), HDMI (LCD), PCIe, and RJ45 (Ethernet).

Boot-loader Layer: This layer is responsible for setting the configurations on the microprocessor (CPU AM5728) and controlling the hardware layer in response to higher layer tasks and requirements.

Hardware Layer: This is the physical layer of the SMGW that is responsible for processing electrical signals, performing modulation, demodulation of radio signals, and so on.

5. Test Results and Analysis

In this section, we present the results of comprehensive tests to evaluate the performance of the proposed system. The field test consists of 50 LoRa devices installed with the proposed LoRa communications protocol as described in Section 3 and the SMGW device installed with the LoRa communication protocol as described in Section 4. The evaluation parameters include reliability (packet loss), latency, and power consumption.

5.1. Packet Loss

To evaluate communications reliability over distance of transmission and packet loss rate, we set up the following configurations and scenarios. The SMGW is in a fixed place, and three LoRa nodes are deployed around it, as shown in Fig. 16. The distance from gateway to nodes ranges from 500 m to 3500 m in the urban environment around West Lake, Hanoi, Vietnam. The spreading factor, bandwidth, coding rate, transmit power and size of the payload of the nodes are 10, 125 kHz, 4/5, 17 dBm and 100 bytes, respectively. In the test, we sent 200 messages every 30 sec.

As shown in Fig. 17, all the nodes have the same shape of the graph, showing the relationship between transmission distance and packet loss rate. This means that all the test nodes have good communication reliability if the distance is less than 2.5 km and the packet loss ratio starts increasing dramatically if the distance is more than 2.5 km. In theory, the transmission distance of LoRa ranges from 2 km to 3 km in urban environments, and the results of this test have proven that our proposed system, in practice, still ensures the transmission distance, consistent with the theory. Although there is a high packet loss ratio when the distance is over 3 km, we can reduce this ratio by using a higher gain antenna or by placing the gateway at a higher position to ensure line-of-sight propagation.

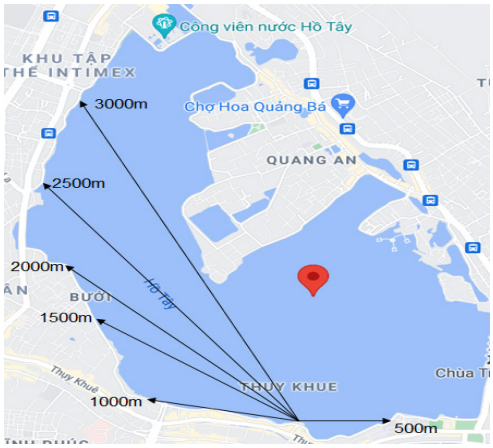

Fig. 16. Transmission range testing scenario

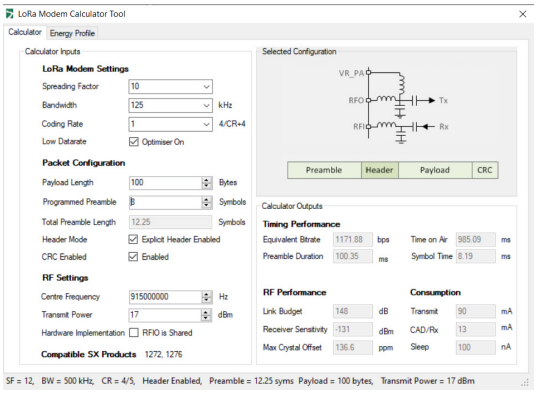

Fig. 18. Calculation of ToA in the LoRa Modem Calculator Tool.

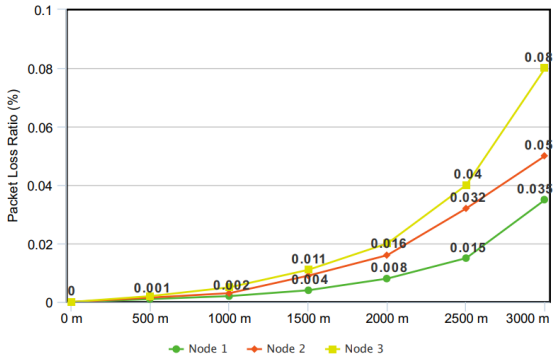

Fig. 17. Average packet loss ratio as a function of distance.

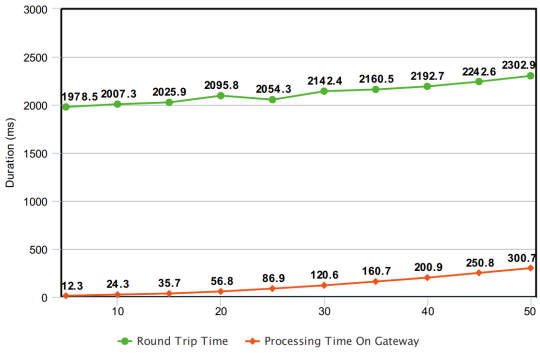

Fig. 19. Latency and processing delay as a function of the number of nodes.

5.1. Latency and Processing Delay

In the next test, we will measure and evaluate two parameters: the round trip time (RTT) and the processing time on the SMGW. The RTT is the duration, measured in milliseconds, calculated from when a node sends a message to the time it receives a response from a server. The formulation for calculating RTT is as shown in Equation (1), in which T_p is the propagation time, which is equal to the time on air (ToA) in this test, and T_s is the processing time on the SMGW.

$$RTT = 2 \times T_p + T_s = 2 \times ToA + T_s \quad (1)$$

To measure processing time T_s, we use the same configuration for the SMGW and nodes as in the distance and reliability test scenario above. In this test, each node randomly transfers 100 bytes to the SMGW. The distance between nodes and gateway is fixed at 1 km. The number of nodes ranges from 5 to 50. We capture the time when a node starts sending a message and the time when the node receives an ACK message from the SMGW to calculate RTTs. We also capture the time the SMGW received messages from the nodes and the time the SMGW sent ACK messages to calculate the processing time on the SMGW. The ToA parameters are calculated as the formulation shown in the SX127x data sheet [11]. In this test, we use the LoRa Modem Calculator Tool [11] to calculate the exact value of the ToA parameter corresponding to the configuration in this test scenario. As shown in Fig. 18, ToA in this test scenario is 985.09 ms.

The results analyzed in Fig. 19 show that as the number of nodes connecting and sending data to the gateway increases, the processing time on the SMGW increases, and therefore the RTT increases. This is obvious because as the number of nodes increases, the amount of data sent to the gateway increases, leading to an increase in processing time and resources on the gateway. However, in the proposed system, the processing time on the SMGW ranges from 20 ms for 10 nodes to 250 ms for 50 nodes. This result is acceptable and absolutely satisfies the requirement of latency (the latency is less than 300 ms in a common LoRa system with the same configuration).

5.2. Power Consumption

The power consumption of a node depends on two main components: the operation of the processor and the transceiver operation of the LoRa module. In the proposed system, the node uses an

STM32L072RBT6 as the processor unit (MCU), and the LoRa module uses the RFM95 chip for the radio transceiver. The results of theoretical calculations (based on the data sheet of STM32L072RBT6 and RFM95) and the results of practice tests show that in a normal operating mode, the node (only the MCU works) consumes approximately 1.8 μA. In sleep mode, current consumption is 350 nA. For the Lora module, at short transmission distances, the current consumed when transmitting is 23 mA, and the current consumed when receiving is 11 mA. Fig. 20 is an image of the LoRa node performance test and power consumption measurement. Table 2 summarizes the measurement results.

Table 2. Power consumption of the Lora Node.

Mode	Power consumption
Sleep mode	350 (nA)
Listening mode	1.8 (μA)
Transmission (Tx)	34 (mA)
Reception (Rx)	11 (mA)

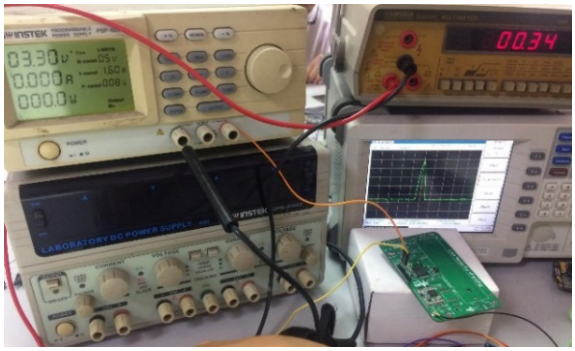

Fig. 20. Lora node performance test and power consumption measurement.

6. Conclusion

In this paper, we have introduced and proposed a complete IoT system including LoRa nodes and smart multiplatform IoT gateway (SMGW) devices. We also developed and proposed the LoRa communication protocol and implemented it on the proposed system. In the proposed IoT system, different communication technologies, such as WiFi/LTE and LoRa, can be supported by the SMGW, thanks to its ability to decode and process messages right at the SMGW. The functions allow the implementation of complex data processing algorithms such as performing edge computing or supporting artificial intelligence-based data processing models at the SMGW.

The test results with the prototypes of the SMGW and LoRa nodes show that the LoRa communications protocol works well based on the evaluation of reliability, packet loss rate, delay, and power consumption. The proposed system is also scalable to meet the requirements of different IoT applications. In the future, we will integrate other radio communication technologies, such as ZigBee and NB-IoT, into the SMGW, improve the system performance, and apply the proposed system in real IoT applications.

References

[1]. Arne Holst, Number of IoT connected devices worldwide 2019-2030, Jan 20, 2021 [Online]. https://www.statista.com/statistics/1183457/iot-connected-devices-worldwide/

[2]. Gubbi, J.; Buyya, R.; Marusic, S.; Palaniswamia, M. Internet of Things (IoT): A vision, architectural elements, and future directions. Future Gener. Comput. Syst. 2013, 29, 1645–1660 https://doi.org/10.1016/j.future.2013.01.010

[3]. T.-Q. Vinh and T. MIYOSHI, Adaptive routing protocol with energy efficiency and event clustering for wireless sensor networks, IEICE Trans. Commun. 91 (9), 2795–2805, 2008. https://doi.org/10.1093/ietcom/e91-b.9.2795

[4]. T.-Q. Vinh and T. MIYOSHI, A novel gossip-based sensing coverage algorithm for dense wireless sensor networks, Computer Networks 53 (13), 2275-2287. https://doi.org/10.1016/j.comnet.2009.04.003

[5]. T.-Q. Vinh and T. MIYOSHI, Energy balance on adaptive routing protocol considering the sensing coverage problem for wireless sensor networks, Commun. Electron. ICCE, 2008.

[6]. Noreen, U., Bounceur, A., & Clavier, L. (2017). A study of LoRa low power and wide area network technology. Proc. In 2017 International Conference on Advanced Technologies for Signal and Image Processing (ATSIP). https://doi.org/10.1109/atsip.2017.8075570.

[7]. Want, R.; Schilit, B.; Laskowski, D. Bluetooth le finds its niche. IEEE Pervasive Comput. 2013, 12, 12–16. https://doi.org/10.1109/MPRV.2013.60

[8]. LoRa Alliance, LoRa and LoRaWAN: A technical overview, Tech. Paper, Semtech, 1-26, 2020. [Online]. Available at: https://lora-developers. Semtech-.com/documentation/tech-papers-and-guides/.

[9]. Q. Zhou, K. Zheng, L. Hou, J. Xing, and R. Xu, Design and implementation of open LoRa for IoT, IEEE Access, vol. 7, pp. 100 649–100 657, July 2019. https://doi.org/10.1109/ACCESS.2019.2930243

[10]. LoRa Alliance, LoRaWAN® distance world record broken, twice. 766 km (476 miles) using 25mW transmission power. Available: https://lora-alliance.org/LoRaWAN-news/LoRaWANr-distance/

[11]. Semtech, Sx1276/77/78/79 -137 MHz to 1020 MHz low power long range transceiver. Rev. 6 - January 2019. Tech. Rep. August 2016. [Online]. Available: https://www.semtech.com/products/wireless-rf/lora-transceivers/sx1276.

[12]. LoRa Alliance, LoRaWAN Specification (V1.1). [Online]. Available: https://lora-alliance.org/resource-hub/LoRaWANtm-specification-v11, accessed Oct. 10, 2020.