DOI: 10.15849/IJASCA.240330.02

Int. J. Advance Soft Compu. Appl, Vol. 16, No. 1, March 2024
Print ISSN: 2710-1274, Online ISSN: 2074-8523
Copyright © Al-Zaytoonah University of Jordan (ZUJ)

The Impact of Protocol Conversions in the Wireless Communication of IOT Network

Geetabai S Hukkeri[1], Shilpa Ankalaki[2*], R H Goudar[3], Lingaraj Hadimani[4]

[1]Department of Computer Science and Engineering, Manipal Institute of Technology Bengaluru, Manipal Academy of Higher Education, Manipal, Karnataka 576104, India
e-mail: geetabai.hukkeri@manipal.edu

[2]Department of Computer Science and Engineering, Manipal Institute of Technology Bengaluru, Manipal Academy of Higher Education, Manipal, Karnataka 576104, India
e-mail:shilpa.ankalaki@manipal.edu

[3]Department of Computer Science and Engineering, Visvesvaraya Technological University, Belagavi, India.
e-mail: rhgoudar.vtu@gmail.com

[4]Department of Computer Science and Business system, Kolhapur Institute of Technology, Kolhapur, India.
e-mail: hadimani.lingaraj@kitcoek.in

*Corresponding author- Shilpa Ankalaki, e-mail:shilpa.ankalaki@manipal.edu

Abstract

The Internet-of-Things (IoT) concept predicts a future of interconnected formation using one or more radio automation due to the high proliferation of sensors. Multiradio mesh deployments made possible by a network of inexpensive IoT platforms, like ASUS Tinker Board S, Banana Pi M64, Raspberry Pis (RPis), etc., seem to be an ordinal fit for such deployments' wireless backbones because many of these platforms come with built-in support for various wireless technologies. To fit the capabilities of low power multiradio solutions, especially when making best effort broadcasts, the cutting-edge routing techniques must be reconsidered. This is because most currently available routing systems prioritise channel diversity while routing, omitting any potential costs related to protocol conversions. Therefore, we covered the effect of protocol conversion in IoT in this study. The three wireless technologies Wi-Fi, Zigbee, and Bluetooth have been examined. For rapid, short-distance communication, Bluetooth technology excels. Zigbee, on the other hand, offers somewhat faster, lower-power communication across longer distances. Therefore, depending on the situation, IOT devices could use Bluetooth or Zigbee. Relay agents that offer protocol conversion can connect to Bluetooth and Zigbee IOT devices using Wi-Fi technology, which operates on IP.

Keywords: *Bluetooth, Internet-of-Things, Raspberry Pi, Wi-Fi, and Zigbee*

Received 20 October 2023; Accepted 30 December 2023

1 Introduction

An interdependent network of apparatus can exchange data and communicate with one another is said to be an Internet of Things (IoT). These devices depend on protocols, which are a collection of guidelines and requirements that specify how data is transferred and received, for effective communication and cooperation.

When referring to interactions and data transfers between devices in an IoT network, we use the term "protocol conversions." Device collaboration, action coordination, information sharing, and task completion depend on these transformations. Several significant features of protocol conversions on the IoT are shown in Table 1.

Table 1: Features of protocol conversions on the IoT.

Features	Description
Device Discovery	Within the network, devices must find and identify one another. This is possible with the help of protocols like the Simple Service Discovery Protocol (SSDP) or mDNS (Multicast DNS), which let devices broadcast their presence and make them accessible to other devices so that they may be found and interacted with.
Data Transmission	Data communication between Internet of Things devices is made easier via protocols. They specify how data packets should be sent and received, as well as their structure and rules. HTTP (Hypertext Transfer Protocol) are common IoT protocols and MQTT (Message Queuing Telemetry Transport), CoAP (Constrained Application Protocol).
Data Representation	Data in different formats is frequently generated and exchanged by IoT devices. To ensure device compatibility and interoperability, protocols specify how data should be organised and encoded. XML (eXtensible Markup Language), JSON (JavaScript Object Notation), and CBOR (Concise Binary Object Representation) are three common formats for data representation.
Security and Authentication	As IoT devices proliferate, it becomes increasingly important to ensure the security and integrity of data. To protect data transmissions and stop unauthorised access, protocols like TLS (Transport Layer Security) and DTLS (Datagram Transport Layer Security) include encryption and authentication procedures.
Event Notification and Subscriptions	IoT devices frequently need to transmit events and status changes. Real-time bidirectional communication is made possible by protocols like WebSockets, allowing devices to transmit event notifications and create subscriptions to receive changes.
Interoperability	The IoT ecosystem is made up of a variety of gadgets from various manufacturers, operating systems, and technologies. Integral communication requires interoperability. It is the goal of protocols like Open Platform Communications Unified Architecture (OPC UA) and Zigbee to create standardised interfaces and protocols for communication between various systems and devices.

The International Telecommunication Union initially suggested the idea of the Internet of Things in 2005 [1], and as a result, IoT has drawn attention from many industries. To communicate and exchange information with all users, the IoT links millions and billions of devices [2].

Numerous IoT-related devices and applications have been investigated with the active improvement of many apparatus operators, including smart homes, smart medicine, intelligent transportation, etc. [3], as well as smart cities using sensors and smart information processing systems to manage daily traffic in cities [4]. Home energy management systems also help people understand how to operate their smart appliances [5]. Cloud computing also provides computer resources that are made available as a service across the entire local network or the Internet [6]. Today's sensor objects may describe information differently from vendor to vendor due to the lack of a consistent communication protocol for object information and format descriptions. As a result, it is hard for different IoT devices to interpret the information contained in one another. To construct an IoT-CPCS, this study suggests that communication protocols be converted. By operating a precise communication protocol and linguistic analysis, the proposed scheme aims to blend the formats of the data collected by various IoT devices, convert that data into useful and essential information, present that information in readable message formats, and subsequently store those messages in virtual servers constructed in the cloud platform. As a result, communication protocols will no longer prevent IoT devices from different vendors from communicating with one another. This will make it easier to develop related IoT applications and services, lower the cost of semantic conversion, and streamline the semantic conversion process.

The interlinking and connectivity of several radio standards is made possible by protocol conversion, which in multiradio multihop networks becomes an essential component of routing. However, despite its clear benefits, protocol conversion between radio technologies can present several difficulties while transmitting data. Mismatched frame formats, limitations in conversion logic, and, most importantly, problems with fragmentation and aggregation are all included in this. Due to variations in the payload supported by each radio technology, packet fragmentation and aggregation [8, 12] take place. The effect of protocol conversion on several network performance characteristics is covered in several earlier papers [7], [9-11].

It is possible to measure the overhead associated with the same in terms of price, packet loss, delay, energy, or link utilization. This most effective relaying strategy involves retrieving the packet content first and adding the new technology header solely to the payload when doing protocol conversion at a relay gateway. The task's intended goal is to demonstrate the effectiveness and impact of converting Wi-Fi-based packets to other protocols and relaying them through the RPi platform. This is accomplished by calculating the delay in relation to both protocols.

This research's objective is to analyse how protocol conversion affects a multi-technology IoT system. This is accomplished by implementing various methods of protocol conversion on the practical and widely used multi-technology Raspberry Pi IoT platform. This work has been implemented in the following three steps:

- To enable communication between two laptops, we set up a Wi-Fi packet-based relay at the Raspberry Pi in the first step.
- Using a Raspberry Pi functioning as the relay gateway, we implemented and examined independently the effects of protocol conversion (Wi-Fi to Bluetooth and Wi-Fi to Zigbee).

- In the third step, we put the protocol conversion (Wi-Fi to Bluetooth and Wi-Fi to Zigbee) into practise and investigated how parallel transmissions were affected. A Raspberry Pi served as the relay gateway.

Overall, protocol conversions are essential for enabling IoT device coordination and communication. They specify how IoT systems' security and interoperability are maintained as well as how devices locate one another, connect, and communicate data. IoT devices may successfully cooperate with one another by adhering to defined standards, creating smart environments that are more productive and interconnected.

The rest of the paper is structured as follows. Research context, research inspiration, and research purpose are presented in Section 1. The literature on IoT routing, multiradio routing, and protocol conversions is found in Section 2. The difficulties of protocol conversion are covered in Section 3. The process and findings of the three phases of protocol conversion are presented in Section 4. The conclusion of Section 5 elaborates on the research findings and examines the contributions of this paper as well as potential future research directions.

2 Related Work

The work in this field is divided into three groups.

2.1. Routing in IoT

For IoT-based deployments, distributed routing methods like LEACH [13] and its upgrades [14] have been shown to be the most popular and successful option. However, when used over a vast area requiring high coverage, IoT routing systems like these either combine the specifics and requirements of each radio technology [15], [16] or suffer from considerable energy-related losses [17],[18],[19]. Considering large-area IoT deployments, the current regulatory norms are shifting in favour of more universal, standard-based solutions. In this context, IPv6-enabled routing protocols for low-power lossy networks (RPLs) have gained a lot of attention, especially since the majority of IoT devices now support 6LoWPAN [20],[21], [22]. While these solutions support larger IoT installations with standardised and IP compatible technologies, Chorus is a more localised, simpler solution supporting multihop deployments at a lower scale and supporting even non-IP-based wireless protocols. Furthermore, Chorus, using its multilayer graph modelling, supply all transmission costs incurred when transmitting across multitechnology example, along - with the part added by potential protocol conversions, in contrast to few new multiradio IoT routing resolution indifferent to radio differences [23], [24].

2.2. Multiradio Routing

Recent research on routing in multiradio systems offers a variety of routing algorithms that may be tuned to fit different performance characteristics related to throughput [25-28], fairness [29], [23], [30], energy [28], [31] etc. These works assumed that protocol conversion overheads were insignificant, which is a reasonable assumption when multiple radios have the same packet sizes and formats and only hardware complexity exists. However, when we consider radios that have a wide range of packet sizes, formats, and transmission techniques. Due to the different formats and sizes, the converting node must handle software-related issues like handling fragmentation, aggregation, missing fields in packet format, etc. [24], [32]

2.3. Protocol Conversions

RTT (round-trip time), time delay ratio, and other performance-related computations have all been examined in relation to protocol conversions. Conversions from IPv6 to 6LoWPAN, from BSN to WiFi, and from ZigBee to BT have all been investigated by various studies; nevertheless, these conversions had the worst case overhead of 100 ms in RTT [33], 100%-time delay miss ratio [34], and up to 10% packet loss [35]. This research [36] dealt with single-hop application-centric calculations that solely used aggregation or fragmentation, but not both. As a result, these studies cannot be directly applied to the research of conversion's effects in a multiradio multihop routing.

3 The Challenge and Future Directions

What changes if a device in a communications peer group is not managed, whether it is a sensor or a small, resource-constrained device, or because it is an outdated piece of apparatus that does not brace any operation protocol? The truth is that while it will be able to function and offer its services, when network issues arise it won't be feasible to pinpoint their cause or take preventative actions to stop them from happening again.

As a result, maintaining a literal version of the operating characteristics of this equipment must be completed manually using remote control programmes like SSH, telnet, or web access. These programmes' results display values in rea-life, but they have boundaries when it comes to historical data, including low data, volatile records, potential typing errors, an insufficiency to incorporate the facts confined with management systems, emptyness in association of events, and disregards.

With this, it is impossible to obtain system performance data, which is essential for managing and running the communications network effectively. In addition, a webwork has appliances from several dealers, running management applications to each of them, which presents challenges in integrating the various applications that are used to manage and control, and to gather functioning information of the facilities in a communications framework [37].

These characteristics increase the threat proxies that make system operation more difficult than it should be. The problems can appear in various contexts, so this research focusses on to convert supervision protocols in legacy systems using a method that can be used in agents of ad hoc networks with low processing power, like mesh networks, the internet of things, or embedded systems.

3.1 Legacy systems

The term "legacy systems" refers to equipment, software, or services that are nearing "obsolescence" or are being substituted by new innovations but are still in use by businesses because they have made significant financial investments in them, their services are still in demand, they have paid for themselves, or their replacement would be extremely challenging to implement [38].

Most of the study on including, maintaining, or migrating legacy systems focuses on software other than hardware; replacing a device possibly be simpler than replacing a single application, albeit it is still expensive. However, the outcomes of such study might be connected to comprehend, for instance, elements like cost, time, flaws, or capabilities that contribute to a successful migration procedure [39]. The integration of legacy systems also faces difficulties that go beyond technical ones, including issues with culture, the quality of the information provided, utility, compatibility, and others [40], where organizational variables have a significant impact.

This difficulty can be seen, for instance, in the satellite earth stations, whose topology is depicted in Fig. 2. They have been operating for a while, but they may not be monitored since their transmitters, amplifiers, converters, radio frequency controllers, and other components are not set up in the management network. The antennas are typically far away from the baseband equipment, beyond the range permitted by other types of connections like Ethernet, hence these devices employ serial communications for their control.

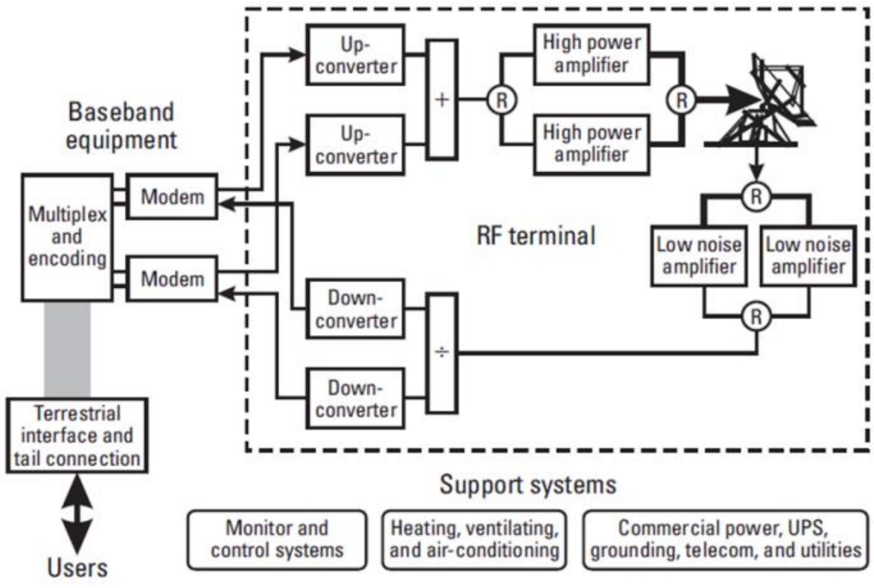


Fig. 1. component parts of an earth station [41].

In Fig 1, RF terminal, baseband hardware, ground interface, and support systems are included. The final one demonstrates the utilization of monitoring and control systems. If an earth station has an acceptable monitoring and control architecture that enables administrators to find, identify, and fix technical issues as well as makes it easier to carry out changes or configurations, it can be administered locally or remotely [41]. The devices have the tools to achieve this, but because the management infrastructure's integration capabilities are limited, those tools are not used.

3.2 Sensors on Ad-Hoc Networks

The IoT networks, whose components have features such as heterogeneity, reduced energy consumption, and unique characteristics of wireless link [9] that must be taken into consideration for the development of any application, are another example of the difficulties associated with the oversight of devices with management limitations. Due to their compact form dimensions and purpose-built functionality, the sensors in an Ad Hoc, Mesh, or Sensor Network have energy restrictions that reduce their longevity or battery life. There are advantageous working directions [10] that make use of a strategy of node collaboration as well as the most efficient design of the node and its wireless link to attempt to overcome this constraint. By combining these two ideas at the level of network management, it is possible to infer that the strategy of giving the sensors the best roles possible in order to increase their efficiency can omit the management procedures, but there may be a collaborative approach with an auxiliary node that supports and manages this service, allowing the sensors to focus solely on processing the data necessary for your application.

3.3 Internet of Things (IoT)

In comparison to other networking research topics, the IoT technology and its corresponding research state are in their infancy. Modern connectivity models and topologies are just now being created. IoT broadly and IoT in healthcare continue to confront numerous obstacles. To fully realize the potential of the IoT in industrial, healthcare, and individual use cases, among other promising fields, it is essential to comprehend and investigate these obstacles, such as the security and privacy threats, lack of standardization, massive data management, and prices.

4 Experimentation on Impact of Protocol Conversions

This section contains the detailed discussion on the impact of protocol conversion implementation.

4.1 Installations and Setups

- *Raspberry Pi*: Using the NOOBS installer, Raspbian OS was installed on a Raspberry Pi 3 and set up in accordance with the project's specifications. As part of our implementation, a Raspberry Pi must serve as a Wi-Fi repeater. As a result, Raspberry Pi was set up as a Wi-Fi hotspot. The Raspberry Pi is ready to function as a repeater after the Wi-Fi hotspot has been set up. Client-1 will now use Raspberry Pi as a repeater to transmit data to Client-2.
- *Bluetooth:* The Raspberry Pi and laptop both have built-in Bluetooth capabilities that are used for Bluetooth connection.
- *Zigbee*: Digi xBee S2S module has been utilized for Zigbee communication. Through the Uni4 Zigbee/xBee USB module, it is linked to both the Raspberry Pi and client 2. Through X-CTU, the Digi xBee S2S module is set up. Table 2 provides a summary of configuration.

Table 2: Zigbee Configuration

Sl.No	Parameter	Configuration
1	SC scan channel	8
2	ID PAN ID	2015
3	DH Destination Address High	Higher bits of serial no of another device
4	DH Destination Address Lower	Lower bits of serial no of another device
5	CE (Coordinator Enabled)	enabled for relay and disabled for client 2
6	AP API Enabled	Transparent Mode

- *NTP (Network Time Protocol) time sync:* It is necessary to synchronize clients 1, 2, and the server (Raspberry Pi) to a main time server to measure the delay brought on by protocol conversion. To do this, pool.ntp.br was used to synchronize the system time with the server after installing ntpdate.

4.2 Procedure and Steps Followed

This work has been implemented in the following three phases:

a. To enable communication between two laptops, we set up a Wi-Fi packet-based relay at the Raspberry Pi in the first step.
b. Using a Raspberry Pi functioning as the relay gateway, we implemented and examined independently the effects of protocol conversion (Wi-Fi to Bluetooth and Wi-Fi to Zigbee).
c. In the third step, we put the protocol conversion (Wi-Fi to Bluetooth and Wi-Fi to Zigbee) into practice and investigated how parallel transmissions were affected. A Raspberry Pi served as the relay gateway.

Phase 1: Understanding Wi-Fi technology and researching the use of Wi-Fi-based repeaters for communication between two clients were part of the first phase. Below is a discussion of the network setup, procedure, and observations. As shown in Fig. 2, the setup for immediate communication among IoT devices includes two laptops interconnected through Wi-Fi.

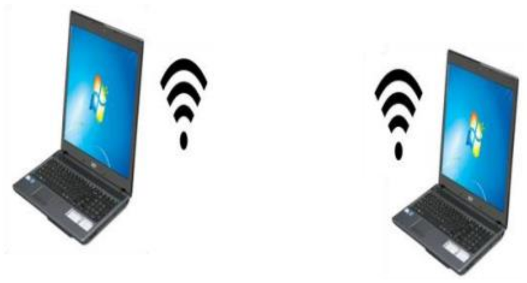


Fig 2. Two Laptops directly connected.

The network setup for RPi based relaying consists of two laptops acting as IOT devices and RPi acting as Wi-Fi based relaying gateway interconnecting two IOT devices as depicted in Fig 3.

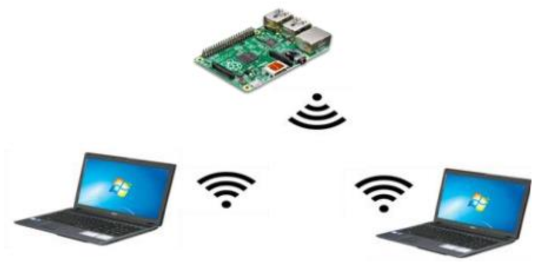


Fig 3. Two Laptops through RPi Gateway.

A. *Transmission of Wi-Fi packets without repeater:* In the first setup of this phase, two computers connected through Wi-Fi allowed for direct communication between IOT devices. To function as a server, one laptop has a Wi-Fi hotspot configuration. Without using a Wi-Fi repeater, another laptop serves as the client and delivers 100 packets to the server. Socket programming is used to carry out the transmission mechanism. The steps followed in this procedure have been given below.

Below is the algorithm for creating a socket file descriptor of datagram type:

Algorithm I: *Client-side Algorithm: Without repeater*

```
Step 1: Create a socket file descriptor using the socket () system call.
Step 2: Create a sockaddr_in structure to store the server information,including the IP address and port number.
Step 3: Fill in the server information in the sockaddr_in structure.
Step 4: Create a buffer of some fixed size to store the message to be sent to the server.
Step 5: Use a for loop to send 100 packets to the server. The sendto () system call can be used to send the packets.
Step 6: Close the socket using the close () system call.
```

Below is the algorithm for creating a socket file descriptor of datagram type, sending 100 packets to the server, and closing the socket:

Algorithm II: *Server-side Algorithm: Without repeater*

```
Step 1: Create a socket file descriptor using the socket () system call.
Step 2: Create a sockaddr_in structure to store the client and server information, including the IP address and port number.
Step 3: Fill in the server information in the sockaddr_in structure.
Step 4: Bind the socket to the sockaddr_in structure using the bind () system call.
Step 5: Create a buffer of some fixed size to store the message from the client.
Step 6: Use a for loop to receive 100 packets and store them in the buffer. The recvfrom () system can be used to receive the packets.
Step 7: Close the socket using the close () system call.
```

B. *Transmission of Wi-Fi packets through repeater:* In the second setup of this phase, two laptops are used as IOT devices, and an RPi serves as a Wi-Fi relaying gateway to connect the two IOT devices. 100 packets will be sent from client-1, a single laptop, to client-2, a second laptop, through Raspberry Pi (a WiFi repeater). To let the server (Raspberry Pi) know the address of client2, client-2 will first send a "hello" message. Client1 to Client2 packet transfer can now be started. Socket programming is used to carry out the transmission mechanism. Below is the algorithm for creating a socket file descriptor of datagram type:

Below is the algorithm for creating a socket file descriptor of datagram type:

Algorithm III: *Server-side Algorithm: With repeater*

```
Step 1: Create a socket file descriptor using the socket () system call.
Step 2: Create a sockaddr_in structure to store the client1, server, and client2 information, including IP address and port number.
Step 3: Fill in the server information in the sockaddr_in structure.
Step 4: Bind the socket to the sockaddr_in structure of the server using the bind () system call.
Step 5: Create a buffer of some fixed size to be used to receive and forward the message from client1 to client2.
Step 6: Receive a hello message from client2 using the recvfrom() system call.
Step 7: Use a for loop to forward 100 packets from client1 to client2 by temporarily storing them in the buffer.
Step 8: Close the socket using the close () system call.
```

Below is the algorithm for creating a socket file descriptor of datagram type, sending a hello message to the server, and receiving 100 packets from the server:

Algorithm IV: *Client2-side Algorithm*

```
Step 1: Create a socket file descriptor using the socket() system call.
Step 2: Create a sockaddr_in structure to store the server information, including IP address and port number.
Step 3: Fill in the server information in the sockaddr_in structure.
Step 4: Create a buffer of some fixed size to store the message that will be sent to the server.
Step 5: Send a hello message to the server using the sendto() system call.
Step 6: Use a for loop to receive 100 packets from the server.
Step 7: In each iteration of the loop, receive a packet from the server using the recvfrom() system call.
Step 8: Close the socket using the close () system call.
```

Phase 2: The goal of the second phase is to demonstrate the effects and effectiveness of converting Wi-Fi-based packets to other protocols and relaying them through the RPi platform. This is accomplished by monitoring the data loss in the conversion from Wi-Fi to Zigbee and the latency that increases with distance in the conversion from Wi-Fi to Bluetooth. Below is a discussion of the network setup, procedure, and observations.

a. Bluetooth communication: Two laptops and a relay agent (RPi) are the components of the Bluetooth communication setup. The relay agent is compatible with Bluetooth and Wi-Fi interfaces. One of the clients connects wirelessly to the relay agent and serves as the sender.

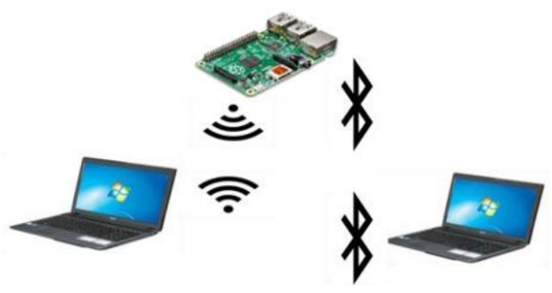


Fig 4. Bluetooth Communication setup.

The other client serves as the receiver and is Bluetooth-connected to the relay agent. The network configuration is shown in Fig 4. The client and server-side algorithms are given below.

Algorithm V*: Client-side Algorithm*

```
Step 1: Create a socket file descriptor using the socket() system call.
Step 2: Create a sockaddr_in structure to store the server information including IP address and port number.
Step 3: Fill in the server information in the sockaddr_in structure.
Step 4: Create a buffer of 1KB size to store the message that will be sent to the server.
Step 5: Use a for loop to send 100 packets of 1KB size to the server.
Step 6: In each iteration of the loop, send a packet to the server using the sendto() system call.
Step 7: Close the socket using the close() system call.
```

Algorithm VI: *Server-side Algorithm*

```
Step 1: Create a socket file descriptor of datagram type using the socket() system call.
Step 2: Create a sockaddr_in structure to store the client and server information, including IP address and port number.
Step 3: Fill in the server information in the sockaddr_in structure.
Step 4: Bind the socket to the sockaddr_in structure using the bind() system call.
Step 5: Create a socket file descriptor of Bluetooth type using the socket() system call.
Step 6: Create a sockaddr_l2 structure to store the client and server information, including Bluetooth address (BD_ADDR), port number.
Step 7: Fill in the server information in the sockaddr_l2 structure.
Step 8: Bind the socket to the sockaddr_l2 structure using the bind() system call.
Step 9: Set the Bluetooth MTU from default value to the required value using struct l2cap_opt, getsockopt() and setsockopt() library.
Step 10: Use listen() for any Bluetooth request to connect to the server.
Step 11: Accept the Bluetooth connection using accept() and store the client-2's Bluetooth address in the sockaddr_l2 structure type variable.
Step 12: Create a buffer of some fixed size which is used to receive the hello message from the client-2 using read() method.
Step 13: Use a for loop to receive 100 packets from client-1 and store in the above create buffer, here recvfrom() method is used to receive Wi-Fi buffer message from the client1, whereas recvfrom() method fetches all the information related to the client1 and stores in the sockaddr_in structure type of client1 and also related to the socket and buffer.
Step 14: Create message fragments according to the Wi-Fi packet size and specified Bluetooth MTU size.
Step 15: Use write() method to send the fragmented messages to the client-2 using established Bluetooth connection and L2CAP protocol.
Step 16: Close both the sockets using close() system call.
```

Algorithm VII: *Client-2 side Algorithm: Bluetooth*

```
Step 1 : Create a socket file descriptor of Bluetooth type using the socket() ystem call.
Step 2: Create a sockaddr_l2 structure to store the client and server information, including Bluetooth address (BD_ADDR), port number.
Step 3 : Fill in the server information in the sockaddr_l2 structure.
Step 4 : Set the Bluetooth MTU from default value to the required value using struct l2cap_opt, getsockopt() and setsockopt() library.
Step 5 : Connect to the server using connect() method by passing above created socket file descriptor and server Bluetooth address.
Step 6 : Send hello message to the server using write() method.
Step 7 : Use a for loop to receive 100 multiplied by number of fragments of messages using read() method from server through established Bluetooth connection and L2CAP protocol.
Step 8 : Close the socket using close() system call.
```

b. *Zigbee Communication:* Two computers and a relay agent are the components of the Zigbee communication configuration. Wi-Fi and Zigbee interfaces are both supported by the relay agent. One of the clients connects wirelessly to the relay agent and serves as the sender. The other client serves as the receiver and is connected through Zigbee to the relay agent. The network configuration is shown in Fig 5.

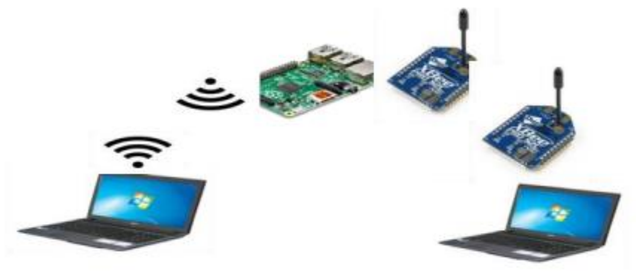


Fig 5. Zigbee Communication Setup.

Algorithm VIII*: Client-side Algorithm*

```
Step 1: Create a socket file descriptor of datagram type using the socket() system call.
Step 2: Create a sockaddr_in structure to store the client and server information, including IP address and port number.
Step 3: Fill in the server information in the sockaddr_in structure.
Step 4: Create a buffer of 1KB size to store the message that will be sent to the server.
Step 5: Fill the buffer with a message.
Step 6: Use a for loop to send 100 packets of 1KB size to the server.
Step 7: In each iteration of the loop, send a packet to the server using the sendto() system call.
Step 8: Close the socket using the close() system call.
```

Algorithm IX : *Server-side Algorithm*

```
Step 1: Create a socket file descriptor of datagram type using the socket() system call.
Step 2: Create a sockaddr_in structure to store the client1, server and client2 information, including IP address and port number.
Step 3: Fill in the server information in the sockaddr_in structure.
Step 4: Bind the above created socket with the sockaddr_in structure type of server.
Step 5: Create a buffer of some fixed size to store the message that will be received from client1 and forwarded to client2.
Step 6: Create a termios structure, for communication on serial port. Configure port for 8-bit data, 1 stop bit, baud rate of 9600, no parity and no hardware or software flow control.
Step 7: Use a for loop to forward 100 packets from client1 to client2 by temporarily storing in the above created buffer, here recvfrom() method is used to receive buffer message from the client1, whereas recvfrom () method fetches all the information related to the client1 and stores in the sockaddr_in structure type of client1, buffer.
Step 8: Add start and finish byte to buffer and forward the message to client2 using serial communication through write () method, where as write() method uses descriptor, buffer and length of buffer to the client2. Continue writing until all received bytes are forwarded.
Step 9: Close the socket using the close () system call.
```

Algorithm X: *Client 2-side Algorithm*

```
Step 1: Create a socket file descriptor of datagram type using the socket () system call.
Step 2: Create a termios structure for communication on a serial port using the tcgetattr()   system call.
Step 3: Configure the port for 8 bit data, 1 stop bit, no parity, no hardware and software flow control and baud rate of 9600 using the tcsetattr() system call.
Step 4: Create a buffer of some fixed size, such as 1024 bytes, using the malloc () function.
Step 5: Send a hello message to the server using the write () system call.
Step 6: Use a do while loop to receive packets and store in the above created buffer.
Step 7: In the do while loop, use the read () system call to receive a packet from the server.
Step 8: If the start byte is received, keep saving bytes until the finish byte is received.
Step 9: When the finish byte is received, calculate the count of received bytes using the strlen () function.
Step 10: Display the summary of the received data, such as the count of received bytes and the data itself.
```

Phase 3: In the third phase, we put the protocol conversion (Wi-Fi to Bluetooth and Wi-Fi to Zigbee) into practice and investigated how parallel transmissions were affected. A Raspberry Pi served as the relay gateway.

A relay agent (RPi) and two computers make up the setup. The Wi-Fi, Bluetooth, and Zigbee interfaces are supported by the relay agent. One of the clients connects wirelessly to the relay agent and serves as the sender. The second client, which connects to the relay agent through Bluetooth and Zigbee, serves as the receiver. Two processes are active at the receiver, one of which is receiving Bluetooth signals and the other Zigbee packets. Fig 6 below shows how the network is set up for parallel transmission.

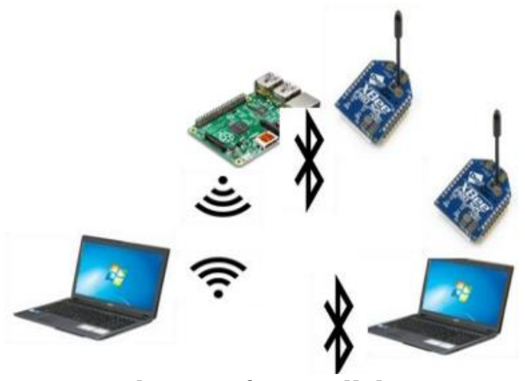


Fig 6. Network Setup for Parallel Transmission

```
Algorithm XI: Client-side Algorithm
Step 1: Create a socket file descriptor of datagram type using the socket () system call.
Step 2: Create a sockaddr_in structure type to store the server information including IP address, port number using the sockaddr_in() function.
Step 3: Fill server information in the above created structure using the htons () and inet_pton() functions.
Step 4: Create a buffer of some fixed size, such as 1024 bytes, using the malloc () function.
Step 5: Fill in the buffer with the message to be sent to the server.
Step 6: Use a for loop to send 100 packets, each of size 1KB, to the server using the sendto () system call.
Step 7: Close the socket using the close() system call.
Step 8: Create message fragments according to the Wi-Fi packet size and specified Bluetooth MTU size.
Step 9: Use the write() system call to send the fragmented messages to the client-2 using established Bluetooth connection and L2CAP protocol.
Step 10: Close both the sockets using the close() system call.
```

```
Algorithm XII: Server-side Algorithm
Step 1: Create two socket file descriptors of type datagram and Bluetooth using the socket () system call.
Step 2: Create a sockaddr_in structure type to store the client and server information including IP address, port number using the sockaddr_in() function.
Step 3: Create a sockaddr_l2 structure type to store the client and server information including Bluetooth address (BD_ADDR), port number using the sockaddr_l2()
        function.
```

```
Step 4: Fill server information in the above created structures using the htons() and inet_pton() functions for sockaddr_in, and bt_addr_t functions for sockaddr_l2.
Step 5: Bind the above created sockets with the sockaddr_in structure type and sockaddr_l2 structure type of server using the bind() system call.
Step 6: Set the Bluetooth MTU from default value to the required value using struct l2cap_opt, getsockopt() and setsockopt() library functions.
Step 7: Use listen() for any Bluetooth request to connect to the server using the listen () system call.
Step 8: Accept the Bluetooth connection using accept() and store the client-2's Bluetooth  address in the sockaddr_l2 structure type variable.
Step 9: Create a termios structure for communication on serial port using the tcgetattr () system call.
Step 10: Configure the port for 8 bit data, 1 stop bit, baud rate of 9600, no parity and no hardware or software flow control using the tcsetattr() system call.
Step 11: Create a buffer of some fixed size, such as 1024 bytes, using the malloc () function.
Step 12: Use a for loop to receive 100 packets from client-1 and store in the above created buffer, here recvfrom() method is used to receive Wi-Fi buffer message from the client1, whereas recvfrom() method fetches all the information related to the client1 and stores in the sockaddr_in structure type of client1 and also relatd to the socket and buffer.
Step 13: Forward packets on bluetooth and zigbee as per desired ratios.
Step 14: For Bluetooth:
    - Create message fragments according to the Wi-Fi packet size and specified Bluetooth MTU size.
    - Use the write() system call to send the fragmented messages to the client-2 using established Bluetooth connection and L2CAP protocol.
Step 15: For zigbee:
    - Add start and finish byte to buffer and forward the message to client2 using serial communication through write() method, where as write() method uses descriptor, buffer and length of buffer to the client2. Continue writing until all received bytes are forwarded.
Step 16: Close both the sockets using the close() system call.
```

Algorithm XIII: *Client-2 side Algorithm: Bluetooth*

```
Step 1: Create a socket file descriptor of Bluetooth type using the socket() system call.
Step 2: Create a sockaddr_l2 structure type to store the server information including Bluetooth address (BD_ADDR), port number using the sockaddr_l2() function.
Step 3: Fill server information in the above created structure using the bt_addr_t functions.
Step 4: Set the Bluetooth MTU from default value to the required value using struct l2cap_opt, getsockopt() and setsockopt() library functions.
Step 5: Connect to the server using connect () by passing above created socket file descriptor and server Bluetooth address.
Step 6: Send hello message to the server using write () method.
```

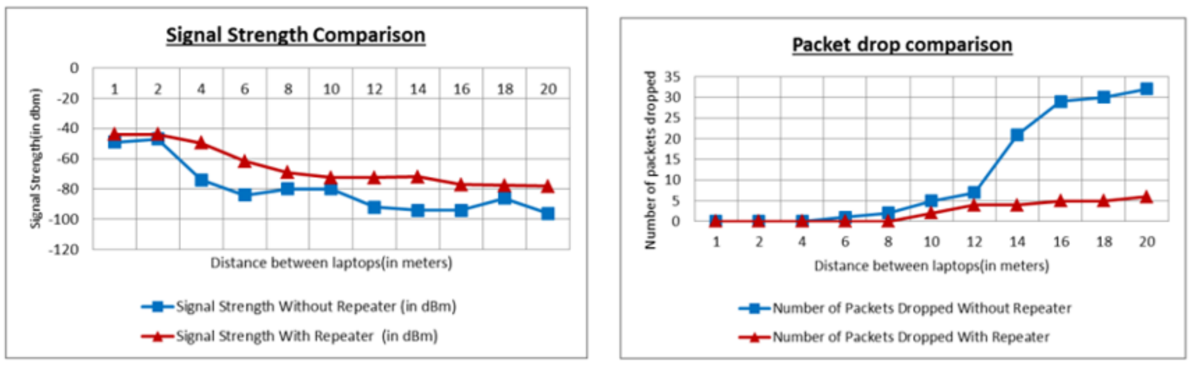
Step 7: Use a for loop to receive 100 multiplied by number of fragments of messages using read () method from server through established Bluetooth connection and L2CAP protocol.
Step 8: Close the socket using the close () system call.

Algorithm XIV: *Client-2 side Algorithm: Zigbee*

Step 1: Create a socket file descriptor of datagram type using the socket () system call.
Step 2: Create a termios structure for communication on serial port using the tcgetattr() system call.
Step 3: Configure the port for 8 bit data, 1 stop bit, no parity, no hardware and software flow control using the tcsetattr () system call.
Step 4: Create a buffer of some fixed size, such as 1024 bytes, using the malloc () function.
Step 5: Fill the buffer with the message to be sent to the server.
Step 6: Send a hello message to the server using the write () system call.
Step 7: Use a do while loop to receive packets from the server and store in the above created buffer.
Step 8: In the do while loop, use the read () system call to receive a packet from the server.
Step 9: If the start byte is received, keep saving bytes until the finish byte is received.
Step 10: When the finish byte is received, calculate the count of received bytes using the strlen () function.
Step 11: Display the summary of the received data, such as the count of received bytes and the data itself.

5 Results and Discussion

Without utilizing a Raspberry Pi repeater, 100 packets were forwarded from the client to the server. The number of missed packets and the client's received Wi-Fi signal strength were recorded for ten different distances. As demonstrated in Fig 7, after adding a Raspberry Pi to act as a Wi-Fi repeater between clients 1 and 2, packet drop and received Wi-Fi signal strength are recorded.



(a) **(b)**

Fig 7. Signal Strength 7 (a) and Packet drop comparison 7 (b)

a. *Transmission of Wi-Fi packets after relaying and getting converted at the server using Bluetooth protocol*:

A laptop acting as client-1 will transmit 100 Wi-Fi packets using the UDP protocol to a Raspberry Pi acting as server. The Raspberry Pi converts the packets using the built-in Bluetooth interfaces and sends the modified pieces to a second laptop acting as client 2. For the server (Raspberry Pi) to know the address of client2, client-2 will first use Bluetooth protocol to send a "hello" message to the server. Client1 to Client2 packet transfer can now be started. Socket programming is used to implement this transfer procedure. Client-1 sends 100 packets to Client-2. Client-1 will transmit 100 Wi-Fi packets to Client-2 via Raspberry Pi (Wi-Fi Bluetooth repeater), where the packets are divided based on Bluetooth MTU. Fig 8 shows the latency associated with switching from the Wi-Fi protocol to Bluetooth's MTU and with Bluetooth propagation.

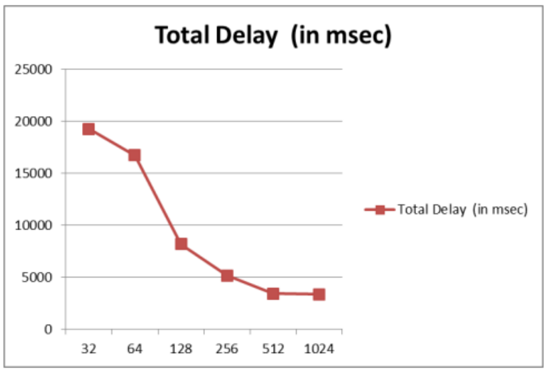


Fig 8. Conversion delay graph Client1 to Client2.

b. *Transmission of Wi-Fi packets after relaying and getting converted at the server using Zigbee protocol:*

A Raspberry Pi (repeater) will be used to repeat 100 packets from one laptop (client-1) to another laptop (client-2). Serial communication and socket programming are used to implement the transmission procedure. From client-1 to client-2, 100 packets are sent. Through the Raspberry Pi (Wi-Fi-Zigbee repeater), Client-1 will transmit 100 packets to Client-2, another laptop. The delay at various distances is shown in Fig 9.

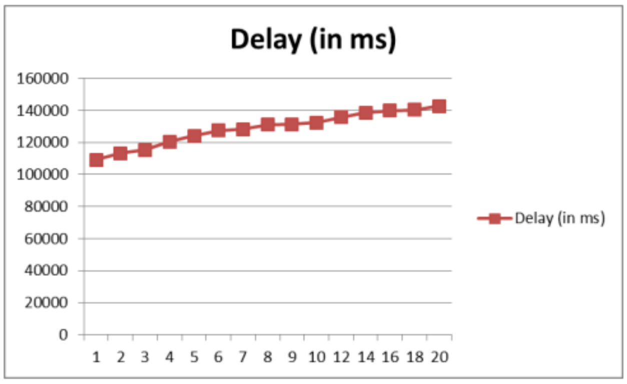


Fig 9. Server to Client2 Delay.

100 packets will be sent from one laptop (client-1) via a Raspberry Pi repeater to another laptop (client-2). Under varied ratios, the Rpi Repeater successfully transmits packets to client-2 over Bluetooth and Zigbee. Two processes are running in Client-2, one listening on Bluetooth and the other on Zigbee. To accomplish the transmission procedure, serial communication and socket programming are used. Client-1 sends 100 packets to Client-2. One laptop (client-1) will use a Raspberry Pi to deliver 100 Wi-Fi packets to another laptop (client-2) at varying Bluetooth and Zigbee packet ratios. According to Bluetooth's MTU, packets have been broken up for Bluetooth. The header and footer have been added to the data for Zigbee. The delay of protocol conversion and propagation from Wi-Fi to Bluetooth and Zigbee under different Bluetooth-Zigbee packet ratios is shown Fig 10.

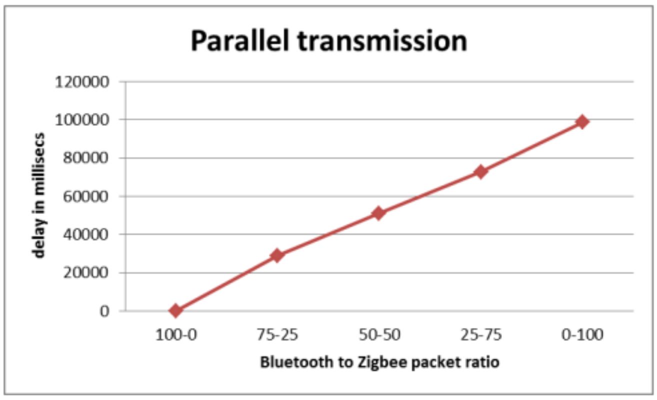


Fig 10. Delay for Bluetooth-Zigbee Ratio.

6 Conclusion

Three wireless technologies—Wi-Fi, Zigbee, and Bluetooth—have been examined in this paper. Wi-Fi enables quick long-distance communication. For rapid, short-distance communication, Bluetooth technology excels. Zigbee, on the other hand, offers somewhat faster, lower-power communication across longer distances. So, depending on the situation, IOT devices could use Bluetooth or Zigbee. Relay agents that offer protocol conversion can connect to Bluetooth and Zigbee IOT devices using Wi-Fi technology, which operates on IP. The data transfer range between the clients is improved by the repeater. The interoperability of heterogeneous networks is supported using protocol conversion. However, an intriguing future extension will be researching the protocol performance characteristics and protocol conversion for emerging technologies/platforms for various application requirements.

References

[1] ITU, The Internet of Things, 2005, https://www.itu.int/net/wsis/tunis/newsroom/stats/The-Internet-of-Things-2005.pdf, Retrieved on September 15, 2018.

[2] G. Ranganathan, R. Bestak & C. O. Chow (2020). Computational Approaches in Cloud Based IoT. *Journal of Internet Technology*, 21(1), 147- 148.

[3] M. Soliman, T. Abiodun, T. Hamouda, J. Zhou, & C.-H. Lung (2013). Smart Home: Integrating Internet of Things with Web Services and Cloud Computing. *In Proceedings of International Conference on Cloud Computing Technology and Science* (317-320). IEEE.

[4] P. Sethi & S. R. Sarangi (2017). Internet of Things: Architectures, Protocols, and Applications. *Journal of Electrical and Computer Engineering* , 1-26.

[5] W. T. Cho, Y. W. Ma, & Y. M. Huang (2015). A smart socket-based multiple home appliance recognition approach over IoT architecture. *Journal of Internet Technology*, 16(7), 1227-1238.

[6] X. J. Chen, B. D. Chen, X. M. Jiang, X. B. Chen, & W. H. Cai (2016). Improved Cloud Computing Architecture for the Internet of Things, *Journal of Internet Technology*, 17(4), 683-693.

[7] H. Derhamy, J. Eliasson, & J. Delsing. (2017). IoT interoperability— On-demand and low latency transparent multi-protocol translator. *IEEE Internet Things Journal*. 4(5), 1754–1763.

[8] R. Narayanan & C. S. R. Murthy (2017). A probabilistic framework for protocol conversions in IIoT networks with heterogeneous gateways. *IEEE Commun. Lett*, 21(11), 2456–2459.

[9] L. Angrisani, M. Bertocco, D. Fortin, & A. Sona (2008). Experimental study of coexistence issues between IEEE 802.11b and IEEE 802.15.4 wireless networks. *IEEE Trans. Instrum. Meas.,* 57(8), 1514–1523.

[10] Alshehri AA, Lin SC, & Akyildiz IF(2017). Optimal energy planning for wireless self-contained sensor networks in oil reservoirs. In: Proc. International Conference on Communications (1-7). IEEE.

[11] G. Chunhong, Z. Yu, Y. Qingwei, & W. Xiaodong (2014). A ZigBee and Bluetooth protocol converter based on multi-sinks wireless sensor network, J. Netw, 9(7), 1854–1860.

[12] J. Pope & R. Simon (2013). The impact of packet fragmentation on Internet-of-Things enabled systems. *In Proc. Int. Conf. Inform. Technol. Interfaces* (13–18). IEEE

[13] J. Shen, A. Wang, C. Wang, P. C. K. Hung, & C.-F. Lai (2017). An efficient centroid-based routing protocol for energy management in WSN-assisted IoT. *IEEE Access*,18469–18479.

[14] D. Yuan, S. S. Kanhere, & M. Hollick (2017). Instrumenting wireless sensor networks—A survey on the metrics that matter. Pervasive Mobile Comput., 37(1), 45–62.

[15] Z.-Y. Ai, Y.-T. Zhou, & F. Song (2018). A smart collaborative routing protocol for reliable data diffusion in IoT scenarios, *Sensors*, 18 (6), 1926–1947.

[16] L. Zhao, A. Al-Dubai, X. Li, G. Chen, and G. Min (2017). A new efficient cross-layer relay node selection model for wireless community mesh networks, *Comput. Elect. Eng.*, 61(10), 361–372.

[17] T. M. Behera, U. C. Samal, & S. K. Mohapatra (2018). Energy-efficient modified LEACH protocol for IoT application. *IET Wireless Sensor Syst*., 8 (5), 223–228, IET.

[18] F. A. Khan, A. Ahmad, & M. Imran (2020). Energy optimization of PR-LEACH routing scheme using distance awareness in Internet of Things networks, *Int. J. Parallel Program,* 48(2), 244–263.

[19] H.-S. Kim, J. Ko, D. E. Culler, & J. Paek (2017). Challenging the IPv6 routing protocol for low-power and lossy networks (RPL): A survey. *IEEE Commun. Surveys Tuts.,* 19(4), 2502–2525.

[20] B. Ghaleb, A. Y. Al-Dubai, E. Ekonomou, I. Romdhani, Y. Nasser, & A. Boukerche (2018). A novel adaptive and efficient routing update scheme for low-power lossy networks in IoT, *IEEE Internet Things J,* 5(6), 5177–5189.

[21] C. Gomez, J. Paradells, C. Bormann, & J. Crowcroft (2017). From 6LoWPAN to 6Lo: Expanding the universe of IPv6-supported technologies for the Internet of Things. *IEEE Commun. Mag.,* 55(12), 148–155.

[22] T. Gomes, F. Salgado, S. Pinto, J. Cabral, & A. Tavares (2018). "A 6LoWPAN accelerator for Internet of Things endpoint devices," *IEEE Internet Things J*, 5(1), 371–377.

[23] R. Narayanan, M. Srinivasan, S. A. Karthikeya, 7 C. S. R. Murthy (2017). A novel fairness-driven approach for heterogeneous gateways' link scheduling in IoT networks. *In Proc. Int. Conf. Commun*. (1–7). IEEE

[24] H. Derhamy, J. Eliasson, & J. Delsing (2017). IoT interoperability— On-demand and low latency transparent multi-protocol translator, *IEEE Internet Things J.,* 4(5), 1754–1763.

[25] M. Alicherry, R. Bhatia, & L. E. Li (2005). Joint channel assignment and routing for throughput optimization in multi-radio wireless mesh networks. I*n Proc. ACM Annu. Int. Conf. Mobile Comput. Netw*.(58–72). *ACM*

[26] S. Sanshi & C. D. Jaidhar (2019). Enhanced mobility aware routing protocol for low power and lossy networks. *Wireless Netw.,* 25(4), 1641–1655.

[27] A. Hava, G.-M. Muntean, & J. Murphy (2014). ABI: A mechanism for increasing video delivery quality in multi-radio wireless mesh networks. *In Proc. IEEE Int. Symp. Broadband Multimedia Syst. Broadcasting*(1–6). IEEE.

[28] M. Shojafar, S. Abolfazli, H. Mostafaei, & M. Singhal (2015). Improving channel assignment in multi-radio wireless mesh networks with learning automata. *Wireless Pers. Commun.,* 82(1), 61–80.

[29] X. Deng, J. Luo, L. He, Q. Liu, X. Li, & L. Cai (2019). Cooperative channel allocation and scheduling in multi-interface wireless mesh networks. *Peer to Peer Netw. Appl.,*12(1), 1–12.

[30] R. Diamant, P. Casari, F. Campagnaro, O. Kebkal, V. Kebkal, & M. Zorzi (2018). Fair and throughput-optimal routing in multimodal underwater networks. *IEEE Trans. Wireless Commun.,*17(3), 1738–1754.

[31] A. Banik & A. Majumder (2019). Classification of channel allocation schemes in wireless mesh network. *In Algorithms, Methods, and Applications in Mobile Computing and Communications (*65–92*).* Hershey, PA, USA: IGI Global.

[32] R. Narayanan and C. S. R. Murthy (2017). A probabilistic framework for protocol conversions in IIoT networks with heterogeneous gateways. *IEEE Commun. Lett.,* 21(11), 2456–2459.

[33] L. Angrisani, M. Bertocco, D. Fortin, & A. Sona (2008). Experimental study of coexistence issues between IEEE 802.11b and IEEE 802.15.4 wireless networks. I*EEE Trans. Instrum. Meas.,* 57(8), 1514–1523.

[34] Seohyang Kim, Hyung-Sin Kim, & Chongkwon Kim. (2019). ALICE: Autonomous link-based cell scheduling for TSCH. *In Proceedings of the 2019 18th ACM/IEEE International Conference on Information Processing in Sensor Networks* (121-132). IEEE.

[35] T. van der Lee, G. Exarchakos, & A. Liotta (2017). Distributed TSCH scheduling: A comparative analysis. *In Proceedings of the 2017 IEEE International Conference on Systems, Man, and Cybernetics* (3517—3522), IEEE.

[36] J. Pope & R. Simon (2013). The impact of packet fragmentation on Internet-of-Things enabled systems. *In Proc. IEEE Int. Conf. Inform. Technol. Interfaces* (13–18), IEEE.

[37] Alexander Clemm. Network Management Fundamentals. Cisco Press, first edition, 2006.

[38] Ian Sommerville. Software Engineering. 2011.
[39] Hennie Huijgens, Arie van Deursen, and Rini van Solingen (2016). Success Factors in Managing Legacy System Evolution: A Case Study. *Proceedings of the International Workshop on Software and Systems Process*(96–105).
[40] Reuben L Mathule and Billy M Kalema (2016). User Acceptance of Legacy Systems Integration (1–8).
[41] Bruce R Elbert. The Satellite Communication Ground Segment and Earth Station Handbook. Artech House, Inc., Norwood, MA, 2001.

Notes on contributors

Dr. Geetabai S. Hukkeri is Assistant Professor in the Department of Computer Science and Engineering at the Manipal Institute of Technology, Bengaluru, India. Her research interests include Artificial Intelligence, Deep Learning, Machine Learning, Big Data, Computer Vision, Computer Networks and Multimedia Information Retrieval. She has published several papers in International Journals, International Conferences, and International Book Chapters. She had published a book on “Understanding Big Data Technologies-A simple Approach”. She is a member of ACM. She holds a Ph.D. Degree in Computer Science and Engineering from Visweswaraya Technological University, Belagavi, India.

Dr. Shilpa Ankalaki is currently working as Assistant Professor, Department of Computer Science and Engineering, Manipal Institute of Technology Bengaluru, Manipal Academy of Higher Education, Manipal. Her research interests include Machine Learning, Deep Learning, Data Mining and Artificial Intelligence applications. She holds a Ph.D. Degree in Computer Science and Engineering from the Visweswaraya Technological University, Belagavi, India.

Corresponding author: Shilpa Ankalaki (shilpa.ankalaki@manipal.edu)

Dr. R H Goudar is currently working as an Associate Professor, Department of Computer Science and Engineering, Visvesvaraya Technological University, Belagavi, Karnataka. He has published over 150 papers in International Journals, International Conferences, and International Book Chapters. He is a Reviewer of IEEE Transactions on Knowledge and Data Engineering and Editor/ Editorial Member of various Computer Science Journals. He has received various awards, such as the Outstanding Faculty Award, the Research Performance Award, the Young Research Scientist Award from VGST Karnataka, and the Eminent Engineer Award from the Honorable Chief Minister of Karnataka.

Dr. Lingaraj Hadimani is currently working as Associate Professor and Head (Computer Science and Business Systems) at KIT's College of Engineering, Kolhapur, Maharashtra. He has more than 18 years of teaching experience at college level in different countries and 3 years full time research experience at Council of Scientific & Industrial Research (CISR) organization. Recipient of AICTE-INAE Teachers Research Fellowship (TRF) to purse full time PhD at CSIR-CSIO, Chandigarh.