Evolution of Information, Communication and Computing Systems (EICCS)
https://publisher.uthm.edu.my/bookseries/index.php/eiccs

Implementation and Simulation of UDP Client-Server Environment using Contiki Cooja Simulator

Muhammad Asif Khan[1], Mohd Anuaruddin Ahmadon[2], Natasha Amira Abdul Rauf[3], Abang Muhamad Zaid[3], Abd Kadir Mahamad[3*], Sharifah Saon[3], Nik Shahidah Afifi Md Taujuddin[3], and Ansar Jamil[3]

[1]Qatar Mobility Innovations Center,
Qatar University, Doha, QATAR

[2]Graduate School of Science and Technology for Innovation,
Yamaguchi University, Yamaguchi, 755-8611, JAPAN

[3]Faculty of Electrical and Electronic Engineering,
Universiti Tun Hussein Onn Malaysia, Batu Pahat, 86400, MALAYSIA

*Corresponding Author

Abstract:
This paper presents a simulation and evaluation of a UDP (User Datagram Protocol) client-server model using the Cooja simulator and Contiki OS System. The goal is to establish communication between the router, client, and server to monitor the data transmission and reception through a web server. The UDP protocol allows for fast and efficient data transfer without the need for a pre-established virtual path. The simulation results demonstrate the feasibility of utilizing UDP for real-time services and live communication. The findings highlight the flexibility of UDP in selecting multiple paths for data transmission, enhancing robustness and reliability. Further research can explore optimization techniques for UDP-based communication in diverse Internet of Things (IoT) networks.

Keywords:
UDP protocol · UDP Client-Server · RPL border router · Contiki Cooja · TCP protocol

1. Introduction

UDP (User Datagram Protocol) is a transport layer protocol that operates within the Internet Protocol (IP) suite of network protocols. UDP serves as a connectionless and lightweight alternative to TCP (Transmission Control Protocol) and is widely employed in network communication. UDP is renowned for its simplicity and efficiency, making it well-suited for specific types of applications where low latency and reduced overhead take precedence over reliable data delivery [1].

The foremost characteristic of UDP is its connectionless nature. While TCP establishes a persistent connection between sender and receiver, UDP treats each datagram or packet as an independent data unit. This lack of connection setup and teardown overhead grants UDP a faster and lighter-weight nature

*Corresponding author: *kadir@uthm.edu.my* compared to TCP. It proves exceptionally advantageous in situations where real-time data transmission holds paramount importance, and the reliability guarantees provided by TCP are unnecessary.

UDP finds common usage in scenarios that demand low-latency communication. Real-time applications such as video streaming, online gaming, and Voice over IP (VoIP) prioritize immediate data transmission over-delivering every individual packet [2]. Small delays or sporadic packet loss may be tolerable in such cases as the primary focus is sustaining a continuous data flow. UDP's connectionless and lightweight design allows for expedited data transmission, resulting in lower latency and enhanced responsiveness in real-time scenarios [3].

Another advantage of UDP is its reduced overhead. TCP ensures reliable data delivery through mechanisms like error detection, retransmission, and flow control. While these features guarantee data reliability, they also introduce additional processing and network resource overhead. In contrast, UDP does not incorporate these mechanisms by default, leading to reduced overhead. Consequently, UDP is well-suited for applications prioritizing minimal processing and resource utilization.

However, it is important to acknowledge that UDP's lack of reliability guarantees can pose limitations in certain contexts. Without error detection and recovery mechanisms, there is no assurance that packets will reach their destination or be received in the correct order. In situations where reliable data delivery is critical, UDP-based applications must implement their own error detection and recovery mechanisms. This allows developers to customize the behavior of UDP-based applications based on their specific requirements and strike an appropriate balance between reliability and performance.

1.1 UDP in Contiki OS

Contiki OS is an open-source operating system designed explicitly for resource-constrained IoT devices. It offers a lightweight and energy-efficient platform for developing IoT applications, making it highly suitable for a wide range of devices, such as sensor nodes and embedded systems. The modular architecture of Contiki OS allows for efficient utilization of memory and energy, enabling IoT devices to operate effectively with limited resources. One of the notable features of Contiki OS is built-in support for various networking protocols, including UDP. The integration of UDP in Contiki OS makes it well-suited for implementing UDP client-server communication in IoT environments.

Cooja provides a simulated environment where developers can emulate and assess the behavior of their Contiki-based applications without the need for physical devices. This brings several advantages in saving cost and time by eliminating the need for physical hardware. Moreover, the Cooja simulator provides a controlled and reproducible environment for testing and evaluating the performance of UDP client-server communication in the development process.

Hence, by leveraging the capabilities of Contiki OS and the Cooja simulator, researchers and developers can effectively design, develop, and evaluate UDP client-server systems in IoT environments [4]. They can ensure efficient resource utilization reliable communication, and address the challenges posed by resource-constrained devices. The combination of Contiki OS and Cooja provides a comprehensive framework for developing and evaluating UDP-based IoT applications, contributing to advancing IoT technologies.

2. Literature Review

Figure 1 shows the UDP client-server model architecture. The UDP (User Datagram Protocol) client-server model architecture is a widely used model in network communication. It comprises two essential components: the UDP client and the UDP server. Each component performs specific functions that enable data exchange between them [5].

The UDP client is responsible for initiating communication with the server. Its primary functionalities include creating a UDP socket, which serves as an endpoint for sending and receiving UDP datagrams. By specifying the IP address and port number of the UDP server, the client establishes a connection to the desired server. It can then send UDP datagrams by encapsulating the data it wishes to transmit into packets. These datagrams contain the destination server's address and port, ensuring they are routed correctly. Once a datagram is sent, the client waits for a response. Upon receiving a response, the client processes the data contained within and takes appropriate actions based on the application's requirements [6].

On the other hand, the UDP server waits for incoming requests from clients and responds accordingly. It begins by creating a UDP socket to establish communication. By binding its socket to a specific port on its host machine, the server can listen for incoming datagrams on that port. The server enters a continuous loop, waiting for datagrams from clients. When a datagram arrives, the server extracts the data contained within and processes it according to the application's logic. Upon receiving a request from a client, the server formulates a response by encapsulating it into a UDP datagram. This response is then sent back to the client's address and port, as specified in the received datagram.

Meanwhile, to handle multiple clients simultaneously, the UDP server can employ concurrent or multithreaded programming techniques. This capability allows the server to respond to requests from multiple clients concurrently, enhancing its scalability and responsiveness.

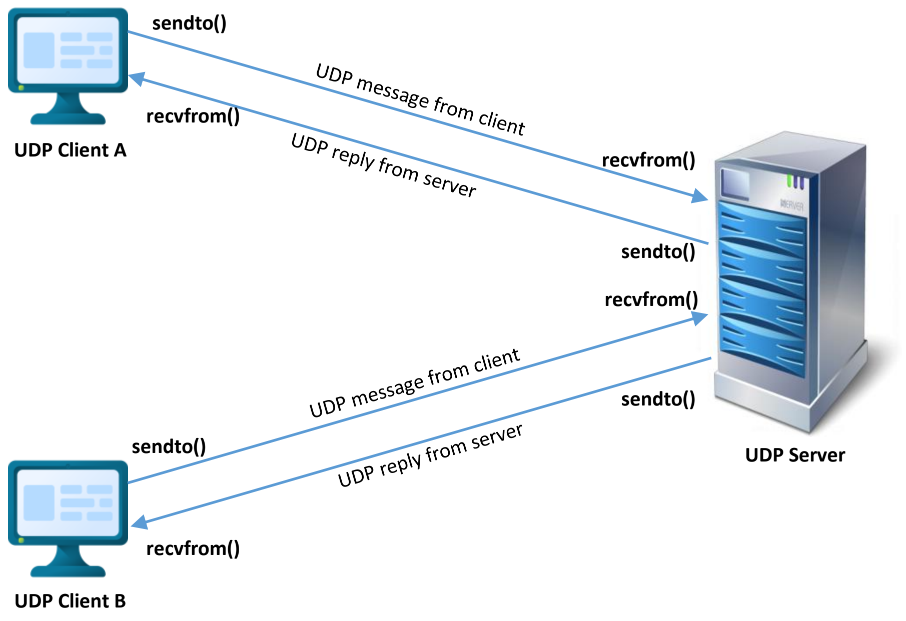


Figure 1: UDP Client-server model architecture

2.1 RPL Border Router

A RPL Border Router serves as the crucial interface between an RPL network and external networks, particularly the Internet. Figure 2 shows the RPL border router architecture. Its primary purpose is to establish connectivity and facilitate communication between the RPL network and other networks or devices that exist beyond its boundaries. The RPL Border Router performs a diverse range of crucial functions. Firstly, it acts as a bridge, establishing interconnection between the RPL and external networks

[7]. This enables seamless communication between devices within the RPL network and devices or services located outside, including cloud servers or other IoT networks.

Additionally, the RPL Border Router has the capability to engage in protocol translation. This ensures compatibility between the protocols used within the RPL network and those employed in external networks. It promotes efficient interoperability by facilitating seamless data exchange and communication between the RPL network and other networks that may operate using different protocols. Lastly, the RPL Border Router is responsible for routing and forwarding packets between the RPL network and external networks. By evaluating routing metrics and policies, it determines the most suitable paths for data transmission, ensuring efficient packet forwarding and maintaining effective network performance [8].

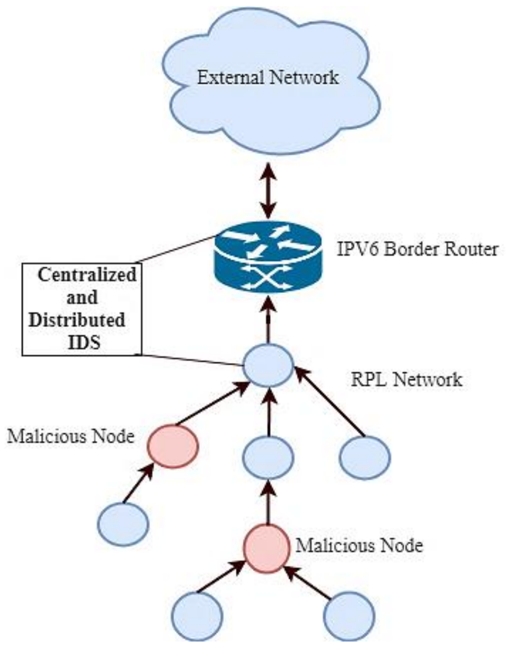


Figure 2: RPL border router [9]

3. Methodology

Several methods were employed to develop a UDP client-server model with a built-in web server to view the simulation results. The most important methods involved mote setup, topology design, and router enabling, as shown in Figure 3.

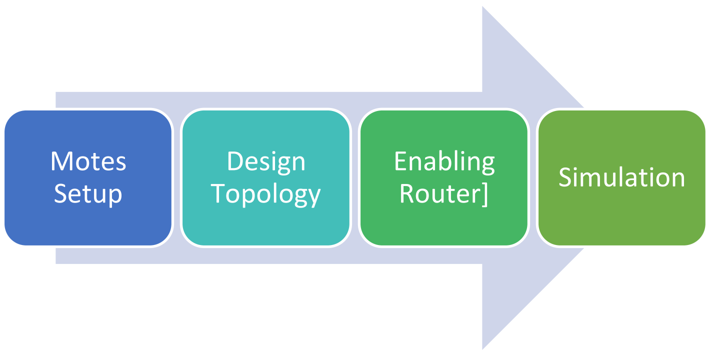
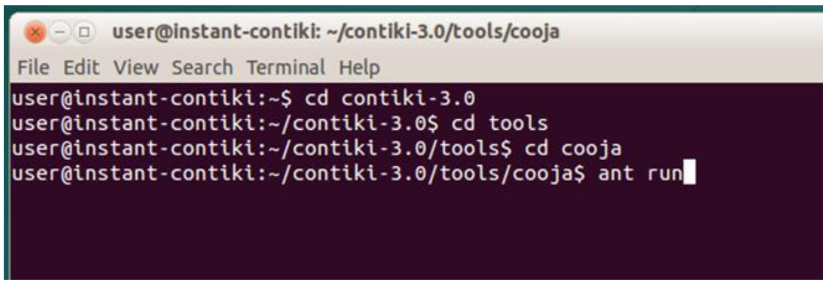


Figure 3: Development of UDP client-server model in Cooja

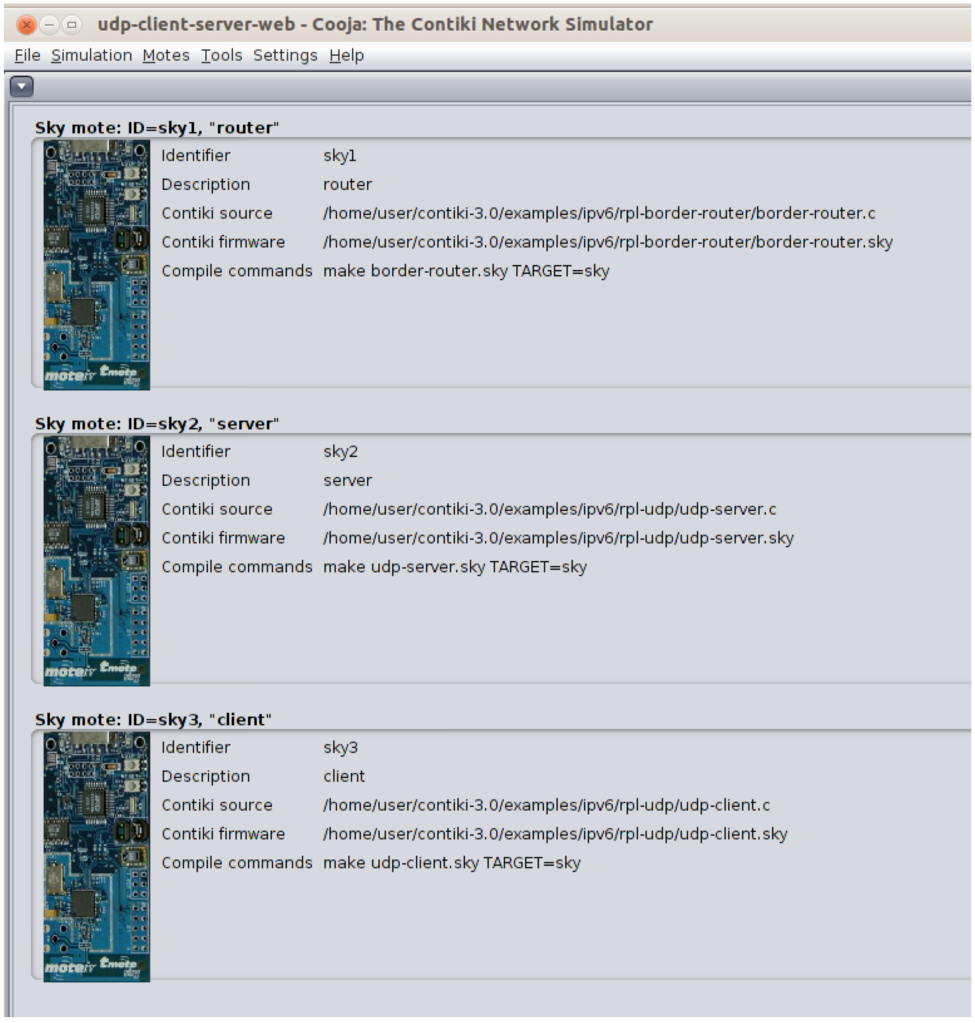
Launching Cooja involves initiating the Cooja simulator, which serves as a platform for emulating and evaluating Contiki-based IoT applications. Figure 4 shows the command to launch the Cooja simulator.


user@instant-contiki: ~/contiki-3.0/tools/cooja
File Edit View Search Terminal Help

```
user@instant-contiki:~$ cd contiki-3.0
user@instant-contiki:~/contiki-3.0$ cd tools
user@instant-contiki:~/contiki-3.0/tools$ cd cooja
user@instant-contiki:~/contiki-3.0/tools/cooja$ ant run
```

Figure 4: Terminal command launching Cooja simulator

Once Cooja is operational, a new simulation project can be created by selecting the corresponding option from the menu, enabling the definition of simulation parameters and settings in accordance with specific requirements. Subsequently, the motes or nodes participating in the simulation need to be chosen. In the context of the UDP client-server model, it is customary to select three types of motes: the UDP client mote(s), the UDP server mote, and the RPL Border Router mote. Figure 5 shows the mote type and the specification of their roles as either clients, servers, or routers. ID type 1 for the border, 2 for UDP server mote, and 3 until ID type 7 are UDP client motes.


udp-client-server-web - Cooja: The Contiki Network Simulator
File Simulation Motes Tools Settings Help

Sky mote: ID=sky1, "router"

Identifier	sky1
Description	router
Contiki source	/home/user/contiki-3.0/examples/ipv6/rpl-border-router/border-router.c
Contiki firmware	/home/user/contiki-3.0/examples/ipv6/rpl-border-router/border-router.sky
Compile commands	make border-router.sky TARGET=sky

Sky mote: ID=sky2, "server"

Identifier	sky2
Description	server
Contiki source	/home/user/contiki-3.0/examples/ipv6/rpl-udp/udp-server.c
Contiki firmware	/home/user/contiki-3.0/examples/ipv6/rpl-udp/udp-server.sky
Compile commands	make udp-server.sky TARGET=sky

Sky mote: ID=sky3, "client"

Identifier	sky3
Description	client
Contiki source	/home/user/contiki-3.0/examples/ipv6/rpl-udp/udp-client.c
Contiki firmware	/home/user/contiki-3.0/examples/ipv6/rpl-udp/udp-client.sky
Compile commands	make udp-client.sky TARGET=sky

Figure 5: Motes type

Once all the configurations are set, the simulation can be executed in Cooja. Figure 6 shows the Cooja layout simulator. This allows for observing the behavior of the UDP client-server model within the simulated environment. Monitoring communication between client and server motes, tracking UDP datagram exchanges, and analyzing the obtained results become feasible. The green, orange, and purple nodes referred to a router, UDP server, and UDP clients.

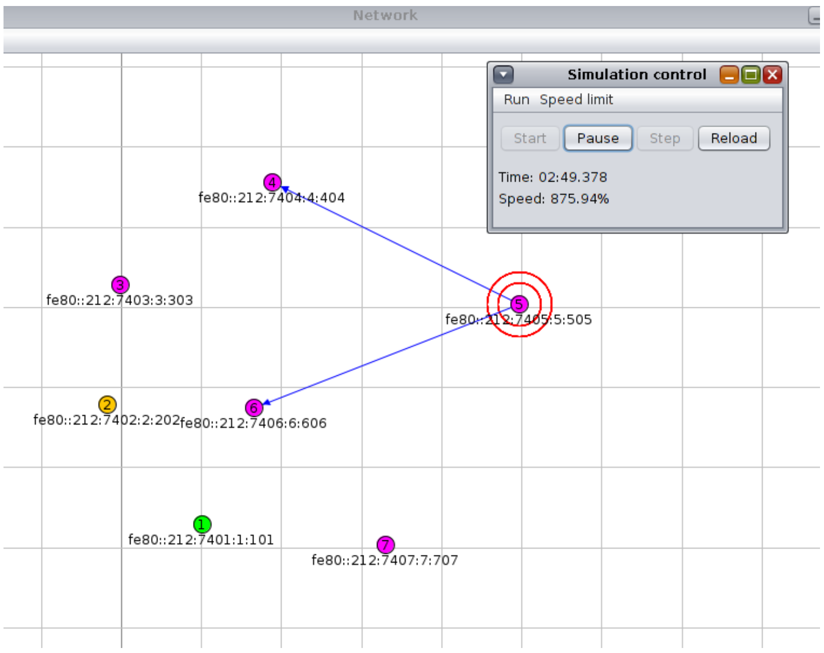
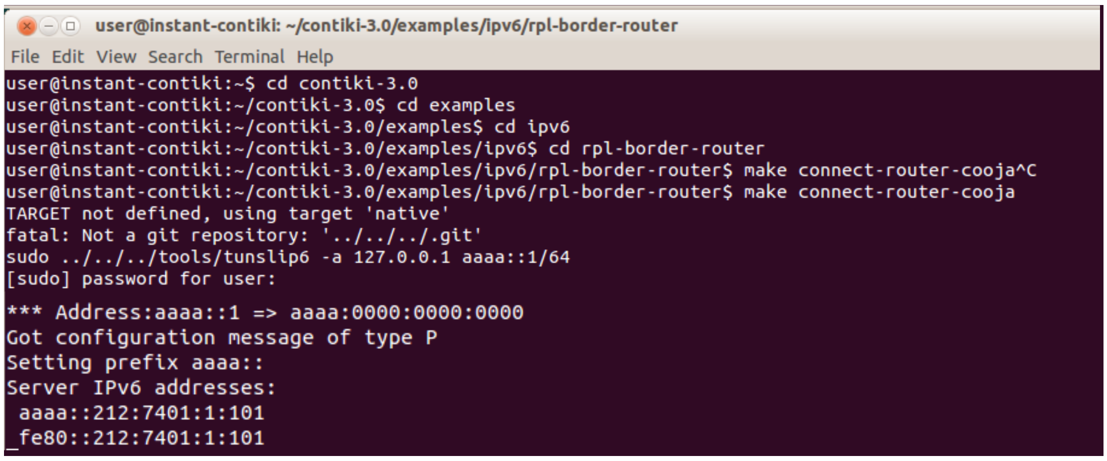


Figure 6: Start simulation

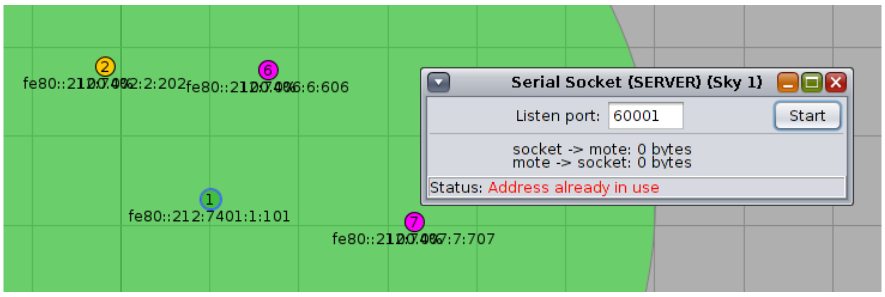
However, at that point, the communication between the RPL border router and the UDP client and UDP server had not yet been established. In order to enable communication with the web server, several steps needed to be performed. Figure 7 illustrates the instructions to enable the RPL border router in the terminal. By executing the command "make connect-router-Cooja," The terminal displayed the public and private IP addresses. The web server header used the public IP address to establish communication with the RPL border router.


user@instant-contiki: ~/contiki-3.0/examples/ipv6/rpl-border-router
File Edit View Search Terminal Help

```
user@instant-contiki:~$ cd contiki-3.0
user@instant-contiki:~/contiki-3.0$ cd examples
user@instant-contiki:~/contiki-3.0/examples$ cd ipv6
user@instant-contiki:~/contiki-3.0/examples/ipv6$ cd rpl-border-router
user@instant-contiki:~/contiki-3.0/examples/ipv6/rpl-border-router$ make connect-router-cooja^C
user@instant-contiki:~/contiki-3.0/examples/ipv6/rpl-border-router$ make connect-router-cooja
TARGET not defined, using target 'native'
fatal: Not a git repository: '../../../.git'
sudo ../../../tools/tunslip6 -a 127.0.0.1 aaaa::1/64
[sudo] password for user:
*** Address:aaaa::1 => aaaa:0000:0000:0000
Got configuration message of type P
Setting prefix aaaa::
Server IPv6 addresses:
 aaaa::212:7401:1:101
 fe80::212:7401:1:101
```

Figure 7: Terminal command enabling RPL border router

The process of enabling the router continued by clicking the start button on the serial socket for the server motes. This action initiated the motes to listen to the desired port, as depicted in Figure 8.



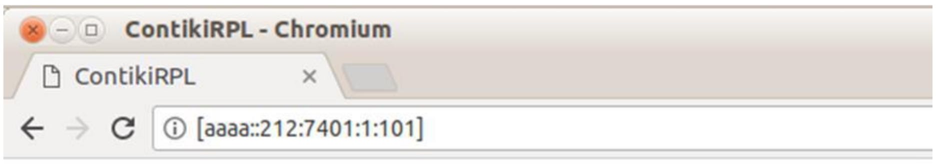
Listen port: 60001

socket -> mote: 0 bytes
mote -> socket: 0 bytes

Status: Address already in use

Figure 8: Serial socket (Server)

The final steps involve opening the browser on the same machine and entering the previously generated public address, as illustrated in Figure 9. In the event of a connection failure, reloading the network topology and restarting the simulation is recommended.

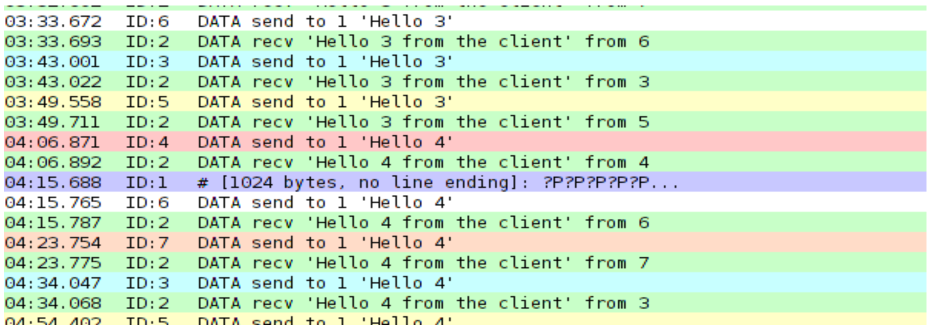


[aaaa::212:7401:1:101]

Figure 9: Web server header

4. Results and Discussion

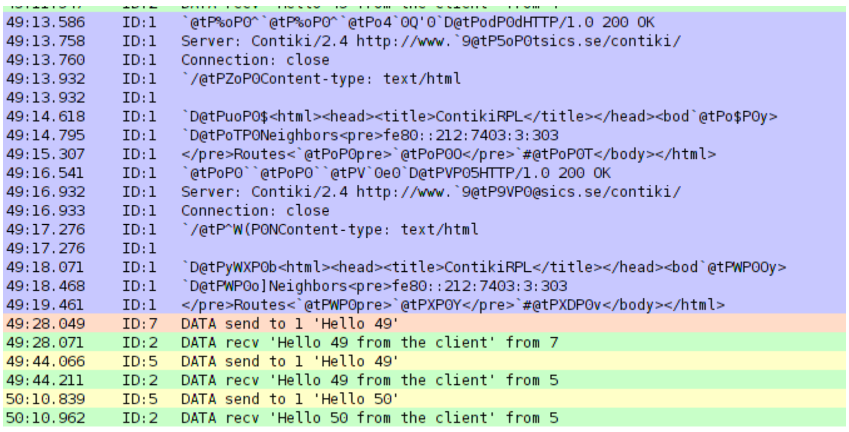
Figure 10 presents the simulation results in the form of output messages. The UDP client successfully transmits the message to the UDP server, and the UDP server successfully receives the message from the UDP client. However, it is worth noting that the system has not yet established a connection with the web server.



```
03:33.672   ID:6   DATA send to 1 'Hello 3'
03:33.693   ID:2   DATA recv 'Hello 3 from the client' from 6
03:43.001   ID:3   DATA send to 1 'Hello 3'
03:43.022   ID:2   DATA recv 'Hello 3 from the client' from 3
03:49.558   ID:5   DATA send to 1 'Hello 3'
03:49.711   ID:2   DATA recv 'Hello 3 from the client' from 5
04:06.871   ID:4   DATA send to 1 'Hello 4'
04:06.892   ID:2   DATA recv 'Hello 4 from the client' from 4
04:15.688   ID:1   # [1024 bytes, no line ending]: ?P?P?P?P?P...
04:15.765   ID:6   DATA send to 1 'Hello 4'
04:15.787   ID:2   DATA recv 'Hello 4 from the client' from 6
04:23.754   ID:7   DATA send to 1 'Hello 4'
04:23.775   ID:2   DATA recv 'Hello 4 from the client' from 7
04:34.047   ID:3   DATA send to 1 'Hello 4'
04:34.068   ID:2   DATA recv 'Hello 4 from the client' from 3
04:54.402   ID:5   DATA send to 1 'Hello 4'
```

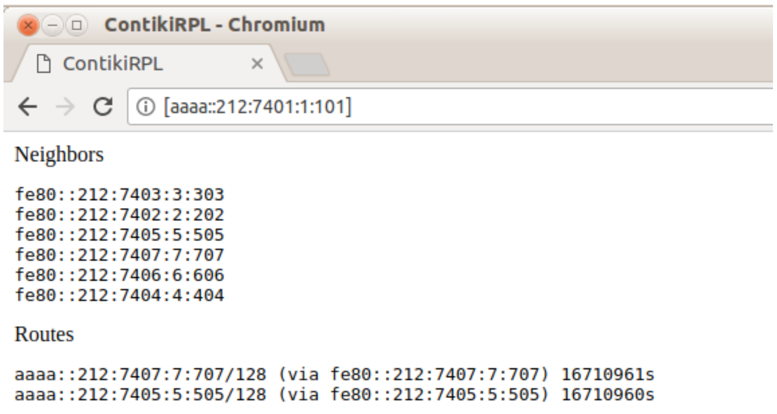
Figure 10: Output message between UDP client and UDP server

In the meantime, Figure 11, with purple color, illustrates the output message displayed once the UDP server and UDP client have successfully established a connection with the router. Figure 12 shows that the server connected to the UDP server, UDP client, and border and successfully received messages to be viewed on the web page.



```
49:13.586    ID:1   `@tP%oP0^`@tP%oP0^`@tPo4`0Q'0`D@tPodP0dHTTP/1.0 200 OK
49:13.758    ID:1   Server: Contiki/2.4 http://www.`9@tP5oP0tsics.se/contiki/
49:13.760    ID:1   Connection: close
49:13.932    ID:1   `/@tPZoP0Content-type: text/html
49:13.932    ID:1
49:14.618    ID:1   `D@tPuoP0$<html><head><title>ContikiRPL</title></head><bod`@tPo$P0y>
49:14.795    ID:1   `D@tPoTP0Neighbors<pre>fe80::212:7403:3:303
49:15.307    ID:1   </pre>Routes<`@tPoP0pre>`@tPoP0O</pre>`#@tPoP0T</body></html>
49:16.541    ID:1   `@tPoP0``@tPoP0``@tPV`0e0`D@tPVP05HTTP/1.0 200 OK
49:16.932    ID:1   Server: Contiki/2.4 http://www.`9@tP9VP0@sics.se/contiki/
49:16.933    ID:1   Connection: close
49:17.276    ID:1   `/@tP^W(P0NContent-type: text/html
49:17.276    ID:1
49:18.071    ID:1   `D@tPyWXP0b<html><head><title>ContikiRPL</title></head><bod`@tPWP0Oy>
49:18.468    ID:1   `D@tPWP0o]Neighbors<pre>fe80::212:7403:3:303
49:19.461    ID:1   </pre>Routes<`@tPWP0pre>`@tPXP0Y</pre>`#@tPXDP0v</body></html>
49:28.049    ID:7   DATA send to 1 'Hello 49'
49:28.071    ID:2   DATA recv 'Hello 49 from the client' from 7
49:44.066    ID:5   DATA send to 1 'Hello 49'
49:44.211    ID:2   DATA recv 'Hello 49 from the client' from 5
50:10.839    ID:5   DATA send to 1 'Hello 50'
50:10.962    ID:2   DATA recv 'Hello 50 from the client' from 5
```

Figure 11: Output message web server successfully connected



Neighbors

```
fe80::212:7403:3:303
fe80::212:7402:2:202
fe80::212:7405:5:505
fe80::212:7407:7:707
fe80::212:7406:6:606
fe80::212:7404:4:404
```

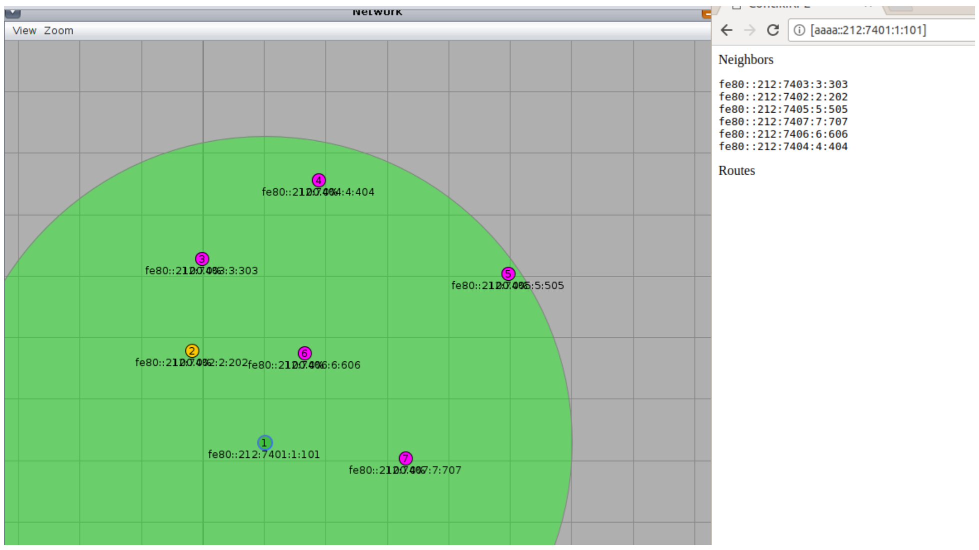
Routes

```
aaaa::212:7407:7:707/128 (via fe80::212:7407:7:707) 16710961s
aaaa::212:7405:5:505/128 (via fe80::212:7405:5:505) 16710960s
```

Figure 12: Web server page

4.1 Evaluate Radio Transmission and Radio Reception

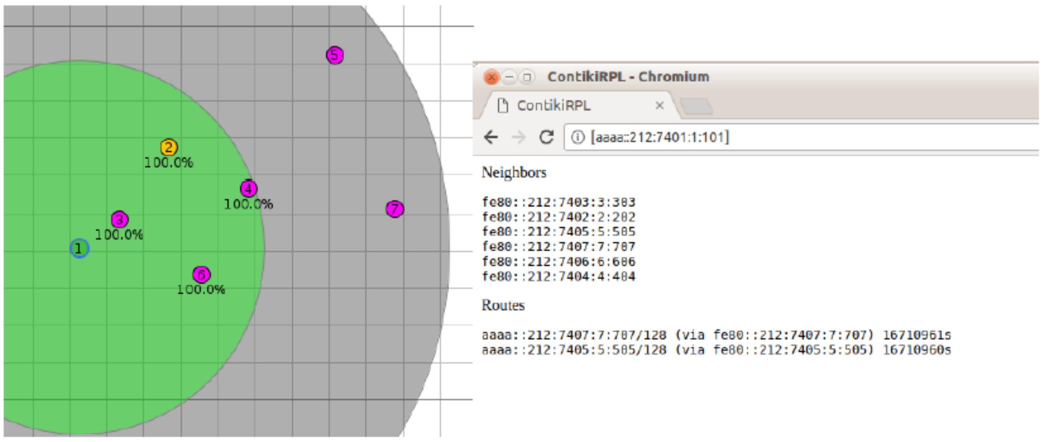
Figure 13 presents an ideal condition (case 1) in which all nodes are within the UDGM (Underneath the Direct Global Mote) of the border. Consequently, the UDP client and UDP server can directly transmit messages to the border motes. As a result, all nodes become neighbor nodes to the border mote, and no nodes function as router nodes.



```
[aaaa::212:7401:1:101]

Neighbors

fe80::212:7403:3:303
fe80::212:7402:2:202
fe80::212:7405:5:505
fe80::212:7407:7:707
fe80::212:7406:6:606
fe80::212:7404:4:404

Routes
```

Figure 13: Simulation layout (case 1)

Figure 14 illustrates a non-ideal case (case 2) where Node 5 and Node 7 are located outside the UDGM radius of the border node. Consequently, Node 5 and Node 7 must pass the message to Node 4 to transmit the message to the router. As a result, Node 5 and Node 7 function as router nodes and their presence is reflected in the web server.

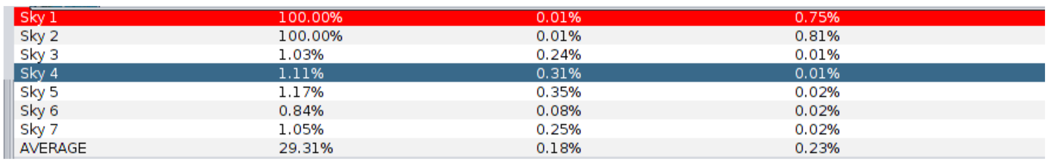


```
[aaaa::212:7401:1:101]

Neighbors

fe80::212:7403:3:303
fe80::212:7402:2:202
fe80::212:7405:5:505
fe80::212:7407:7:707
fe80::212:7406:6:606
fe80::212:7404:4:404

Routes

aaaa::212:7407:7:707/128 (via fe80::212:7407:7:707) 16710961s
aaaa::212:7405:5:505/128 (via fe80::212:7405:5:505) 16710960s
```

Figure 14: Simulation layout (case 2)

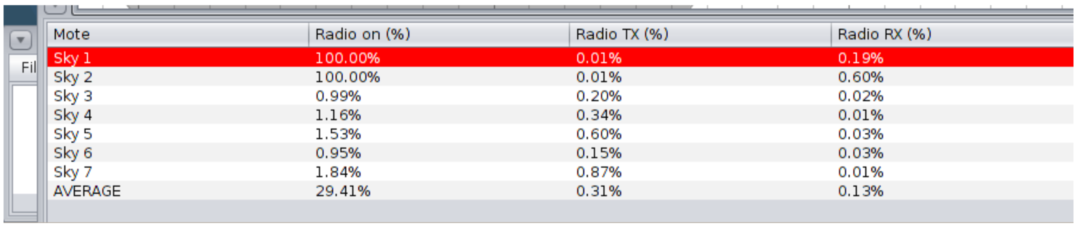
Figures 15(a) and 15(b) compare the transmission and reception ratios between the two aforementioned cases. Based on this simulation, it can be concluded that the ideal case, where all nodes are within the UDGM radius of the router, exhibits a lower radio transmit average of 0.18% and a higher radio receive average of 0.23%. This can be attributed to the favorable positioning of the nodes, allowing for direct connectivity to the border node.

Conversely, in case 2, where only a few nodes are within the UDGM radius, the radio transmission percentage increases to 0.31% compared to the ideal case. Moreover, the radio receive ratio for case 2 decreases to 0.13% compared to case 1.



Sky 1	100.00%	0.01%	0.75%
Sky 2	100.00%	0.01%	0.81%
Sky 3	1.03%	0.24%	0.01%
Sky 4	1.11%	0.31%	0.01%
Sky 5	1.17%	0.35%	0.02%
Sky 6	0.84%	0.08%	0.02%
Sky 7	1.05%	0.25%	0.02%
AVERAGE	29.31%	0.18%	0.23%

Figure 15(a): Radio duty cycle case 1



Mote	Radio on (%)	Radio TX (%)	Radio RX (%)
Sky 1	100.00%	0.01%	0.19%
Sky 2	100.00%	0.01%	0.60%
Sky 3	0.99%	0.20%	0.02%
Sky 4	1.16%	0.34%	0.01%
Sky 5	1.53%	0.60%	0.03%
Sky 6	0.95%	0.15%	0.03%
Sky 7	1.84%	0.87%	0.01%
AVERAGE	29.41%	0.31%	0.13%

Figure 15(b): Radio duty cycle (case 2)

In conclusion, these results highlight the significance of proximity to the border router in achieving higher transmission and reception rates in the network based on two case studies.

5. Conclusion

In conclusion, this paper presented the design and simulation of a network using Cooja as the simulator and the Contiki OS System for User Datagram Protocol (UDP). The communication between the client and server was observed through a web server. Due to the nature of UDP, the specific transmission and receiving paths between the server and client could not be determined, as the data can be transmitted through multiple available paths in the network channel. The use of UDP offers fast data transmission, as it does not require the establishment of a pre-established virtual path for transfer. This makes UDP well-suited for real-time services and live communication scenarios. Hence, the simulation results and analysis provided insights into the performance and behavior of the UDP client-server model in the simulated environment. Further research and experimentation can be conducted to explore additional aspects of the network and investigate the optimization of UDP-based communication for various IoT applications.

References

[1] N.J. Ayidu and O. V. Elaigwu, “Probability Prediction of User Datagram Protocol (UDP) Upstream Throughput in a Network,” Journal of Energy Technology and Environment, vol. 4, no. 2, June 2022, https://doi.org/10.37933/nipes.e/4.2.2022.17

[2] K. Gatimu, A. Dhamodaran, T. Johnson and B. Lee, “Experimental study of QoE improvements towards adaptive HD video streaming using flexible dual TCP-UDP streaming protocol,” Multimedia Systems, vol. 26, pp. 479–493, August 2020, https://doi.org/10.1007/s00530-020-00653-w

[3] Y. Yu and S. Lee, "Remote Driving Control With Real-Time Video Streaming Over Wireless Networks: Design and Evaluation," in IEEE Access, vol. 10, pp. 64920-64932, 2022, doi: 10.1109/ACCESS.2022.3183758

[4] S. Deshmukh-Bhosale, and S. S. Sonavane, “A Real-Time Intrusion Detection System for Wormhole Attack in the RPL based Internet of Things,” Procedia Manufacturing, vol. 32, pp. 840–847, 2019, doi:10.1016/j.promfg.2019.02.292

[5] A. Faisal, and M. Zulkernine, “A secure architecture for TCP/UDP-based cloud communications,” International Journal of Information Security,” vol. 20, pp. 161-179, 2021, https://doi.org/10.1007/s10207-020-00511-w

[6] E. Gamess, and B. Smith, “Performance Evaluation of TCP and UDP over IPv4 and IPv6 for the ESP8266 Module,” Proceeding of the 2nd International Electronics Communication Conference, Singapore, July 8-10, 2020, pp. 161-169. Association for Computing Machinery, NY, US, 2024, https://doi.org/10.1145/3409934.3409956

[7] T. Czachórski, E. Gelenbe, G. S. Kuaban and D. Marek, "Transient Behaviour of a Network Router," 2020 43rd International Conference on Telecommunications and Signal Processing (TSP), Milan, Italy, 2020, pp. 246-251, doi: 10.1109/TSP49548.2020.9163477.

[8] M. Yadollahzadeh Tabari, and Z. Mataji, “Detecting sinkhole attack in rpl-based Internet of things routing protocol,” Journal of AI and Data Mining, vol. 9, no. 1, pp. 73-85, Jan 2021, https://doi.org/10.22044/jadm.2020.9253.2060

[9] M. C. Belavagi, and B. Muniyal, “Multiple intrusion detection in RPL based networks,” International Journal of Electrical and Computer Engineering, vol. 10, no. 1, pp. 467-476, Feb 2022, http://doi.org/10.11591/ijece.v10i1.pp467-476