Communication using UDP

Akshaj Jauhri
Electronics and Communication Engineering
Manipal Institute of Technology
Manipal, Karnataka, India
akshaj.jauhri@learner.manipal.edu

Tejash Agarwal
Electronics and Communication Engineering
Manipal Institute of Technology
Manipal, Karnataka, India
tejash.agarwal@learner.manipal.edu

***Abstract*—This project aims to develop a user-friendly program facilitating communication between two computers on a local Ethernet network. Utilizing a custom protocol built over UDP, the program enables seamless exchange of text messages and files. It comprises two main components: broadcasting and receiving. The broadcasting node sends files, breaking them into smaller fragments if necessary, while the receiving node acknowledges the receipt of each fragment. Error checking and retries are integrated to ensure reliable transmission. Key features include user-defined maximum fragment size to optimize transmission, periodic maintenance packets for connection stability, and the ability for nodes to switch between transmitter and receiver functions without restarting. The solution prioritizes simplicity and functionality, offering an accessible means for efficient network communication.**

Keywords—UDP, File exchange, Error checking, Connection stability

I. Introduction

The objective of this project is to create a program that facilitates communication between two participants on a local Ethernet network using a custom protocol built on top of UDP. The program should enable the exchange of text messages and files between computers.

The program proposal consists of two parts - transmitting and receiving, i.e. client-server model. On starting the program, a client and server process is created which communicates with each other by reading and writing to sockets.

Users can set the maximum fragment size to prevent further fragmentation at the network layer. The receiving node acknowledges the receipt of each fragment and outputs a message indicating successful reception. Once the entire file is received, the destination node displays a message confirming receipt and the absolute path where the file is saved.

The program includes error checking and retries for erroneous fragments, with both positive and negative confirmations.

The program is designed so that both nodes can switch between transmitter and receiver functions without restarting. The solution emphasizes simplicity and exercise functionality.

II. Communication Protocol

The UDP protocol is a protocol of the TCP/IP network model working on the Transport Layer. It allows fast data transfer to ensure real-time communication services. These mostly include services that emphasize fast data transfer like Live transmissions of Audio-Visual media, online gaming, videoconferencing, and so on. However, this does not ensure reliable data transfer. UDP is a connectionless protocol that, unlike TCP, does not resend lost or damaged data and at the same time it does not ensure data sorting. All these features must therefore be replaced by the proposed communication protocol working on the application layer of the TCP/IP network model.

The TFTP protocol (Trivial File Transfer Protocol) is a protocol working on the application layer of the TCP/IP network model used in file transfer while working with the UDP protocol on the transport layer. TFTP is based on File Transfer Protocol (FTP) technology, but it's a different protocol. It operates on a client-server model. In general, it is a simple FTP that allows a client to get a file from or put a file onto a remote host.

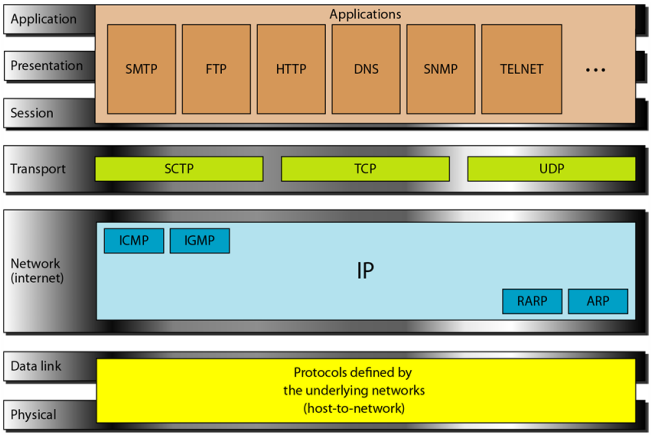

Fig. 1. TCP/IP Protocol

III. Methodology

The program architecture works with network sockets. These sockets work as the intermediary between the application and the transport layer within the host system. Essentially, these sockets work as the interface between the application and the network and handle the communication protocols, source and destination IP addressed, and the destination and source ports. The socket is mainly used to establish control over the application layer, with lesser control over the transport layer, as it is typically managed by the Operating System.

The proposed design is represented in a flow chart format below:

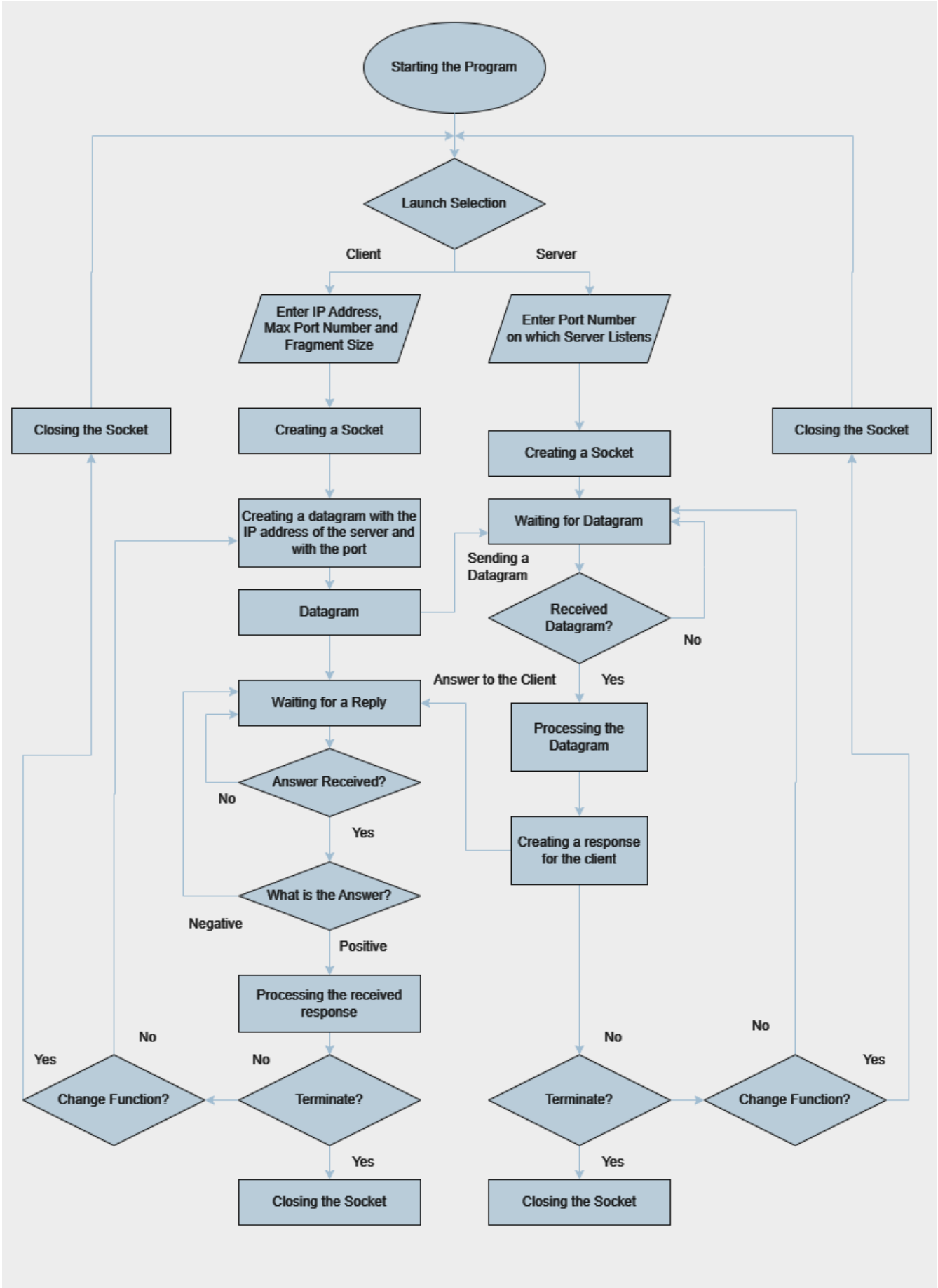

Fig. 2. Flow Chart of Methodology

IV. Header Structure Design

The proposed header structure is as follows:

Message type	Serial number
Serial number (cont.)	Size
Size (cont.)	Number of fragments
Number of fragments (cont.)	Checksum
Checksum (cont.)	Data

Fig. 3. Header Format

A. *Message Type (2 Bytes)*

It determines the type of message that is being transmitted.

- 0 determines the re-request of data, i.e. a negative response.
- 1 determines the correct delivery of data, i.e. a positive response.
- 2 determines whether to send or receive a text message.
- 3 determines whether to send or receive a file
- 4 specifies a keepalive packet

B. *Serial Number (4 Bytes)*

It helps to sort the fragments into the correct order.

C. *Size (4 Bytes)*

This is the size of the received or sent fragment.

D. *Number of Fragments (4 Bytes)*

It is the total number of divisions made of the packet.

E. *Checksum (4 Bytes)*

This is the checksum value for Error detection.

F. *Data*

This contains the data being transmitted.

V. Error Detection and ARQ

A. *Message Type (2 Bytes)*

Cyclic redundancy is used as an Error Detection method. This error detection technique is widely used today. It detects errors in digital data but does not correct when errors are detected.

A certain number of check bits called a checksum, are appended to the message, and transmitted as a Code word. The Rx determines whether the check bits agree with the data, to ascertain with a certain degree of probability whether or not an error occurred in transmission. If an error occurs, the receiver sends a “negative acknowledgment” (NAK) back to the sender, requesting that the message be retransmitted.

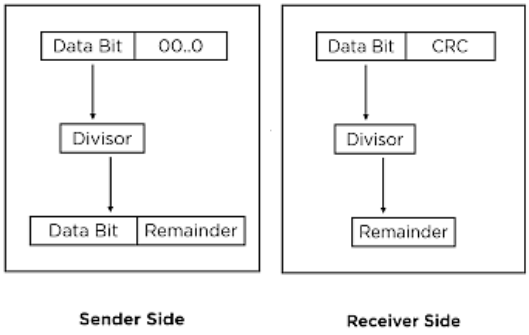

Fig. 4. Checksum Implementation

For the data block D, the sender chooses r-bits R, which are connected to the data block D, and therefore the resulting bit pattern d + r is divisible by a number without a remainder G using modulo-2 division. Thus, the receiver divides d + r received bits by G, and if the remainder is zero, no error has occurred. If not, it is clear that an error has occurred.

B. *Automatic Repeat reQuest*

ARQ is used to control the error rates that the transmitter requires from the receiver confirmation of data receipt. The ARQ method Stop & Wait is used in the entry, in which the transmitter expects an acknowledgment for each packet it sends, while other packets are not sent until an acknowledgment (ACK) arrives. If it does not arrive within a certain time, the transmitter resends the packet.

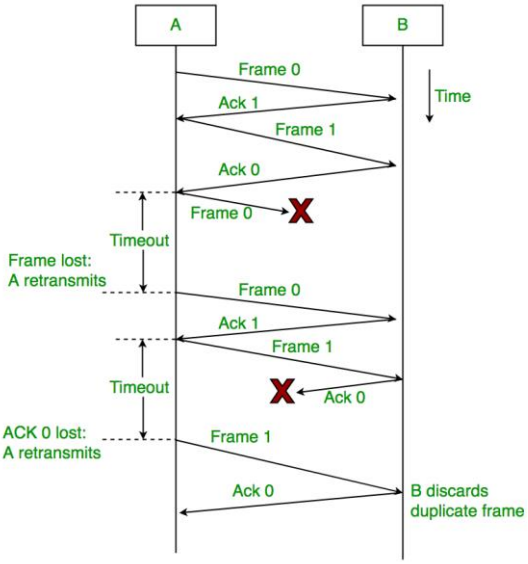

Fig. 5. Stop and Wait ARQ Implementation

You can cite your references in text by including the corresponding number, in square brackets [1]. If you need to cite a specific part of the source, you can include a page number [2, p. 13] or range [3, pp. 41–56].

VI. Code Outline

When implementing UDP sockets, the Python socket module is used, which enables creating and working with network sockets within the program. Inserting the socket module into the program is performed for both parts of the client-server program.

For communication using the UDP protocol, the implementation spans three source files written using Python. The communication protocol and the interaction between the client and server are defined in three specific files:

A. Client.py Source File:

This file describes the behaviour of the client. This includes:

- User Environment Setup
- Client Configuration Settings
- Sending Text Messages
- Sending Files

B. Server.py Source File:

This file describes the behaviour of the server. This includes:

- User Environment Setup
- Server Configuration Settings
- Receiving Data
- Outputting Text Messages to the Console

C. Protocol.py Source File:

This file describes the implementation of the protocol. This includes:

- Specifying the Protocol Structure
- Implementing CRC
- Creating Headers During Communication

VII. IMPLEMENTATION

The program can be run via the command line, while the user interface is the console application itself.

A. Client:

The client's interactive user environment takes place through communication with the user at the console level. The user sets the destination IP address of the server, also the destination port server and maximum fragment size.

The available menu allows the user to:

- end the program by entering the number 0.
- send a text message by entering the number 1.
- send the file by entering the number 2.
- change functionality from client to server by entering the number 9.

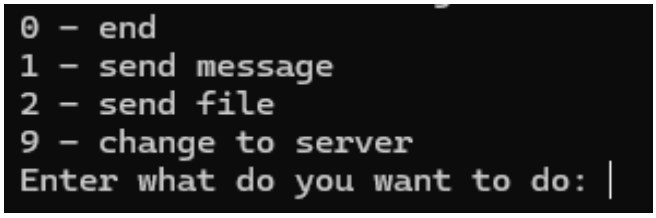

Fig. 6. Client UI

Before displaying the client's user menu, it is necessary to initialize, i.e. set the client, and therefore the user enters the following at the program's prompt

- server, i.e. destination, IP address in the form of IPv4
- client source port in the range from 1024 to 65535.
- serve port in the range from 1024 to 65535.
- Maximum size of fragment

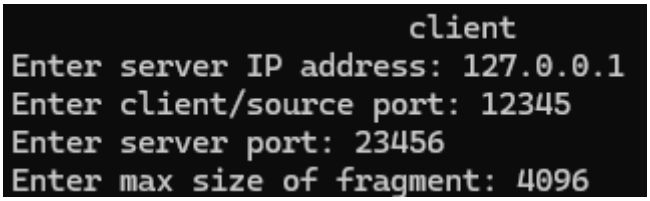

Fig. 7. Client Setup

B. Server:

An interactive user interface is also available for the communication server with the user at the console level. The user sets the source port of the server on which the server is "listening" (waiting for data) and the location path for saving the received files.

The available menu allows the user to:

- end the program by entering the number 0.
- accept data by entering the number 1.
- change functionality from server to client by entering number 9.

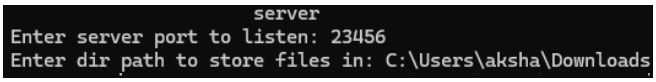

Fig. 8. Server UI

Before displaying the user menu of the server, it is necessary to initialize, the server, and therefore the user enters the following:

- server source port in the range from 1024 to 65535.
- the existing path to the folder where the server will store the received files.

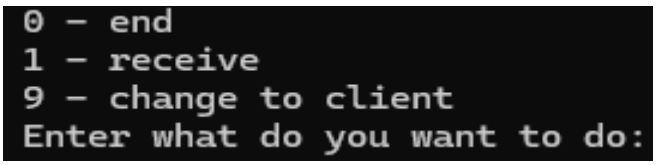

Fig. 9. Server Setup

VIII. RESULTS

The observed results are as follows:

A. Message Communication:

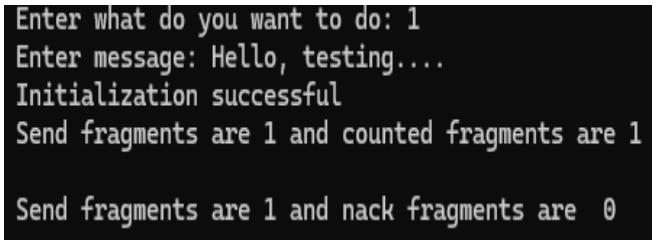

Fig. 10. Client sending message.

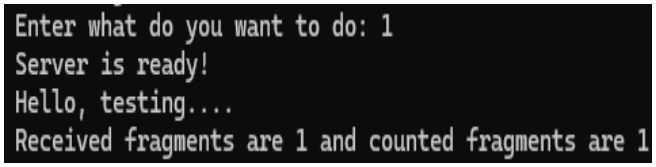

Fig. 11. Server receiving message.

B. Message Communication:

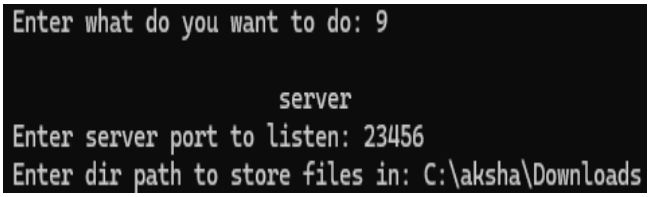

Fig. 12. Changing client to server

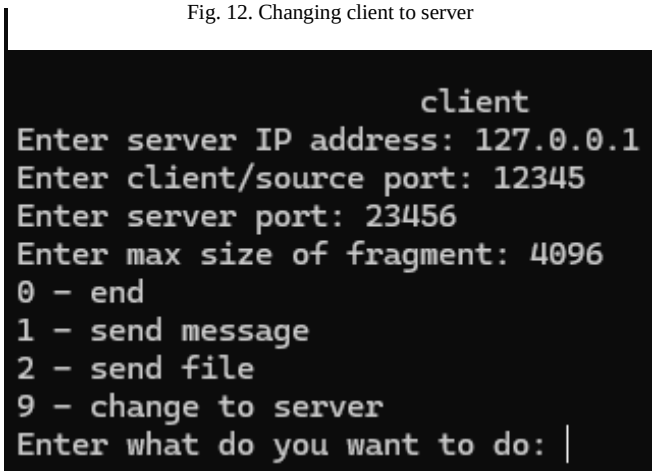

Fig. 13. Changing server to Client

IX. Evaluation and Conclusion

The process of implementing a communication protocol at the level of communication between clients and servers using the UDP protocol on the transport layer of the TCP/IP model was implemented according to the developed draft program and communication protocol. however, in contrast to the proposal, specific situations arose during the implementation in which the implementation differed from the design itself, so the following lines will describe what differences occurred between the protocol design and the actual implementation.

A. Header Structure:

The client's interactive user environment takes place through communication with the user at the console level. The user sets the destination IP address of the server, also the destination port server and maximum fragment size.
Message type (1B)-change compared to the original proposal in the size of bytes from two bytes minimized to one byte, determines which message it is and here too there were changes compared to the original proposal. Message types now include the following signaling messages:

- 0 specifies the initialization of the communication for sending files in which the file name is sent in the data part for maximum optimization with data.
- 1 specifies the data to be written.
- 2 determines a positive answer, le correctly received data.
- 3 determines a negative response, ie sending data again.
- 8 determines the initialization of the text message.

Fragment size (4B) unchanged, determines the maximum size of the received/sent fragment specified by the user.

Checksum(1B)-change compared to the original proposal in size of bytes from four bytes minimized to one byte.

To minimize and optimize work with data, the following were eliminated fields from the draft protocol header:

Serial number (4B) - discarded due to the Stop & Wait ARQ method used, in which the number of fragments is not necessary for the correct classification of files, because if a fragment is delivered incorrectly, the server will ask to send it again of the same fragment.

Number of fragments (4B) - discarded because using the Python select module and ARQ of the Stop & Wait method, there is no need to transmit information about the number fragments because the mentioned module is used for waiting for completion input-output operations for the possibility of further work with data, however in the initial initialization message, this information is transmitted as data, there is no need to store this information in the header.

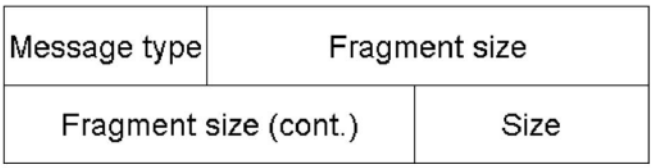

Fig. 14. Changed header format

B. Checksum and ARQ:

The checksum method and ARQ operation unchanged, ie the CRC method is used checksum and Stop & Wait ARQ method.

References

[1] Qifeng Sun and Hongqiang Li, "Research and application of a UDP-based reliable data transfer protocol in wireless data transmission," 2011 International Conference on Computer Science and Service System (CSSS), Nanjing, 2011, pp. 1514 -1516, doi: 10.1109/CSSS.2011.5972154

[2] G. Gehlen, F. Aijaz and B. Walke, "Mobile Web Service Communication Over UDP," IEEE Vehicular Technology Conference, Montreal, QC, Canada, 2006, pp. 1-5, doi: 10.1109/VTCF.2006.590. keywords: {Web services;Mobile communication;Simple object access protocol;Middleware;Automatic repeat request;Acceleration;Error correction;Transport protocols;Service oriented architecture;Web server}.

[3] A Reliable UDP for Ubiquitous Communication Environments, Doan Thanh Tran, Eunmi Choi∗, School of Business IT, Kookmin University, Jeongnueng-Dong, Seongbuk-Gu, Seoul, 136-702 Korea.

[4] N. J. Ploplys and A. G. Alleyne, "UDP network communications for distributed wireless control," Proceedings of the 2003 American Control Conference, 2003., Denver, CO, USA, 2003, pp. 3335-3340 vol.4, doi: 10.1109/ACC.2003.1244046. keywords: {Wireless communication;Distributed control;Communication system control;Sampling methods;Bluetooth;Performance loss;Control systems;Ethernet networks;Protocols;Peer to peer computing},

[5] W. Bai, P. Zarros, M. Lee and T. Saadawi, "Design, implementation and analysis of a multimedia conference system using TCP/UDP," Proceedings of ICC/SUPERCOMM'94 - 1994 International Conference on Communications, New Orleans, LA, USA, 1994, pp. 1749-1753 vol.3, doi: 10.1109/ICC.1994.368734. keywords: {TCPIP;Transport protocols;Delay;Videoconference;Open systems;Multimedia systems;FDDI;Jitter;Performance evaluation;Application software},