University of Maryland, College Park

ENTS640 Networks and Protocols I

Final Project Report

Handa Li
Barok Imana

Outline

- Problem Description and Specifications
- Design and Solutions
- Output and Analysis
- Measurement and Analysis
- Conclusion

Problem Description and Specifications

The general idea of this project is to achieve reliable data transfer over UDP. This distributed networking application not only should successfully achieve the data transmission between the client and the server, but also should calculate the round trip time(RTT) and display the transferring process during the communication. The application has to comply with the stop-and-wait protocol in order to meet the requirement. The requirement, together with the protocol regulate the application's communication protocol. The client transmits a packet data unit(PDU) of information and then wait for a response. The server receive each PDU and sends an acknowledgement back to the client only if the corresponding PDU is received correctly. In this project, the transmission process has two phases. First, the client will send a initial packet to the server with the information of the transmission. And the server should receive and respond with initial packet acknowledgement if the initial packet is received correctly by checking if it has the correct format. Second, once the initial phase is passed without any errors, the client start to transfer data packets to the server using stop-and-wait protocol in a total of set number of packet the project has specified.

Also, the project has 4 different types of packets in the transmission. Including INIT packet, IACK packet, DATA packet and the DACK packet. Each Packet has its packet type field and the integrity check field. The packet type value is different for all 4 packets, The INIT packet, meaning the initial packet, consists of the fields of its packet type, randomly generated initial sequence number, the number of the total data packets to be transmitted, the number of the byte in the payloads and the integrity check field. The IACK packet and the DACK packet have the similar fields: packet type, ACK(acknowledgement) number and the integrity check field. And the DATA packet has fields of packet type, sequence number, the set number of randomly generated payload and the integrity check field.

Before starting the transmission process, the IP address of the server should be set, and the port number of the server should be set to the same value on the client side. The client should then send the INIT packet to indicate the beginning of the transmission. After receiving the INIT packet, the server should check whether the received packet is INIT packet by checking its

packet type value. And then go through the integrity check process to validate if the value is 0 or not. The integrity check will break the byte array of the receiving packet into a 2 byte word and exclusive-or it with a variable that initially sets to 0. Then store the result back to the variable to be XORed with next 16 bits word. After the INIT packet is correctly received, the server will send back a IACK packet with sequence number increased by 1 because the INIT packet consume 1 sequence number. Also with the corresponding integrity check value and packet type. The the IACK packet received by the client will go through the integrity check, ack number check and packet type check again to be evaluated for its correctness. Once the initial hand-shake phase has passed, the client will start to send the DATA packet to the server and receiving the DACK packet from the server as in the initial phase.

Design and Solution

Design Approach

A state machine approach is used in the implementation of this protocol. As described in previous sections, the operation of this protocol is divided into two phases: the initial handshake and data transmission. These two phases were used to formulate two states both in the client and server application.

Client Application

Initial State

In this state, the client attempts to establish communication by sending the INIT packet and waiting for IACK packet response. The client enters this state once it builds the INIT packet and passes it to the UDP socket. It starts a timer and waits for an IACK packet from the server application. In this state, there are two possible events: a receipt of a packet from the UDP socket or a socket timeout. In the first case, the client should process the received packet and verify that it is an IACK packet and has the correct sequence number and checksum value. If the received packet passes all the checks, the client shall enter the data transmission state and transmit the first data packet. On the other hand, if the packet fails any of the checks, timer value will be reset and the INIT packet will be retransmitted and client will continue in its current state. In the case of a timeout event, like the previous case, the INIT will be retransmitted but

the timeout interval will be doubled. If such timeout event occurs for a fourth time, the client application will exit, outputting a communication failure message.

Data Transmission State

This state is where the data transmission takes place on the client side of the application. The application enters this state once a correct IACK packet is received and the first data packet is transmitted. In this state, the client transmits all its data packets. Again in this state, like the previous state, there are two possible events: a packet receipt and a socket timeout. In the case of a receipt of the a packet that meets all the requirements, the client will transmit the following DATA packet. If not, the timer will be reset and the last DATA packet will be retransmitted. In the case of a timeout, like the same case in the initial state, the last DATA packet is retransmitted and the timeout interval is doubled. Again, if such timeout occurs for the fourth time before the receipt of a DACK packet, the application exits with an error message noting communication failure. On the other hand, if all the DACK packets for the corresponding DATA packets are received successfully, the application will exit it signalling the communication was successful.

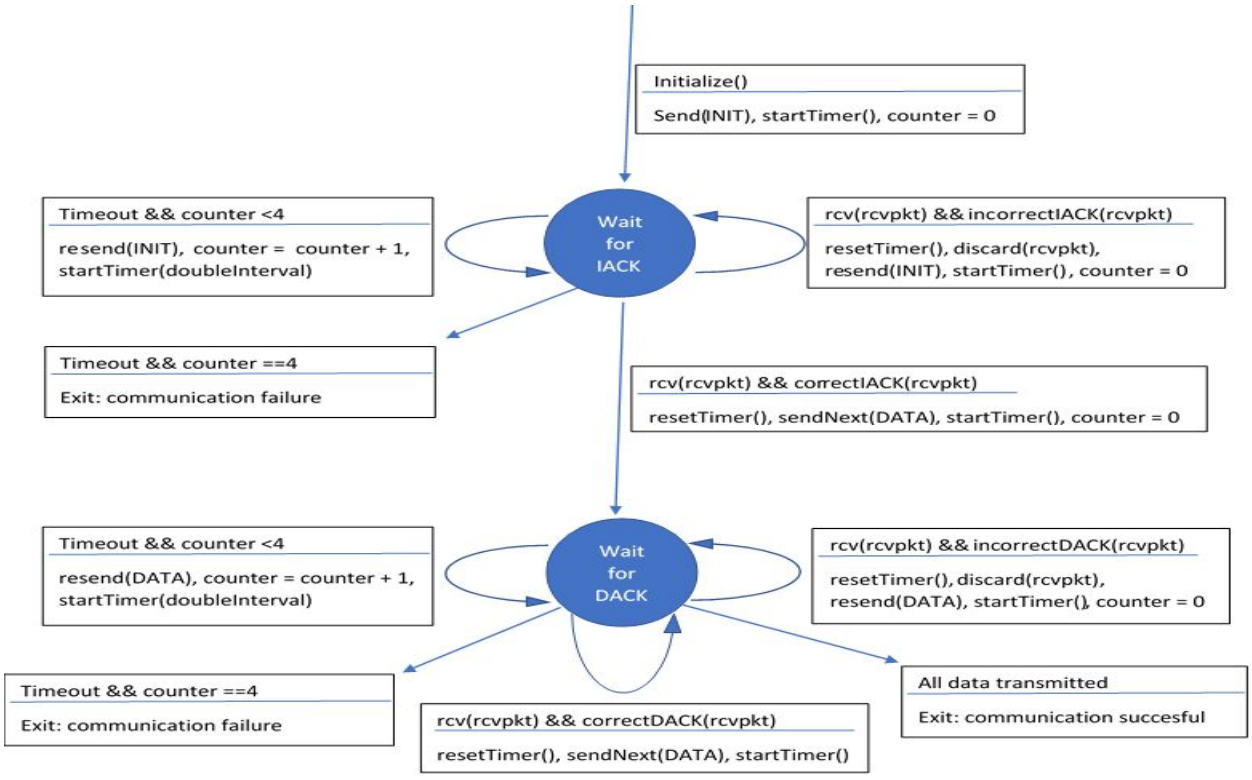

Figure: Client Protocol State Diagram

Server Application

Initial State

This is the state where the receiver await a valid INIT packet from the client application. As there are no cases for timeouts on the server's application, the only possible event is a receipt of packet. Once a packet is received, if it has the correct packet type and passes integrity check, the application advances to the next state by sending an IACK packet. The initial sequence number, the number of payload bytes and the total number of packets are all extracted from the packet and set for the next state. On the other hand, if the packet received does not correspond to a valid INIT packet, the packet is discarded and the server continues to listen for another INIT packet.

Data Transmission State

This is the state where primarily data transmission takes place on the server side. But it also handles the case where the retransmission of IACK packets is needed. Again, the only possible event in this state is the receipt of a packet. Once the server has transmitted the IACK packet in the transitioning from the previou state, it would be expecting the first DATA packet but it could receive a duplicate INIT packet. In the case of a duplicate INIT packet, the server retransmits the IACK packet and this continues until a DATA packet arrives. Once a DATA packet arrives, if it passes all the checks, the corresponding DACK packet is transmitted and the server awaits the next DATA packet. In the case a duplicate DATA packet is received, the server retransmits the last DACK packet. In any case if a packet received (whether an INIT or DATA) is corrupted, the packet is discarded and no transmission is performed by the server application. Once all the DATA packets have been received as specified in the INIT packet, the application exits acknowledging successful communication.

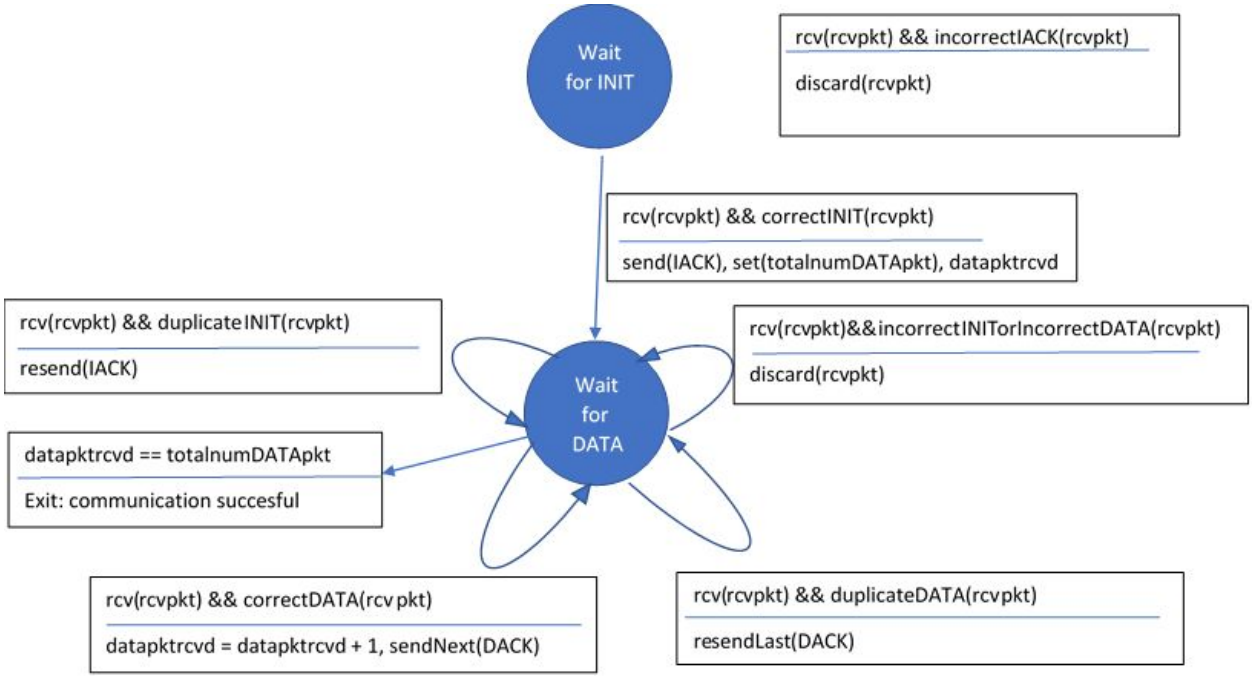

Figure: Server Protocol State Diagram

JAVA Implementation

Program Design

Both for the client application and the server application, a model-controller approach was used to structure the code into two sections. On the client application, the model consists of the classes for the two types of packet to be transmitted by the client: “ClientInitPacket.java” and “ClientDataPacket.java”. Both classes extend a class called “Packet.java”. The controller for the client application is implemented in class named “Client.java”.

On the server side, only one class was used to represent both packet types as IACK and DACK packet types are inherently the same except for their “packet type” field. This class is called “ServerPacket.java” and it also extends the class “Packet.java”. The controller for the server application is implemented in “Server.java” and this makes use of the “ServerPacket.java” class.

UML Diagram

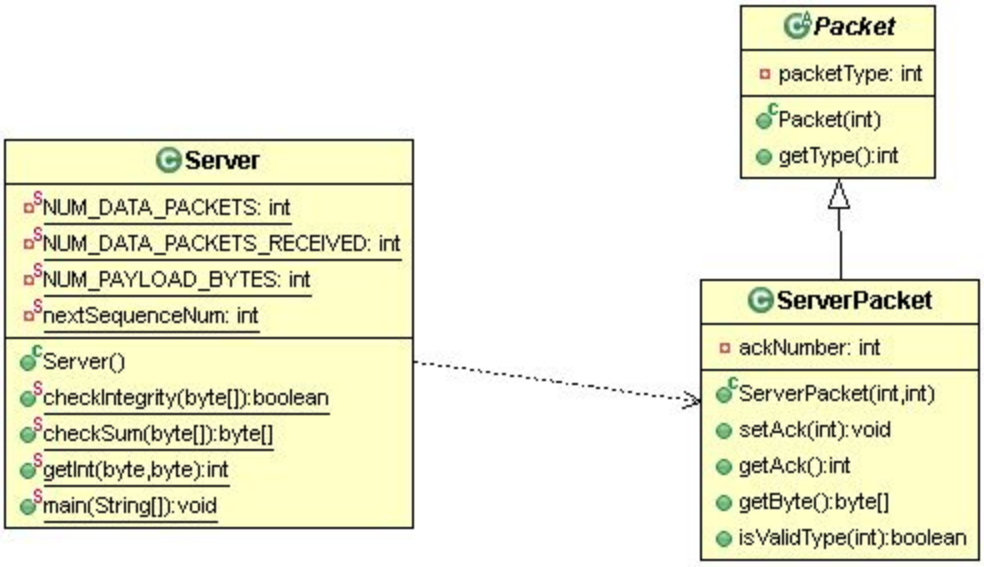

Figure: UML Diagram for Classes in Server Application

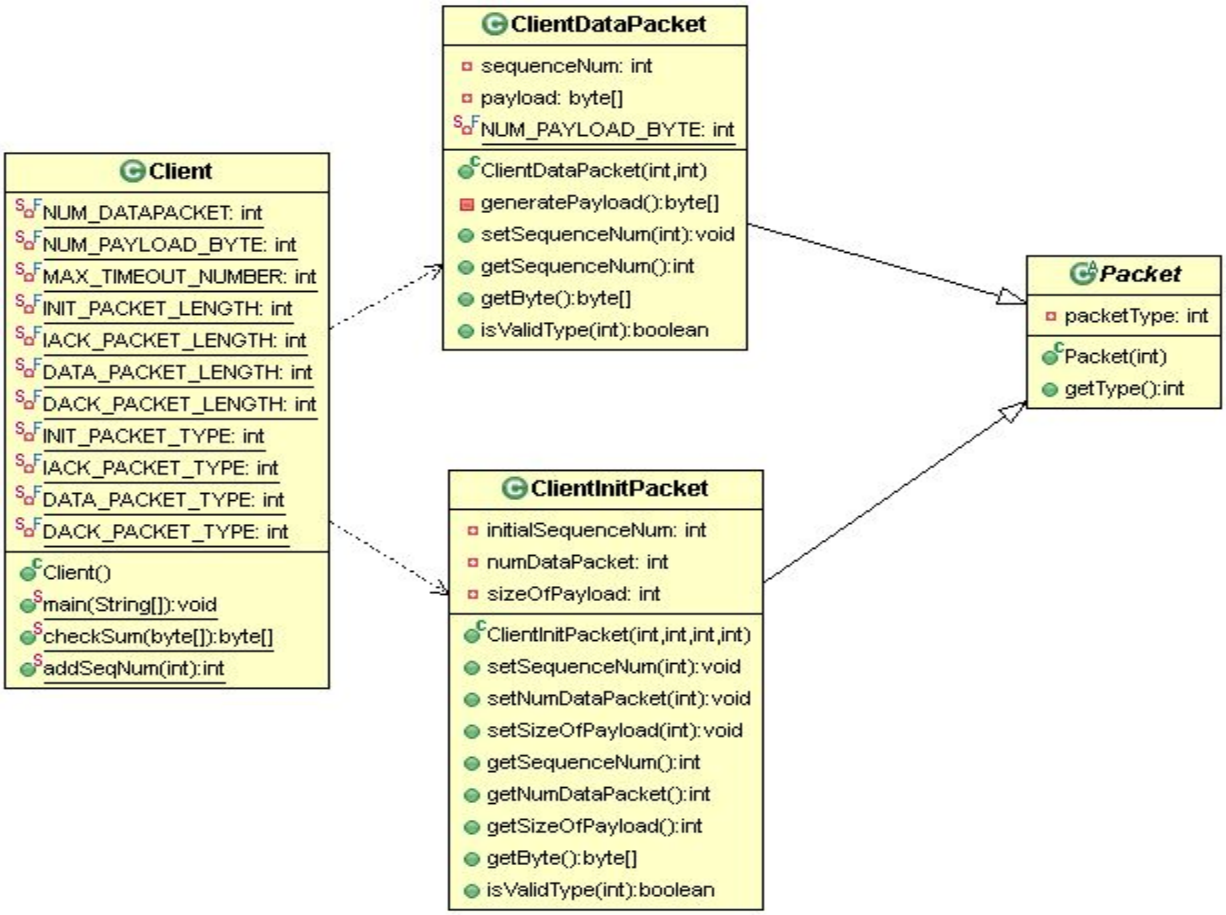

Figure: UML diagram for classes used in Client Application

Output and Results

Successful Transmission

Client	Server
initial sequence number is: 16466 InitPacket sent... waiting for iack packet Iack Packet received Iack Packet type is correct. the ack number of the IackPacket is correct integrity check passed.. initial phase passed, starting to transmit the data packet Number of data packet transmitted:1 waiting for dack packet dack packet received Dack Packet type is correct. the ack number of the dackPacket is correct integrity check passed.. Number of data packet transmitted:2 . . . Number of data packet transmitted:10 waiting for dack packet dack packet received Dack Packet type is correct. the ack number of the dackPacket is correct integrity check passed.. transmission success!	Waiting for INIT packet... Packet received Received a valid INIT packet Sendning an IACK Expecting DATA packets... Packet received Received in-order DATA packet Number of data packets received:1 Sending a DACK Expecting DATA packets... Packet received Received in-order DATA packet Number of data packets received:2 Sending a DACK . . . Received in-order DATA packet Number of data packets received:9 Sending a DACK Expecting DATA packets... Packet received Received in-order DATA packet Number of data packets received:10 Sending a DACK Communication was successful!!!

Summary: All packets are transmitted successfully.

First IACK Lost

Client	Server
initial sequence number is: 23831 InitPacket sent... **waiting for iack packetReceive timed out** **Resending the INIT packet** InitPacket sent... waiting for iack packet Iack Packet received Iack Packet type is correct. the ack number of the IackPacket is correct integrity check passed.. initial phase passed, starting to transmit the data packet Number of data packet transmitted:1 . . . Number of data packet transmitted:10 waiting for dack packet dack packet received Dack Packet type is correct. the ack number of the dackPacket is correct integrity check passed.. transmission success!	Waiting for INIT packet... Packet received Received a valid INIT packet Sending an IACK Expecting DATA packets... Packet received **Received duplicate INIT packet** **Resending IACK packet** Expecting DATA packets... Packet received Received in-order DATA packet Number of data packets received:1 Sending a DACK Expecting DATA packets... . . . Number of data packets received:10 Sending a DACK Communication was successful!!!

Summary: Client times-out and resends the INIT packet. The server resends the IACK once it receives the duplicate INIT packet and communication proceeds to successful completion.

Second DATA Packet Corrupted

Client	Server
initial sequence number is: 14163 InitPacket sent... waiting for iack packet Iack Packet received Iack Packet type is correct. the ack number of the IackPacket is correct integrity check passed.. initial phase passed, starting to transmit the data packet Number of data packet transmitted:1 waiting for dack packet dack packet received Dack Packet type is correct. the ack number of the dackPacket is correct integrity check passed.. Number of data packet transmitted:2 **waiting for dack packetReceive timed out Resending the DATA packetNumber of data packet transmitted:2** waiting for dack packet dack packet received Dack Packet type is correct. . . . Number of data packet transmitted:10 waiting for dack packet dack packet received Dack Packet type is correct. the ack number of the dackPacket is correct integrity check passed.. transmission success!	Waiting for INIT packet... Packet received Received a valid INIT packet Sending an IACK Expecting DATA packets... Packet received Received in-order DATA packet Number of data packets received:1 Sending a DACK Expecting DATA packets... **Packet received** **Packet failed integrity check!** Expecting DATA packets... . . . Expecting DATA packets... Packet received Received in-order DATA packet Number of data packets received:10 Sending a DACK Communication was successful!!!

Summary: The server does not send a DACK for the second DATA packet because it is corrupted. The client times-out and resends the DATA packet. Communication proceeds to successful completion.

Sample DATA Packet

[51, 6, 94, -10, -105, -108, 58, 9, -53, -103, -99, -106, -67, -33, -19, 104, 37, 2, 122, -89, 86, -57, -96, -41, 20, -87, 19, 8, 56, 113, 88, -18, 107, -69, -108, 73, -108, -15, -88, 102, -26, -36, -30, -85, 106, 85, 111, 49, -68, -9, -31, -86, 46, -20, 91, 48, -61, 90, -97, 20, -48, -108, 20, -19, 90, 2, -121, -15, -34, 18, 43, 13, -112, -78, -121, -53, 107, -83, -113, 92, 1, 110, -83, 23, -52, 71, -26, -111, 38, -24, 20, -44, 99, 66, 20, 68, 57, 68, -116, -106, 80, 121, 125, -78, 58, -8, -23, -115, 31, 51, 15, 55, -11, -1, 67, 95, -25, -47, 37, -82, 118, 19, -63, -70, -1, 4, -78, 22, -23, 98, 64, 16, 116, -59, -64, 43, -15, 20, -29, -34, 3, -88, 95, -42, 102, 41, 118, 102, 90, -82, 54, -97, 97, 103, 22, -52, -16, 79, 85, -122, -61, -104, 72, -17, 24, -32, -70, -15, -74, -34, 7, 124, -52, -96, -29, -112, -120, 32, -40, 4, 91, 60, 123, 96, -12, 65, 32, 34, -61, 6, -111, 62, -11, 6, 32, -10, -75, -78, 28, -92, 120, -101, -50, 124, -55, -2, -86, -128, -108, -99, -111, -121, -123, -30, 23, 82, 19, 36, 56, 107, -112, -106, -46, -20, -22, 2, -64, -100, -101, -103, -64, -124, -116, -1, 76, -59, -99, -65, 117, 5, -127, 66, 108, 47, -73, 97, -89, -72, -126, 71, -85, 105, -98, 122, -124, -65, -51, -12, -106, -118, -10, -7, 112, -8, -62, 75, -38, -109, 69, -99, -13, 66, -59, -108, -82, -79, -72, -72, 3, 59, 127, -67, -4, 112, 77, 70, 72, -18, 15, -39, 8, 31, -94, -91, -119, 11, 7, -19, -125, -12, 108, 56, -52, -26, -50]

Table 1: Client RTT Measurements

Connection		Same Computer	Wired Ethernet	Wireless
Packet's RTT (ms)	1	0.375247	1.603828	3.585822
	2	0.244695	1.483122	2.472113
	3	0.215886	1.722347	2.635850
	4	0.154256	1.787988	3.063246
	5	0.146599	1.716876	46.254650
	6	0.184159	2.486335	2.396626
	7	0.135293	1.534905	2.910084
	8	0.160455	1.440819	2.492899
	9	0.167385	1.424774	2.611782
	10	0.151339	1.868581	3.069081
Average RTT(ms)		0.1935314	1.7069575	7.1492147
Maximum RTT(ms)		0.375247	2.486335	46.25465
Minimum RTT(ms)		0.135293	1.424774	2.396626

Table 2: Server Effective Data Rate

Connection	Effective Data Rate (Mbit/s)
Same Computer	80.0
Wired Ethernet	1.43
Wireless	0.33

Analysis of measurements

The measurements of the round trip time of the data packet transmission phase are calculated by recording the time before the DATA packet was sent and the time right after the corresponding DACK packet was received by the client. And calculate the elapsed time by taking the difference of the two values. The measurements also include calculating and determining the average, maximum and minimum RTT values to be displayed.
At the same time the server should calculate the overall data rate based on the total elapsed time of the data transmission phase and total bits of the data. As shown in the table above, the values are in milliseconds, but are in the program are calculated using the java method *System.nanoTime()* instead of the recommended *System.currentTimeMillis()*. The reason is that the granularity of the value calculated by using *System.currentTimeMillis()* depends on the underlying operating system. For Windows, the resolution time for this method is longer than 1 ms, and this is a minimum time needed to update the timer. This also means that if the method was called twice, the result would have two exactly same value. And when calculating the difference, the result will become 0. Or in the other case, the result would have two different values where their difference is equal to the resolute time of the operating system. The method *System.nanoTime()* instead gives more precise results. But the disadvantage of using this method is costing more CPUs.

In table 1, different transmission mediums are showing different results that vary significantly. In the first case, when using the same computer to act as both client and server, the data is transferred at a DRAM(dynamic random-access memory) speed. And the latency in this case is around 14 nano second according to the type of computer that the code is running on. Although, in this project, where UDP was used and no control is being operated in a different layer, the

rate at which packets can be sent from the client is not impacted by the network latency. However, as mentioned, it is in fact influenced by other factors like application and operating system. From the table 2, the effective data rate is almost a factor of 60 larger than the Ethernet case, and almost a factor of 250 larger than the wireless case. Furthermore, in the second case when using Ethernet cable for the network, not only the type of cable is being used will affect the connection speed and quality, but also latency, which defines the delay in how long it takes for traffic to get from the source to the destination, will reduces the effective data rate. In the third case when the client and the server are connected via Wi-Fi, the latency gets even higher, as reflected from the RTT value. The delay is increased even more while using Wi-Fi as the connection of the network because the network connection is affected by the interference of the other end system, if exist, that are connected to the same network and the bandwidth provided for the data transmission. Also, Wi-Fi offers much lower connection speed than that of a Ethernet cable.

Conclusion

This project, which consisted of implementing a reliable data transfer over UDP, was successfully completed with full functions achieved including integrity check, ACK number check, packet type check and error handling. Measurement for RTT values and effective data rate were carried out for various connection settings and tabulated. The performance of the protocol was observed for various failure points and analyzed. Good coding practices with modularized implementation was followed.