NETWORKS AND PROTOCOLS-I (ENTS 640)

Reliable Data Transfer over UDP with Data Transfer Statistics

Aakash Aggarwal (UID:115586437)
Vinayak Agnihotri (UID:115567252)

I. STATEMENT OF PROBLEM

To create “reliable” data transfer between a Transmitter and Receiver over unreliable UDP sockets.

Explanation:

A distributed networking application in Java consisting of a transmitter and a receiver that can gather and display data transfer statistics while ensuring reliable data transfer. The application uses Java’s UDP sockets (classes DatagramPacket and DatagramSocket and their methods) and provide the necessary reliable data transfer functionality on the top of UDP’s unreliable communication. The data transfer is one-directional with data bytes flowing from the client (transmitter) to the server (receiver). While transferring the data, the networking application will measure the round-trip time (RTT) statistics and the overall data rate and display them at the end.

The Data Transfer includes following features:

- Integrity Check
- Reliable Data Transfer
- Data Transfer Statistics
    1. RTT calculations at Transmitter
    2. Data Transfer Rate calculations at Receiver

The Data transfer statistics are recorded under 3 scenarios:

a) when the sender and the receiver protocols run on the same computer as two different processes
b) when the sender and the receiver are connected via wired Ethernet cable
c) when the sender and the receiver are connected via WiFi connection

A comparison of measurement statistics under the above scenarios is presented.

II. INTEGRITY CHECK CALCULATION

The integrity check value is calculated for all packet types by a 16-bit variable that is initialized to zero. Then, the whole packet (except for the integrity check field) is broken up into a sequence of 16-bit words in such a way that the first packet byte in order is the most significant byte, and the second packet byte in order is the least significant byte. If the packet consists of an odd number of bytes, the least significant byte of the last 16-bit word is set to zero. Each 16-bit word is bitwise exclusive OR-ed (XORed) with the content of the variable, and the result is stored back in the variable. After all the 16-bit words have been XORed, the resulting value in the variable is the integrity check value. At the receiver side, the same process is repeated, but this time the integrity check field is included in the XORing process. If the result is 0 (all bits are zero), the integrity check passes. If any nonzero result is obtained, the integrity check fails.

III. PROTOCOL OPERATION

The system works according to the following description. First, we set up the transmitter and receiver UDP socket parameters (IP addresses, port numbers etc.) so that the two sides could communicate with each other.

A. Transmitter Side Operation:

The transmitter should generate 10 data packets, each containing 300 bytes of random data in their payloads, print them on the screen, and send them to the receiver using the following steps:

Initial handshake phase:

1. The transmitter generates a 16-bit random initial sequence number, and send an INIT packet to the receiver. It also starts a timer upon sending the INIT packet, and retransmit the packet if the timer expires before receiving an IACK packet from the receiver. The initial timeout value is set to 1 second and doubled at each timeout event. After the 4th timeout event, the transmitter declares a communication failure and print an error message.

2. It waits for an IACK packet from the receiver. An IACK packet is only accepted if:
a) the integrity check passes
b) it has the correct packet type
c) the ACK number is one larger than the sent initial sequence number.

Data transmission phase:

1. For each 300-byte data block, the transmitter creates a DATA packet and send it to the receiver. Starting from the initial sequence number plus one, the sequence number field is incremented by the length of the current payload for each DATA packet.

2. Each DATA packet is sent to the receiver using the stop-and-wait protocol. The transmitter sends a DATA packet, and wait until the sent DATA packet is acknowledged by the receiver by sending a DACK packet before sending the next DATA packet.
A DACK packet is only accepted if:
a) the integrity check passes
b) it has the correct packet type
c) the acknowledgement number field is correct.
All other received packets are discarded (ignored).

3. The transmitter also starts a timer upon sending each DATA packet, and retransmit the packet if the timer expires before receiving the corresponding DACK packet. The initial timeout value is set to 1 second and doubled at each timeout event. After the 4th timeout event, the transmitter declares communication failure and print an error message. If a DACK packet is received for a DATA packet after some timeout events and retransmissions, the timeout value and the timeout counter is reset to their initial values.

B. Receiver Side Operation:

The receiver protocol operates according to the following description:

Initial handshake phase:

1. The receiver receives the INIT packet, and sends an IACK packet to the transmitter if the INIT packet is correctly received. The INIT packet is correctly received only if:
   a) the integrity check passes
   b) it has the correct packet type.

All other received packets are discarded (ignored).

2. The receiver learns the total number of data packets and the number of payload bytes in each DATA packet from the INIT packet. This way the receiver knows how many DATA packets to expect and the length of the payload in each DATA packet.

3. It sends back an IACK packet with the value of the initial sequence number plus one to indicate that the INIT packet was received. Note that the IACK packet could be lost/corrupted, so the receiver may receive the INIT packet multiple times. It sends back an IACK packet to the transmitter each time a correct INIT packet is received.

Data transfer phase:

1. The receiver receives each DATA packet, and send a DACK packet for each correctly received data packet with the correct ACK number. A DATA packet is correctly received only if:
a) the integrity check passes
b) it has the correct packet type
c) it has the correct (in order) sequence number.

All other received packets should be discarded (ignored).

2. The payload bytes of each correctly received DATA packet is printed on the screen.

IV. RTT & DATA RATE MESUREMENTS

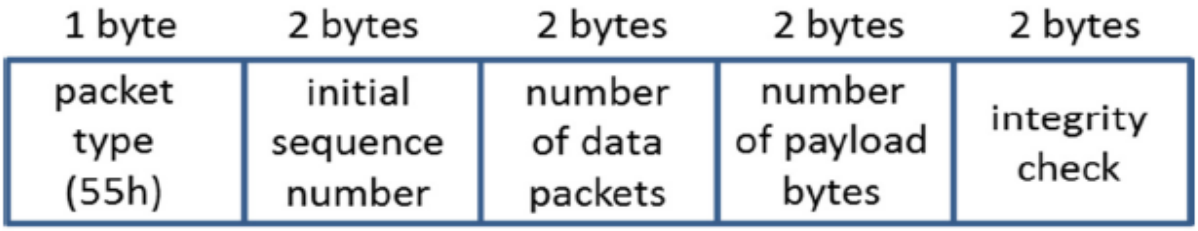
The sender protocol measures the round-trip time (RTT) for each acknowledged DATA packet. Before sending each DATA packet, the sender records the current time using Java's System.currentTimeMillis() method. After receiving the DACK packet for the DATA packet, the sender again records the current time and determine the RTT (in milliseconds) by calculating the difference between the two recorded time values. The RTT measurement is valid for first-time DACK packets. If timeout occurs and the DATA packet is retransmitted, the RTT measurement process starts again. At the end of the data transmission phase, the sender displays the RTT values for all DATA packets, their average, maximum and minimum.

The receiver measures the overall data rate during the data transmission phase. When receiving the first DATA packet, it records the current time as described above. When the receiver receives the last DATA packet, it again records the current time, and calculates the total time it took to receive all DATA packets. At the end of the data transmission phase (after sending the last ACK), it prints out the total effective data rate in megabits per second (Mbit/s), calculated by dividing the total number of received bytes by the total transmission time.

V. PACKET STRUCTURE

The communication session should consist of two phases: an initial 2-way handshake, and the actual data transfer using four types of packets: INIT, IACK, DATA and DACK packets. When sending multibyte values or fields, they should be transmitted in network byte order; that is, the most significant byte should be transmitted first, and the least significant byte should be transmitted last.

The initial 2-way handshake should consist of the following steps. First, the client/transmitter should send an **INIT packet** to the server/receiver with the fields shown below:



The INIT packet should have the following fields ('h' stands for hexadecimal values):

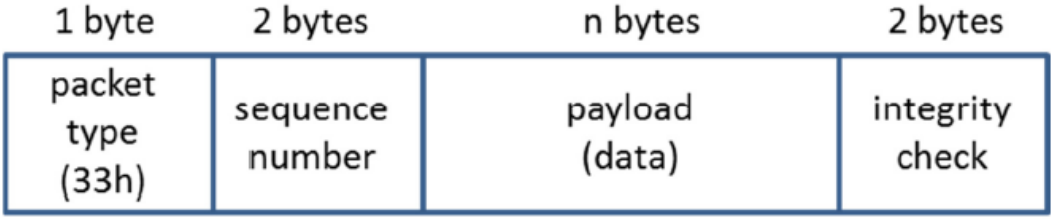
- Packet type (1 byte): This field describes the type of the packet, and it should have a value of 55h.
- Initial sequence number (2 bytes): Initial sequence number randomly generated by the transmitter, interpreted as a 16-bit unsigned integer. The sequence number should count bytes (and NOT packets) and wrap around after reaching its maximum value.
- Number of data packets (2 bytes): The total number of DATA packets the transmitter will send to the receiver in this communication session.
- Number of payload bytes (2 bytes): The number of data bytes in the payload. For a communication session, the transmitter will send the same number of bytes in each DATA packet.
- Integrity check (2 bytes): Integrity check value calculated over the whole packet (except for the Integrity check field). The algorithm to calculate this field will be described later.

Then, the receiver should respond with **an IACK packet**, whose structure is shown below:

The IACK packet should have the following fields:

- Packet type (1 byte): This field describes the type of the packet, and it should have a value of AA h.
- Acknowledgment (ACK) number (2 bytes): It should be the value of the received initial sequence number plus one (INIT packets consume one sequence number).
- Integrity check (2 bytes): Integrity check value calculated over the whole packet (except for the Integrity check field).

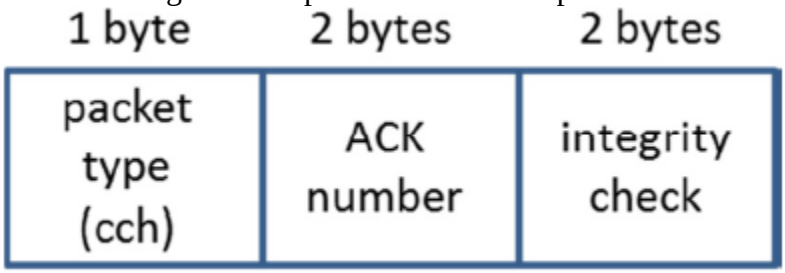
During the second phase, the data transmission phase, the transmitter should send DATA packets to the receiver. The structure of the **DATA packets** is given as:



The DATA packets should have the following fields:

- Packet type (1 byte): This field describes the type of the packet and its value should be 33h.
- Sequence number (2 bytes): The sequence number of this DATA packet, which is the sequence number of the first payload byte in each packet (the sequence number counts bytes). It should start with the value of the initial sequence number sent to the receiver in the 2-way handshake plus one (the IACK packet consumed one sequence number), and it should be incremented for the next DATA packet according to the number of payload bytes sent in the current DATA packet.
- Payload: The user data carried by the DATA packet. The receiver learns the number of payload bytes (and the number of sent data packets) from the INIT packet.
- Integrity check (2 bytes): Integrity check value calculated over the whole packet (except for the Integrity check field).

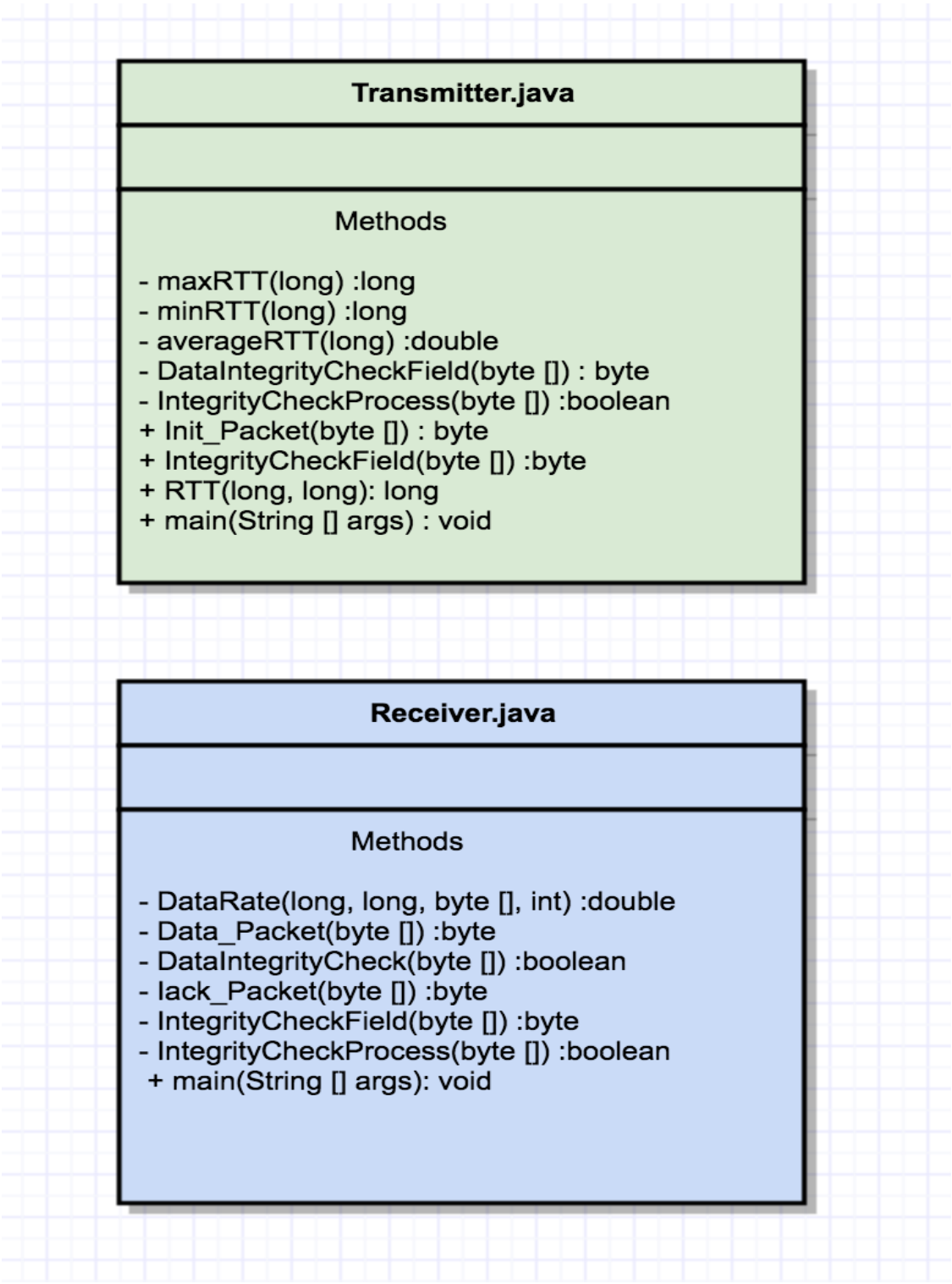
Upon the reception of a DATA packet, the receiver should send a **DACK packet** to the transmitter to acknowledge the reception of the DATA packet with the following fields:



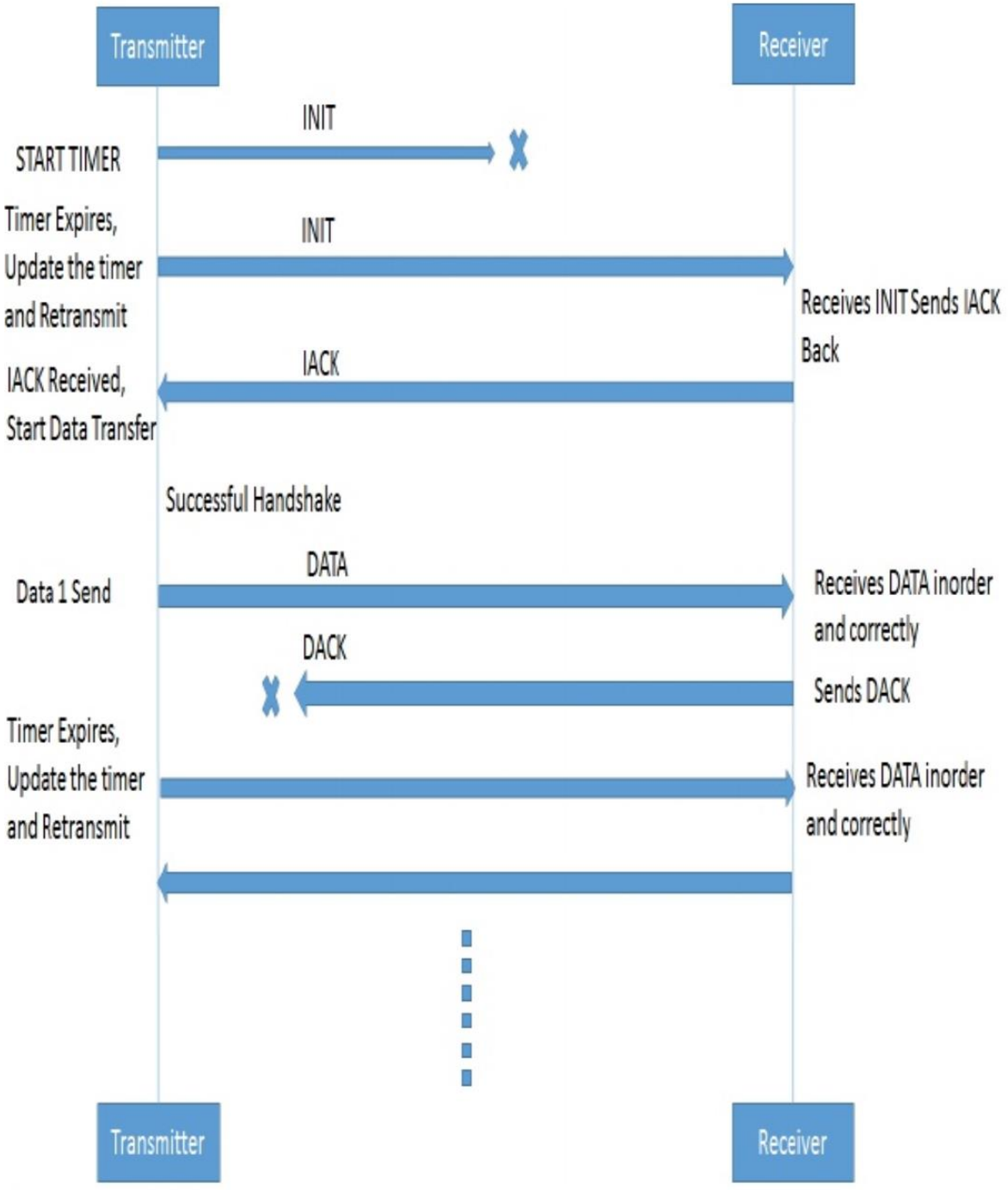
The DACK packet should have the following fields:

- Packet type (1 byte): This field describes the type of the packet, and it should have a value of cch.
- Acknowledgment (ACK) number (2 bytes): The sequence number of the next data byte the receiver is expecting from the transmitter. That is, the sequence number of the last correctly received byte plus one.
- Integrity check (2 bytes): Integrity check value calculated over the whole packet (except for the Integrity check field).

VI. UML DIAGRAM



VII. PROTOCOL STATE DIAGRAM



VIII. FUNCTION DESCRIPTION

Transmitter functions:

1. **maxRTT( ):** Calculates maximum Round Trip Time value from all the RTT values stores in rtt[ ] array.

2. **minRTT( ):** Calculates minimum Round Trip Time value from all RTT values stored in rtt [ ] array.

3. **averageRTT( ):** Calculates the average of all RTT values in the rtt [ ] array from all the sent packets in milliseconds.

4. **DataIntegrityCheckField( ):** Creates the integrity check field value for the data packet. It does so by XORing a 16 bit variable initialized to 0 with all fields in the data packet grouped in 16 bits. It XORs the first byte of the variable with the even numbered fields and the second byte with the odd numbered fields for the 303 bytes long data buffer. For odd number of total fields, second variable byte is XORed with 0 at the end. The variable value is returned and stored at the 304^{th} and 305^{th} positions in the data buffer.

5. **IntegrityCheckProcess( ) :** Does the Integrity check process for received IACK and DACK packets. Here all the fields including the checksum are XORed with a 16 bit variable initialized to 0. If the answer comes out to 0, integrity check is successful and handshake will be successful. In the case of DACK packet, once acknowledgement is received ie integrity check is successful, the next data buffer is created and sent and timer starts for that.

6. **Init_Packet( ) :** Creates the INIT buffer which stores the appropriate packet type, sequence number etc for the INIT packet.

7. **IntegrityCheckField( )** : Creates the checksum value for the INIT buffer by XORing all the grouped bytes together. Returns the checksum which is stored in the appropriate positions in the INIT buffer in the Init_Packet method.

8. **RTT( ):** Calculates round trip time for a packet. It is called in main every time the times of sending data packet and receiving the appropriate acknowledgement are recorded. It then calculates time elapsed between the two.

Receiver functions:

1. **DataRate( ):** Data Rate method calculates the data rate for all the packets received cumulatively. The round trip time is calculated in similar fashion as in the transmitter but for all the 10 packets received (after the fist packet is received and before the last acknowledgement is sent). Total number of bits that are received in each packet is multiplied by the number of packets. The resultant is divided by the round-trip time calculated earlier and data rate is obtained in Mega Bits per Seconds. The function takes 4 arguments, start time, end time, count of data packets and size of each data packet.

2. **Dack_packet( ):** Creates the DACK packet with the appropriate acknowledgement number, packet type, checksum etc. The acknowledgement number is 300 more than the received data sequence number and wraps around if it reaches maximum value.

3. **DataIntegrityCheck( ):** Method to do integrity check on received data packets. All the fields are XORed in groups of 2 bytes and if result is 0, integrity check is successful and data acknowledgement is sent to the transmitter.

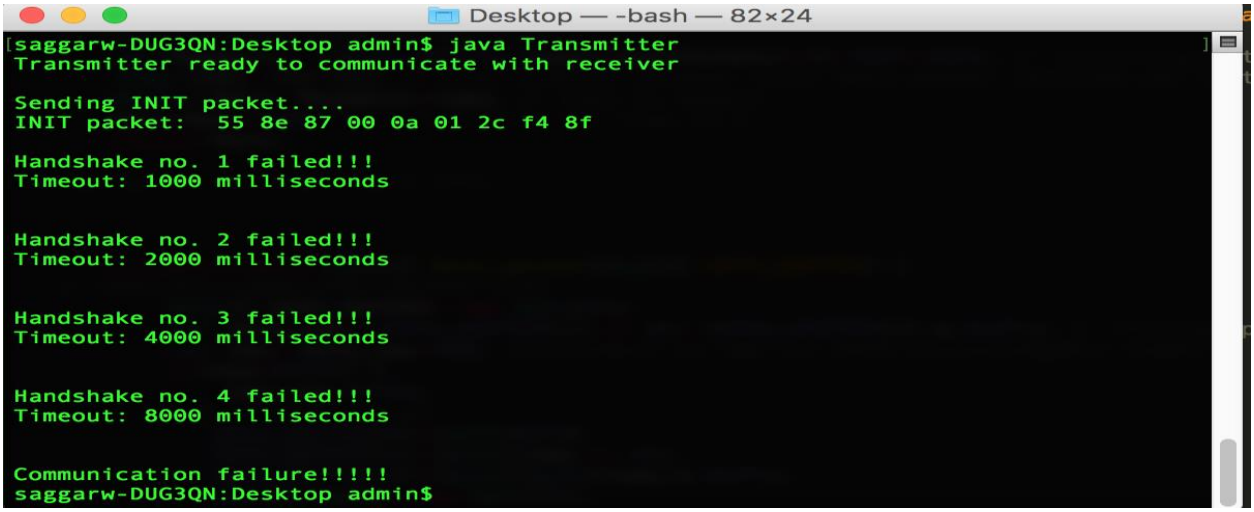
4. **Iack_Packet( ):** Creates the IACK buffer with appropriate field values like packet type, acknowledgment number etc. Wraps around for exceeding sequence numbers and send back the received sequence number plus one to complete handshake.

5. **IntegrityCheckField( ):** Calculates the integrity checksum value for both the IACK and DACK buffers by XORing all current fields.

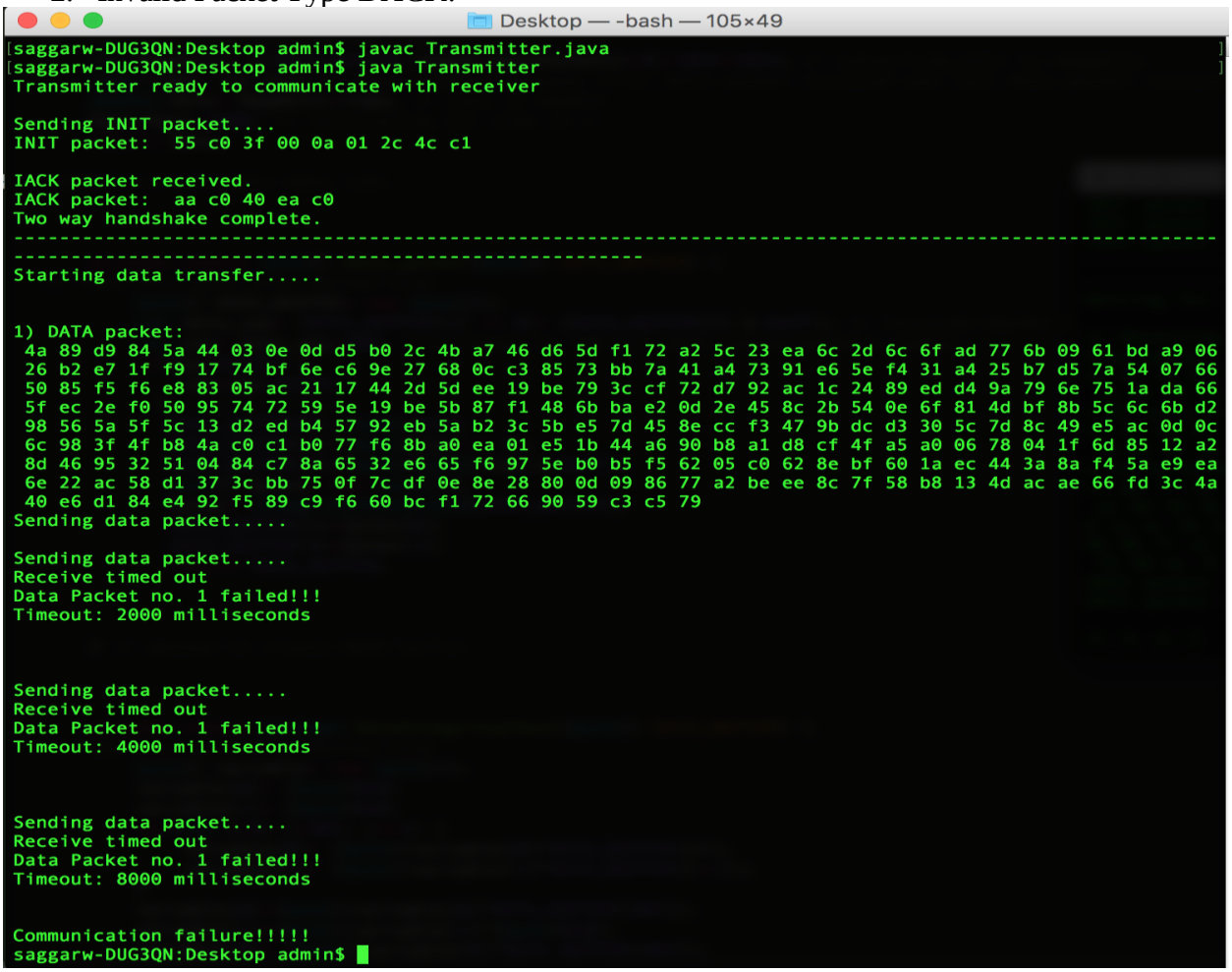
6. **IntegrityCheckProcess( ):** Method to do integrity check process for the received INIT packet by XORing all fields including the checksum. If integrity check is successful, IACK packet is sent.

IX. Output

1. When Receiver is not working:



2. Invalid Packet Type DACK:



3. Transmission of packets when both Transmitter and Receiver run on same machine as different processes:

Transmitter Output:

```
saggarw-DUG3QN:Desktop admin$ javac Transmitter.java
saggarw-DUG3QN:Desktop admin$ java Transmitter
Transmitter ready to communicate with receiver

Sending INIT packet....
INIT packet:  55 1a 20 00 0a 01 2c 53 1b

IACK packet received.
IACK packet:  aa 1a 21 8b 1a
Two way handshake complete.
---------------------------------------------------------------------------------------------------------------------------------------------------------
Starting data transfer.....


1) DATA packet:
 bc e6 26 ae 86 d7 94 13 75 73 38 08 06 78 b2 2f ca b8 75 24 2c 6f af 5c 1e 5e b7 a3 51 86 39 bd ee 3f 6e f3 f9 68 87 1f 7d b1 29 fa
86 cc d9 35 6c ec 34 e0 8d ad 4d f0 28 c2 e1 f3 82 8c a8 2a 5c c9 d5 f1 de 5c 88 59 cd 71 94 9f f4 e4 4f 69 44 b3 63 bd 04 6a a4 e7 75
00 c9 70 53 03 5e ea 03 d5 81 a9 51 a0 e4 4a 0c 2e 9e 77 11 01 01 0a f0 8e 0d e3 a4 65 93 76 20 70 01 5b 10 e0 86 34 42 2c 5d 1f fe
cd dd 9c e9 3f fd cc 55 5a 7c 29 b1 0a ea 70 55 d4 13 5e 89 c4 8f 97 fb 78 bc 3e 5d 18 83 97 a8 cc 72 11 93 5d cb ff 69 26 6e fb ef cb
8e af 53 63 5c a2 d1 d1 16 99 c7 2f 0c c9 fd fa b1 b2 a4 bd 2d 98 41 26 7b 43 ad 69 67 d9 b3 32 a3 41 ad 8f 0f d6 61 61 49 c2 1d ec
59 33 ae 76 48 f1 5b 3b 51 dc ac be 42 6c b3 99 dd c7 2b 8a b7 3e c7 4c 6e 93 63 16 12 00 1a 66 a7 9d 60 27 8f 2c cc e7 fe 3a 90 7d
8d bf c2 0f 8c 5d 29 c9 a1 d2 d8 fa c8 21 47 62 23 f6 cf 40 0a 9d e7 ea 50 2d 07 4b 31 89 2f a8 46 db
Sending data packet.....
DACK packet received
DACK packet:  cc 1b 4d 81 1b
Round Trip Time: 13 ms

2) DATA packet:
 95 a6 35 a3 14 92 4e 9b 45 bc 04 27 a2 ab 40 3f 2a 10 a7 7a 0b de 3b 74 8a 49 66 4c f3 fc 61 af 14 6b 3c a4 f2 32 61 16 94 a2 2a 1c
78 5f 46 8f 94 44 bd af 6c 7d 4c b7 d8 b4 ae b3 a3 d7 2b d3 92 1d 59 c5 af 78 3f 7b b9 fa ee 22 8f c5 39 40 f5 73 69 0d 44 1f 9f 68 f7
96 fa 1e 64 cd 1d b9 35 8d 3a 3c 0f e4 e2 6a 65 33 b8 36 00 62 e4 fb 2d d6 fd fa 8d f7 4f f3 de 96 8d 55 d3 d5 fc a8 35 ec e1 d5 5d 1d
cd a7 79 26 16 d3 0a 16 f3 1f 70 a3 79 d2 01 d6 df a9 fb c5 23 99 ef 73 67 fa 62 9e 5c ca e5 e0 ab 89 8d cb 08 7d 8f 20 6f 43 c4 4a 81
e1 9f 5a da 6e ae fa e0 2d 9c 3e 0e 6f 9b 57 79 c7 77 fb dd 63 6a ba 7a aa 95 ae 26 25 86 f8 b6 13 2a 3a 5e c3 cc fa e0 1d 23 f9 44 2b
58 8a 92 b8 eb 56 eb 53 90 7d 99 0c c6 fb 66 bb e6 6c 2d d4 aa 92 b9 af 1c 76 b2 9d b0 3b 6e e9 21 26 c8 ba 97 16 42 bd b1 b4 f5 6c
f0 02 aa 07 66 22 ef f0 e9 4f 8c 8e dd d0 46 16 d3 ab 03 53 14 a6 42 7e db c3 ca 4f 34 07 95 a0
Sending data packet.....
DACK packet received
DACK packet:  cc 1c 79 b5 1c
Round Trip Time: 10 ms

3) DATA packet:
 27 7e 75 19 67 c1 ca bb 7b ef 39 8f c1 37 8f 57 4c 9d ac 31 ee 8a 66 a2 f4 b4 e2 a9 40 d4 fb d3 24 ec 7b db c4 7f c6 9a b2 f4 74 c2 40
e5 e8 8a ed ca 8a c8 2a 9d 12 5a c6 29 71 e6 5e 4f a3 2e 56 d5 fd 91 64 55 60 6f 8e 73 ff 13 79 20 07 f9 0c 24 df 68 c0 73 71 d7 6d 43
c2 4e 4d 3d 33 0b 94 93 d0 8e d7 bc ae af be b1 39 b7 5f c5 17 8a 5b 24 9b 3a 12 cf f5 41 05 ae d2 fa 20 51 2c 82 0b c8 cb c9 70 07
46 26 67 7e bc dc 4d 6e c9 11 1f 30 bf 92 85 6e d0 15 5b 42 52 5a 94 fe 72 4b b9 71 83 86 c5 f1 aa 70 a2 6c 8d 60 7c 5c 17 56 b2 92
0f 61 02 6d a7 14 74 68 d1 b5 85 6a 21 53 f2 53 be 05 32 48 d3 6c fb dd af a3 8e f0 79 af 02 cd fa 06 76 ad 2c 0e 04 4a 48 c2 77 eb 76
8f df ac e2 3f 57 9b 66 83 f3 dc dc b1 06 b3 7d b4 10 34 8b ee 52 57 65 57 f8 d4 61 6a 0e 1a 07 68 fa 3b d3 52 96 04 b1 00 7f c7 6b
63 cd ec 02 e1 73 df 7d 79 0b 63 b4 0f ed 8d 34 34 96 71 8a 20 94 98 f6 62 96 fa 79 3a 88 c9 b5 62
Sending data packet.....
DACK packet received
DACK packet:  cc 1d a5 69 1d
Round Trip Time: 9 ms

4) DATA packet:
 0f 9f ba 48 70 97 64 39 b0 45 a5 d0 1f 06 be 5d 8c a5 ba b9 63 ac ce ab e1 f7 9c 3a 3b 3f aa 1e 0e 3b 8e 10 fb 4d 78 a4 8a 38 5f aa be
ec 99 59 06 60 a0 df 5a c4 40 24 55 27 07 f9 db 74 e0 56 ce 3d 6d bf 96 85 c4 aa 46 b3 0a 18 47 b6 8c 1f d9 85 41 54 e2 76 51 74 0e
af f0 29 fb 27 a3 2e 2a 10 34 3d ef 9b 7b 29 a1 56 63 42 38 b1 73 59 09 6d 76 bc 8f d8 42 56 06 1a 6d c8 cb 7d 26 15 f4 5d b0 81 9b
a8 a0 da 6d c1 36 f4 92 52 8b 6f d4 69 e5 41 17 f2 5e 92 86 3e 7b 37 3e 26 eb 67 70 00 a3 a4 93 55 87 0c 10 bf 56 2e 59 02 54 fa 26
e1 28 27 d3 a0 cc 55 af d4 5b 72 8a 70 d3 90 d7 b8 24 83 b2 47 46 7b 02 c6 32 dd 18 f8 93 9a 23 2b b5 b2 43 07 f0 8d f4 5c b4 4f ac
6f 16 75 d6 ba f7 15 38 17 de 01 52 c6 b2 5f 5a 27 08 58 a1 45 a8 7a 87 8d 38 58 9e 56 b8 a6 e2 2e c3 8c 0c 07 90 bb 4a d8 16 0f 22
48 2d 28 32 3c 96 67 ab 56 ed 76 e6 71 7e f0 47 c6 d2 12 2e f6 c5 7e 8f 95 dc 87 6f 33 56 88 26 da 92 41
Sending data packet.....
```

DACK packet received
DACK packet: cc 1e d1 1d 1e
Round Trip Time: 6 ms

5) DATA packet:
c4 40 bb bf d5 2c c1 76 24 b3 5a c7 e3 01 e1 d1 90 c3 a0 5b 97 38 06 e7 c0 5e d7 10 b9 55 68 50 50 f7 5c fd 8d 10 7a 4a 17 57 71 b6 af 3b 44 85 24 e3 90 c3 2b bc 83 31 68 57 97 44 3e e3 b6 3b 52 5a 3f bb 6b d9 94 8b ea 35 06 36 c8 4d 13 aa 35 d3 79 49 ab 5a d0 af dc 95 92 37 21 19 0c 07 75 39 8f b1 d5 be c4 6f db 26 13 69 51 ee c5 db 78 2f 19 ab c0 27 cf d2 73 05 df 88 97 98 db 73 17 73 17 70 d5 f2 2e a2 f7 10 79 22 b1 08 de a2 94 59 10 d2 f5 b7 27 74 94 4a 3a 89 5a c8 ba 8e cc 96 28 69 9a d2 09 d2 a4 91 5a 5a 14 a2 17 5c a8 c3 93 f0 3b d9 d7 37 3f 51 d8 c7 27 81 5a 95 d9 57 e2 33 f5 a6 c1 76 18 f3 b4 2c e8 cd 00 4b c3 0a c9 57 67 db 13 cc 85 2b 05 9c 3e ca 59 1c b9 cf 48 c2 dd f4 c4 3e df 72 b1 cc 62 b2 eb 36 6b e1 dd e0 af 68 b7 6c 0c 40 b4 14 e5 a5 03 91 9c 90 cf aa eb 5d 5a 60 59 d8 68 0d f7 9c 6c 1d 17 ca 21 19 31 ee c8 b5 8c c8 56 00 e5 ed 49 e3 e3 87 44 30 de 10 22 ce 39 5b 13 be
Sending data packet.....
DACK packet received
DACK packet: cc 1f fd 31 1f
Round Trip Time: 4 ms

6) DATA packet:
28 7b e7 70 ac e6 c2 a1 cd ef 2b 0b 19 62 1f fa f6 f3 40 02 02 04 cf af 14 18 44 62 9c 6d 76 f8 9c 88 c5 95 c5 8f 26 22 b3 01 75 93 40 03 d1 de 37 7f d2 1c 41 bc fe f0 9b fa 22 f5 96 32 2f b5 2c aa 38 14 de 2c bd 3c b0 c1 69 64 b3 94 d2 df 8d ba 07 7d b6 e2 bf 94 cd 39 39 77 db 1c fc bd c0 d4 d1 7a 67 de de 50 0b c9 5b 17 7b c0 e9 2f 62 4b 81 b2 29 8b 85 fa 43 e8 ec 26 5c 2a f2 e7 68 a4 14 02 22 47 8f 69 d0 a6 2a 72 16 ed cb 0d 87 4c 33 18 f8 9c f3 ff 99 26 f5 3f c2 e8 f0 19 d5 9c cf 04 a8 02 04 3b 4e 54 fb 65 f2 d7 e5 70 b0 16 31 28 96 31 c6 9a f0 5f c4 7f e5 37 39 e6 de f9 0b 26 e4 6b 13 f5 97 b9 b9 2c f7 ff 8b 1c f5 98 f9 54 06 43 d6 8b 6f 96 4d 71 56 dc 29 96 fd 0f f7 c9 96 b3 45 61 03 42 96 2e fa df dd 0f 53 41 c6 c1 64 aa 95 13 bd bf 76 4a d6 e1 0b 79 d0 3c 67 11 d1 46 82 20 ad 80 f1 8c 4b f9 e5 07 29 75 9e 21 5c 6e 9c 81 89 31 ca 5d dd cd 18 ba 98 67 de 19 7d 85 93 a1 f4 e4 67 ac
Sending data packet.....
DACK packet received
DACK packet: cc 21 29 e5 21
Round Trip Time: 3 ms

7) DATA packet:
a6 d9 77 10 be e5 81 58 b6 b0 98 13 41 5e 41 cd c9 87 ad 21 63 bb 76 aa 20 32 17 18 68 ba 9f 6d 8e 38 8c 3c d2 5d d6 f8 e0 ee 27 41 f9 f0 f8 f5 62 28 ac 4b ad 72 c3 61 ef 7a 1d 27 36 c6 d2 af 2e 52 75 25 24 8f 29 ec ec 50 c8 9a d1 c3 70 5e b7 b4 8d b1 86 9b 38 3a 1e 30 e7 a3 9e d8 d3 ac 47 0a 1c ba 1b 5c fa 16 b9 d6 30 1e 8a bf e7 76 8d d6 8b 5f 30 33 77 6b 60 9c c8 62 cb 61 ab 9c c1 ee e9 e8 72 70 9c d1 eb b0 03 b9 58 a8 40 11 0b e0 be 12 7f c1 67 7b 69 03 7d eb 9b 70 a3 a5 25 d8 be 6c ba 97 00 d1 68 15 d2 21 65 98 8e 1e 3e f4 fc 11 5e 97 a1 69 73 3e 43 54 22 55 11 22 21 1a ca 79 d5 63 c1 96 dc 23 16 69 4d 4a c6 92 96 bf 45 17 4a 5d 0b f0 ba 68 41 64 25 13 c1 d7 08 44 29 6d 7c 23 ae 4e 35 da d4 64 a1 33 5e 20 67 c0 bc 78 d5 f1 8e a8 00 9c 01 59 a0 a4 6c e5 4f 71 6e 5c bb 28 4e cd 27 19 da fd 64 6f e7 2c 6a 1f 74 de 25 98 fa 48 0a bd 19 93 41 0c e9 8d 11 2a 70 24 70 5f 16 98 a8 b8 c0 91
Sending data packet.....
DACK packet received
DACK packet: cc 22 55 99 22
Round Trip Time: 4 ms

8) DATA packet:
73 6d f0 a0 f5 88 6c da 59 2d 51 5e 21 3e 7d a9 56 53 95 0d 6d 79 92 3a 9c 4c 81 de 99 60 fb 8c b9 c8 3e 23 cc 0f 72 19 c1 d6 f8 61 5d 50 33 33 f9 51 a3 09 e9 42 48 ec b7 47 01 ec 4f 29 ae c0 5a 7c 9b fa 48 e4 b7 d6 87 ab ca 16 b4 91 fe 16 07 20 8a 66 d4 53 2d 2a fd d9 42 a6 07 77 2a ed 56 3e 78 44 cb 3a 1d 6a 8e 48 b2 a3 09 44 33 ac 4f 00 1a 8b f9 81 7f 69 75 96 39 74 1c e4 a2 37 6e 99 46 2f 28 f7 46 5e 1e 37 59 c0 d1 27 95 31 a8 12 c5 9c d6 55 a9 ee c8 de 5e f1 bf 5e 1a 11 ab 41 ec 7b 28 a7 12 f9 65 95 2a 09 96 3a 49 32 6a 70 b4 49 1f 2d da 15 c4 93 77 53 73 4f 23 de d8 78 bd 24 09 4d 7e f0 f1 4e f3 f3 75 c5 d0 b6 f3 58 b4 90 39 2d 90 a8 db 55 2f 54 8f 0f ed 5f 03 83 af 6e f9 f9 ce 77 37 e5 20 86 9f bb e7 1b 23 6e c9 24 bf 05 7e 1e a4 55 ab 8d bf af 19 3c 00 2b b5 cd 9f 38 00 c0 10 c8 e0 4b db 68 60 0a e0 00 81 32 01 83 76 4b 59 b4 76 3b 84 0e 16 76 e6 56 77 71 93 47 c6 9d 37 3e a2 f5
Sending data packet.....
DACK packet received
DACK packet: cc 23 81 4d 23
Round Trip Time: 4 ms

9) DATA packet:
20 7c a2 24 24 62 57 23 11 c2 4c f9 5b b7 17 22 6f f3 20 34 ec 94 fc 7c 8d 46 0b 61 a9 81 63 b8 47 3d dc 75 78 2a a9 54 00 1a 6c 83 c5 29 a1 7b 90 ed d3 c4 29 40 6d 29 bd 76 db 48 ca 0d c6 61 2f a0 a0 75 48 fb b8 19 0a b8 71 cf 2d d2 31 0f 88 14 16 1b 3a d8 ca 16 4b cb 03 88 0e e6 84 81 44 fd ec d3 f1 38 57 07 5f 0f 67 6c d0 78 6a cd 54 88 d3 48 a6 e6 75 aa 49 46 09 ab eb b4 93 ac 34 e3 af ea c4 7a 79 c3 13 7d e3 32 00 54 ab ce 74 3f d0 1b 27 e3 1f 69 ee 0d d9 f9 2d e0 e5 5c 73 1b 78 e0 85 75 ca 2d 9e 20 02 05 a2 36 44 85 6d d3 f2 a3 2c 46 5f de 14 4d 73 28 74 1f e3 a5 b8 be 4f fb ed b5 5c 61 5c 44 0a 09 b1 0e 0c 5b 4a 5d 6c 06 58 6b c3 9a b9 a3 e1 30 b3 62 68 21 7e 59 35 d7 28 2b 0a 63 0d 56 8c 6c b9 b3 f9 e1 17 62 4f d6 01 44 92 8a 7e 27 5b 3e b3 08 8f dc 84 20 e5 6f 58 81 4a 77 56 6f f5 f3 5c 05 98 58 65 cf 67 eb b1 d9 99 42 2b f4 b7 eb 8e 65 b4 f8 4e 35 e4 71 10 4c 8a e9 3f 6e db 58
Sending data packet.....
DACK packet received
DACK packet: cc 24 ad 61 24
Round Trip Time: 2 ms

10) DATA packet:

49 5b c3 2a 07 1c 27 b3 01 4d c3 8d 80 bd fa fe a3 13 ec 53 e1 34 0c a1 1c 9d 82 d1 85 ae a3 e7 34 6f 0b 0b 44 d6 f8 f1 91 79 ee 82 6a 11 a3 03 2d 24 a8 65 e0 c7 c0 27 19 0f 25 95 ba 25 2c 76 8a 39 6d e5 93 06 4a af 78 89 c9 8d 4b de 15 62 63 c2 c0 e3 31 ff 77 a5 f0 30 a8 d8 a0 a3 21 b8 81 7a ba e4 88 31 49 6b f6 6a 7a 38 8e 58 01 11 33 18 0f 53 9e 7f fd 5b 28 92 1d cb 21 85 a6 6b 62 67 7e ea 3c ad bb 58 7d d9 4a 35 bf cb 9e 8c e3 81 9f 98 af d3 90 bf bd ae 0d 00 8e 0c e0 80 0c b9 c7 ed cd cd 0c a0 34 50 19 94 13 db a8 9f 89 04 7e 3e 36 e0 6a b2 a2 89 f5 13 1b 2a 40 6a 70 0a b6 4a db 21 66 5f 75 cc 05 59 de 51 a9 0f 91 14 2b f0 c4 68 cd 6a 39 17 ed b7 60 d4 0f c4 13 87 e4 f9 0d bf e2 f9 4f 60 04 d2 1a 84 00 ca 3c ac 72 3d 31 a4 ce 64 dc 77 c0 72 da f0 bb 24 8f e9 0b b0 8a 9e 2f b8 04 95 3f 54 e6 3e 48 b9 1b b2 9f 0f 18 e1 52 f5 58 02 06 25 67 cc a6 d5 11 ca 75 d7 81 09 2b f5 8c 9d 74 7d
Sending data packet.....
DACK packet received
DACK packet: cc 25 d9 15 25
Round Trip Time: 4 ms

Data transfer statistics:

Average Round Trip Time: 5.9 ms
Minimum Round Trip Time: 2 ms
Maximum Round Trip Time: 13 ms
---
saggarw-DUG3QN:Desktop admin$

Receiver Output:

saggarw-DUG3QN:Desktop admin$ javac Receiver.java
saggarw-DUG3QN:Desktop admin$ java Receiver
Waiting for INIT packet from transmitter.....
INIT packet received
INIT packet: 55 1a 20 00 0a 01 2c 53 1b
IACK packet sent

Waiting for data.....

1) Receiving data packet....
Data packet: bc e6 26 ae 86 d7 94 13 75 73 38 08 06 78 b2 2f ca b8 75 24 2c 6f af 5c 1e 5e b7 a3 51 86 39 bd ee 3f 6e f3 f9 68 87 1f 7d b1 29 fa 86 cc d9 35 6c ec 34 e0 8d ad 4d f0 28 c2 e1 f3 82 8c a8 2a 5c c9 d5 f1 de 5c 88 59 cd 71 94 9f f4 e4 4f 69 44 b3 63 bd 04 6a a4 e7 75 00 c9 70 53 03 5e ea 03 d5 81 a9 51 a0 e4 4a 0c 2e 9e 77 11 01 01 0a f0 8e 0d e3 a4 65 93 76 20 70 01 5b 10 e0 86 34 42 2c 5d 1f fe cd dd 9c e9 3f fd cc 55 5a 7c 29 b1 0a ea 70 55 d4 13 5e 89 c4 8f 97 fb 78 bc 3e 5d 18 83 97 a8 cc 72 11 93 5d cb ff 69 26 6e fb ef cb 8e af 53 63 5c a2 d1 d1 16 99 c7 2f 0c c9 fd fa b1 b2 a4 bd 2d 98 41 26 7b 43 ad 69 67 d9 b3 32 a3 41 ad 8f 0f d6 61 61 49 c2 1d ec 59 33 ae 76 48 f1 5b 3b 51 dc ac be 42 6c b3 99 dd c7 2b 8a b7 3e c7 4c 6e 93 63 16 12 00 1a 66 a7 9d 60 27 8f 2c cc e7 fe 3a 90 7d 8d bf c2 0f 8c 5d 29 c9 a1 d2 d8 fa c8 21 47 62 23 f6 cf 40 0a 9d e7 ea 50 2d 07 4b 31 89 2f a8 46 db
DACK packet: cc 1b 4d 81 1b
DACK packet sent

2) Receiving data packet....
Data packet: 95 a6 35 a3 14 92 4e 9b 45 bc 04 27 a2 ab 40 3f 2a 10 a7 7a 0b de 3b 74 8a 49 66 4c f3 fc 61 af 14 6b 3c a4 f2 32 61 16 94 a2 2a 1c 78 5f 46 8f 94 44 bd af 6c 7d 4c b7 d8 b4 ae b3 a3 d7 2b d3 92 1d 59 c5 af 78 3f 7b b9 fa ee 22 8f c5 39 40 f5 73 69 0d 44 1f 9f 68 f7 96 fa 1e 64 cd 1d b9 35 8d 3a 3c 0f e4 e2 6a 65 33 b8 36 00 62 e4 fb 2d d6 fd fa 8d f7 4f f3 de 96 8d 55 d3 d5 fc a8 35 ec e1 d5 5d 1d cd a7 79 26 16 d3 0a 16 f3 1f 70 a3 79 d2 01 d6 df a9 fb c5 23 99 ef 73 67 fa 62 9e 5c ca e5 e0 ab 89 8d cb 08 7d 8f 20 6f 43 c4 4a 81 e1 9f 5a da 6e ae fa e0 2d 9c 3e 0e 6f 9b 57 79 c7 77 fb dd 63 6a ba 7a aa 95 ae 26 25 86 f8 b6 13 2a 3a 5e c3 cc fa e0 1d 23 f9 44 2b 58 8a 92 b8 eb 56 eb 53 90 7d 99 0c c6 fb 66 bb e6 6c 2d d4 aa 92 b9 af 1c 76 b2 9d b0 3b 6e e9 21 26 c8 ba 97 16 42 bd b1 b4 f5 6c f0 02 aa 07 66 22 ef f0 e9 4f 8c 8e dd d0 46 16 d3 ab 03 53 14 a6 42 7e db c3 ca 4f 34 07 95 a0
DACK packet: cc 1c 79 b5 1c
DACK packet sent

3) Receiving data packet....
Data packet: 27 7e 75 19 67 c1 ca bb 7b ef 39 8f c1 37 8f 57 4c 9d ac 31 ee 8a 66 a2 f4 b4 e2 a9 40 d4 fb d3 24 ec 7b db c4 7f c6 9a b2 f4 74 c2 40 e5 e8 8a ed ca 8a c8 2a 9d 12 5a c6 29 71 e6 5e 4f a3 2e 56 d5 fd 91 64 55 60 6f 8e 73 ff 13 79 20 07 f9 0c 24 df 68 c0 73 71 d7 6d 43 c2 4e 4d 3d 33 0b 94 93 d0 8e d7 bc ae af be b1 39 b7 5f c5 17 8a 5b 24 9b 3a 12 cf f5 41 05 ae d2 fa 20 51 2c 82 0b c8 cb c9 70 07 46 26 67 7e bc dc 4d 6e c9 11 1f 30 bf 92 85 6e d0 15 5b 42 52 5a 94 fe 72 4b b9 71 83 86 c5 f1 aa 70 a2 6c 8d 60 7c 5c 17 56 b2 92 0f 61 02 6d a7 14 74 68 d1 b5 85 6a 21 53 f2 53 be 05 32 48 d3 6c fb dd af a3 8e f0 79 af 02 cd fa 06 76 ad 2c 0e 04 4a 48 c2 77 eb 76 8f df ac e2 3f 57 9b 66 83 f3 dc dc b1 06 b3 7d b4 10 34 8b ee 52 57 65 57 f8 d4 61 6a 0e 1a 07 68 fa 3b d3 52 96 04 b1 00 7f c7 6b 63 cd ec 02 e1 73 df 7d 79 0b 63 b4 0f ed 8d 34 34 96 71 8a 20 94 98 f6 62 96 fa 79 3a 88 c9 b5 62
DACK packet: cc 1d a5 69 1d

DACK packet sent

4) Receiving data packet....
Data packet: 0f 9f ba 48 70 97 64 39 b0 45 a5 d0 1f 06 be 5d 8c a5 ba b9 63 ac ce ab e1 f7 9c 3a 3b 3f aa 1e 0e 3b 8e 10 fb 4d 78 a4 8a 38 5f aa be ec 99 59 06 60 a0 df 5a c4 40 24 55 27 07 f9 db 74 e0 56 ce 3d 6d bf 96 85 c4 aa 46 b3 0a 18 47 b6 8c 1f d9 85 41 54 e2 76 51 74 0e af f0 29 fb 27 a3 2e 2a 10 34 3d ef 9b 7b 29 a1 56 63 42 38 b1 73 59 09 6d 76 bc 8f d8 42 56 06 1a 6d c8 cb 7d 26 15 f4 5d b0 81 9b a8 a0 da 6d c1 36 f4 92 52 8b 6f d4 69 e5 41 17 f2 5e 92 86 3e 7b 37 3e 26 eb 67 70 00 a3 a4 93 55 87 0c 10 bf 56 2e 59 02 54 fa 26 e1 28 27 d3 a0 cc 55 af d4 5b 72 8a 70 d3 90 d7 b8 24 83 b2 47 46 7b 02 c6 32 dd 18 f8 93 9a 23 2b b5 b2 43 07 f0 8d f4 5c b4 4f ac 6f 16 75 d6 ba f7 15 38 17 de 01 52 c6 b2 5f 5a 27 08 58 a1 45 a8 7a 87 8d 38 58 9e 56 b8 a6 e2 2e c3 8c 0c 07 90 bb 4a d8 16 0f 22 48 2d 28 32 3c 96 67 ab 56 ed 76 e6 71 7e f0 47 c6 d2 12 2e f6 c5 7e 8f 95 dc 87 6f 33 56 88 26 da 92 41
DACK packet: cc 1e d1 1d 1e
DACK packet sent

5) Receiving data packet....
Data packet: c4 40 bb bf d5 2c c1 76 24 b3 5a c7 e3 01 e1 d1 90 c3 a0 5b 97 38 06 e7 c0 5e d7 10 b9 55 68 50 50 f7 5c fd 8d 10 7a 4a 17 57 71 b6 af 3b 44 85 24 e3 90 c3 2b bc 83 31 68 57 97 44 3e e3 b6 3b 52 5a 3f bb 6b d9 94 8b ea 35 06 36 c8 4d 13 aa 35 d3 79 49 ab 5a d0 af dc 95 92 37 21 19 0c 07 75 39 8f b1 d5 be c4 6f db 26 13 69 51 ee c5 db 78 2f 19 ab c0 27 cf d2 73 05 df 88 97 98 db 73 17 73 17 70 d5 f2 2e a2 f7 10 79 22 b1 08 de a2 94 59 10 d2 f5 b7 27 74 94 4a 3a 89 5a c8 ba 8e cc 96 28 69 9a d2 09 d2 a4 91 5a 5a 14 a2 17 5c a8 c3 93 f0 3b d9 d7 37 3f 51 d8 c7 27 81 5a 95 d9 57 e2 33 f5 a6 c1 76 18 f3 b4 2c e8 cd 00 4b c3 0a c9 57 67 db 13 cc 85 2b 05 9c 3e ca 59 1c b9 cf 48 c2 dd f4 c4 3e df 72 b1 cc 62 b2 eb 36 6b e1 dd e0 af 68 b7 6c 0c 40 b4 14 e5 a5 03 91 9c 90 cf aa eb 5d 5a 60 59 d8 68 0d f7 9c 6c 1d 17 ca 21 19 31 ee c8 b5 8c c8 56 00 e5 ed 49 e3 e3 87 44 30 de 10 22 ce 39 5b 13 be
DACK packet: cc 1f fd 31 1f
DACK packet sent

6) Receiving data packet....
Data packet: 28 7b e7 70 ac e6 c2 a1 cd ef 2b 0b 19 62 1f fa f6 f3 40 02 02 04 cf af 14 18 44 62 9c 6d 76 f8 9c 88 c5 95 c5 8f 26 22 b3 01 75 93 40 03 d1 de 37 7f d2 1c 41 bc fe f0 9b fa 22 f5 96 32 2f b5 2c aa 38 14 de 2c bd 3c b0 c1 69 64 b3 94 d2 df 8d ba 07 7d b6 e2 bf 94 cd 39 39 77 db 1c fc bd c0 d4 d1 7a 67 de de 50 0b c9 5b 17 7b c0 e9 2f 62 4b 81 b2 29 8b 85 fa 43 e8 ec 26 5c 2a f2 e7 68 a4 14 02 22 47 8f 69 d0 a6 2a 72 16 ed cb 0d 87 4c 33 18 f8 9c f3 ff 99 26 f5 3f c2 e8 f0 19 d5 9c cf 04 a8 02 04 3b 4e 54 fb 65 f2 d7 e5 70 b0 16 31 28 96 31 c6 9a f0 5f c4 7f e5 37 39 e6 de f9 0b 26 e4 6b 13 f5 97 b9 b9 2c f7 ff 8b 1c f5 98 f9 54 06 43 d6 8b 6f 96 4d 71 56 dc 29 96 fd 0f f7 c9 96 b3 45 61 03 42 96 2e fa df dd 0f 53 41 c6 c1 64 aa 95 13 bd bf 76 4a d6 e1 0b 79 d0 3c 67 11 d1 46 82 20 ad 80 f1 8c 4b f9 e5 07 29 75 9e 21 5c 6e 9c 81 89 31 ca 5d dd cd 18 ba 98 67 de 19 7d 85 93 a1 f4 e4 67 ac
DACK packet: cc 21 29 e5 21
DACK packet sent

7) Receiving data packet....
Data packet: a6 d9 77 10 be e5 81 58 b6 b0 98 13 41 5e 41 cd c9 87 ad 21 63 bb 76 aa 20 32 17 18 68 ba 9f 6d 8e 38 8c 3c d2 5d d6 f8 e0 ee 27 41 f9 f0 f8 f5 62 28 ac 4b ad 72 c3 61 ef 7a 1d 27 36 c6 d2 af 2e 52 75 25 24 8f 29 ec ec 50 c8 9a d1 c3 70 5e b7 b4 8d b1 86 9b 38 3a 1e 30 e7 a3 9e d8 d3 ac 47 0a 1c ba 1b 5c fa 16 b9 d6 30 1e 8a bf e7 76 8d d6 8b 5f 30 33 77 6b 60 9c c8 62 cb 61 ab 9c c1 ee e9 e8 72 70 9c d1 eb b0 03 b9 58 a8 40 11 0b e0 be 12 7f c1 67 7b 69 03 7d eb 9b 70 a3 a5 25 d8 be 6c ba 97 00 d1 68 15 d2 21 65 98 8e 1e 3e f4 fc 11 5e 97 a1 69 73 3e 43 54 22 55 11 22 21 1a ca 79 d5 63 c1 96 dc 23 16 69 4d 4a c6 92 96 bf 45 17 4a 5d 0b f0 ba 68 41 64 25 13 c1 d7 08 44 29 6d 7c 23 ae 4e 35 da d4 64 a1 33 5e 20 67 c0 bc 78 d5 f1 8e a8 00 9c 01 59 a0 a4 6c e5 4f 71 6e 5c bb 28 4e cd 27 19 da fd 64 6f e7 2c 6a 1f 74 de 25 98 fa 48 0a bd 19 93 41 0c e9 8d 11 2a 70 24 70 5f 16 98 a8 b8 c0 91
DACK packet: cc 22 55 99 22
DACK packet sent

8) Receiving data packet....
Data packet: 73 6d f0 a0 f5 88 6c da 59 2d 51 5e 21 3e 7d a9 56 53 95 0d 6d 79 92 3a 9c 4c 81 de 99 60 fb 8c b9 c8 3e 23 cc 0f 72 19 c1 d6 f8 61 5d 50 33 33 f9 51 a3 09 e9 42 48 ec b7 47 01 ec 4f 29 ae c0 5a 7c 9b fa 48 e4 b7 d6 87 ab ca 16 b4 91 fe 16 07 20 8a 66 d4 53 2d 2a fd d9 42 a6 07 77 2a ed 56 3e 78 44 cb 3a 1d 6a 8e 48 b2 a3 09 44 33 ac 4f 00 1a 8b f9 81 7f 69 75 96 39 74 1c e4 a2 37 6e 99 46 2f 28 f7 46 5e 1e 37 59 c0 d1 27 95 31 a8 12 c5 9c d6 55 a9 ee c8 de 5e f1 bf 5e 1a 11 ab 41 ec 7b 28 a7 12 f9 65 95 2a 09 96 3a 49 32 6a 70 b4 49 1f 2d da 15 c4 93 77 53 73 4f 23 de d8 78 bd 24 09 4d 7e f0 f1 4e f3 f3 75 c5 d0 b6 f3 58 b4 90 39 2d 90 a8 db 55 2f 54 8f 0f ed 5f 03 83 af 6e f9 f9 ce 77 37 e5 20 86 9f bb e7 1b 23 6e c9 24 bf 05 7e 1e a4 55 ab 8d bf af 19 3c 00 2b b5 cd 9f 38 00 c0 10 c8 e0 4b db 68 60 0a e0 00 81 32 01 83 76 4b 59 b4 76 3b 84 0e 16 76 e6 56 77 71 93 47 c6 9d 37 3e a2 f5
DACK packet: cc 23 81 4d 23
DACK packet sent

9) Receiving data packet....
Data packet: 20 7c a2 24 24 62 57 23 11 c2 4c f9 5b b7 17 22 6f f3 20 34 ec 94 fc 7c 8d 46 0b 61 a9 81 63 b8 47 3d dc 75 78 2a a9 54 00 1a 6c 83 c5 29 a1 7b 90 ed d3 c4 29 40 6d 29 bd 76 db 48 ca 0d c6 61 2f a0 a0 75 48 fb b8 19 0a b8 71 cf 2d d2 31 0f 88 14 16 1b 3a d8 ca 16 4b cb 03 88 0e e6 84 81 44 fd ec d3 f1 38 57 07 5f 0f 67 6c d0 78 6a cd 54 88 d3 48 a6 e6 75 aa 49 46 09 ab eb b4 93 ac 34 e3 af ea c4 7a 79 c3 13 7d e3 32 00 54 ab ce 74 3f d0 1b 27 e3 1f 69 ee 0d d9 f9 2d e0 e5 5c 73 1b 78 e0 85 75 ca 2d 9e 20 02 05 a2 36 44 85 6d d3 f2 a3 2c 46 5f de 14 4d 73 28 74 1f e3 a5 b8 be 4f fb ed b5 5c 61 5c 44 0a 09 b1 0e 0c 5b 4a 5d 6c 06 58 6b c3 9a b9 a3 e1 30 b3 62 68 21 7e 59 35 d7 28 2b 0a 63 0d 56 8c 6c b9 b3 f9 e1 17 62 4f d6 01 44 92 8a 7e 27 5b 3e b3 08 8f dc 84 20 e5 6f 58 81 4a 77 56 6f f5 f3 5c 05 98 58 65 cf 67 eb b1 d9 99 42 2b f4 b7 eb 8e 65 b4 f8 4e 35 e4 71 10 4c 8a e9 3f 6e db 58
DACK packet: cc 24 ad 61 24
DACK packet sent

10) Receiving data packet....

Data packet: 49 5b c3 2a 07 1c 27 b3 01 4d c3 8d 80 bd fa fe a3 13 ec 53 e1 34 0c a1 1c 9d 82 d1 85 ae a3 e7 34 6f 0b 0b 44 d6 f8 f1 91 79 ee 82 6a 11 a3 03 2d 24 a8 65 e0 c7 c0 27 19 0f 25 95 ba 25 2c 76 8a 39 6d e5 93 06 4a af 78 89 c9 8d 4b de 15 62 63 c2 c0 e3 31 ff 77 a5 f0 30 a8 d8 a0 a3 21 b8 81 7a ba e4 88 31 49 6b f6 6a 7a 38 8e 58 01 11 33 18 0f 53 9e 7f fd 5b 28 92 1d cb 21 85 a6 6b 62 67 7e ea 3c ad bb 58 7d d9 4a 35 bf cb 9e 8c e3 81 9f 98 af d3 90 bf bd ae 0d 00 8e 0c e0 80 0c b9 c7 ed cd cd 0c a0 34 50 19 94 13 db a8 9f 89 04 7e 3e 36 e0 6a b2 a2 89 f5 13 1b 2a 40 6a 70 0a b6 4a db 21 66 5f 75 cc 05 59 de 51 a9 0f 91 14 2b f0 c4 68 cd 6a 39 17 ed b7 60 d4 0f c4 13 87 e4 f9 0d bf e2 f9 4f 60 04 d2 1a 84 00 ca 3c ac 72 3d 31 a4 ce 64 dc 77 c0 72 da f0 bb 24 8f e9 0b b0 8a 9e 2f b8 04 95 3f 54 e6 3e 48 b9 1b b2 9f 0f 18 e1 52 f5 58 02 06 25 67 cc a6 d5 11 ca 75 d7 81 09 2b f5 8c 9d 74 7d
DACK packet: cc 25 d9 15 25
DACK packet sent

The data transmission rate for the process is: 0.242 Mbps

---

saggarw-DUG3QN:Desktop admin$

Transmitter:

Data transfer statistics:

Average Round Trip Time: 5.9 ms
Minimum Round Trip Time: 2 ms
Maximum Round Trip Time: 13 ms

Receiver:

The data transmission rate for the process is: 0.242 Mbps

4. Scenario when Transmitter and Receiver run on two different machines using ad-hoc wifi: (IP in range of that provided by the hotspot wifi network)

Transmitter:

Data transfer statistics:

Average Round Trip Time: 12.5 ms
Minimum Round Trip Time: 1 ms
Maximum Round Trip Time: 32 ms

Receiver:

The data transmission rate for the process is: 0.013 Mbps

5. Scenario when Transmitter and Receiver run on two different machines using Ethernet (LAN Cables) connect to UMD Secure LAN: (IP in range of the Ethernet subnet)

Transmitter:

Data transfer statistics:

Average Round Trip Time: 3.5 ms
Minimum Round Trip Time: 0 ms
Maximum Round Trip Time: 8 ms

Receiver:

The data transmission rate for the process is: 0.329 Mbps

X. CHART FOR MEASUREMENT STATISTICS

Stats	Case 1	Case 2	Case 3
Average RTT	5.9	12.5	3.5
Minimum RTT	2	1	0
Maximum RTT	13	32	8
Data Transmission Rate	0.242	0.013	0.329

Round Trip Time: RTT

Case 1: Tx Rx running on same machine

Case 2: Tx Rx on different machines connected to ad hoc wifi

Case 3: Tx Rx on different machines connected via Ethernet (LAN cables)