ENTS 640 Networks and Protocols I
Fall 2017
Project

Assigned: November 9; Due: December 4, 4:00pm

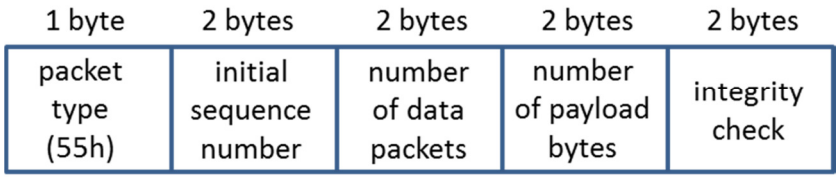
Overview

Write 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 should use 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 services by implementing the data transfer protocol described below. The data transfer should be 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.

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 multi-byte 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.

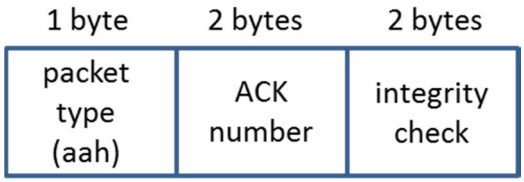
The initial 2-way handshake should consist of the following steps. First, the client/transmitter should send an INIT packet to the server/receiver with the fields shown below:



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 particular 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:

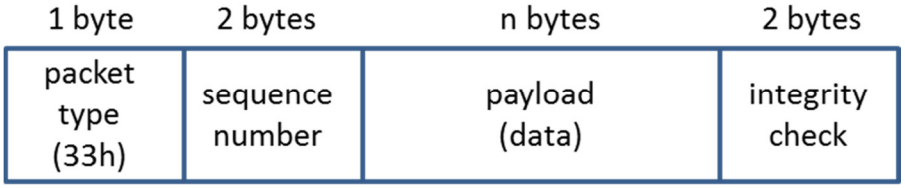


The IACK packet should have the following fields:

- Packet type (1 byte): This field describes the type of the packet, and it should have a value of aah.
- 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).

Since the 2-way handshake is carried out over an unreliable UDP socket, the transmitter and the receiver protocols should take care of making it reliable including the application of timeout timers and retransmissions.

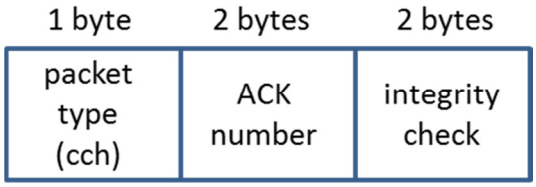
During the second phase, the data transmission phase, the transmitter should send DATA packets to the receiver. The structure of the DATA packets is given as:



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).

Integrity Check Calculation

The integrity check value should be calculated for all packet types as follows. First, a 16-bit variable should be initialized to zero. Then, the whole packet (except for the integrity check field) should be broken up into a sequence of 16-bit words in such a way that the first packet byte in order will be the most significant byte, and the second packet byte in order will be the least significant byte. If the packet consists of an odd number of bytes, the least significant byte of the last 16-bit word should be set to zero. Each 16-bit word should be bitwise exclusive or-ed (XORed) with the content of the variable, and the result should be stored back in the variable. After all 16-bit words have been XORed, the resulting value in the variable will be the integrity check value. At the receiver side, the same process should be repeated, but this time the integrity check field should also be 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.

Protocol Operation

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

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:

A. Initial handshake phase:

1. The transmitter should generate a 16-bit random initial sequence number, and send an INIT packet to the receiver. It should also start 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 should be set to 1 second and should be doubled at each timeout event. After the 4th timeout event, the transmitter should declare communication failure and print an error message.
2. It should wait for an IACK packet from the receiver. An IACK packet can only be accepted if: a) the integrity check passes, b) it has the correct packet type, and c) the ACK number is one larger than the sent initial sequence number.

B. Data transmission phase:

1. For each 300-byte data block, the transmitter should create a DATA packet and send it to the receiver. Starting from the initial sequence number plus one, the sequence number field should be incremented by the length of the current payload for each DATA packet.
2. Each DATA packet should be sent to the receiver using the stop-and-wait protocol. The transmitter should send 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 can only be accepted if a) the integrity check passes, b) it has the correct packet type, and c) the acknowledgement number field is correct. All other received packets should be discarded (ignored).
3. The transmitter should also start 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 should be set to 1 second and should be doubled at each timeout event. After the 4th timeout event, the transmitter should declare 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 should be reset to their initial values.

The receiver protocol should operate according to the following description.

A. Initial handshake phase:
    1. The receiver should receive the INIT packet, and send 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, and b) it has the correct packet type. All other received packets should be discarded (ignored).
    2. The receiver should learn the total number of data packets and the number of payload bytes in each DATA packet from the INIT packet. This way the receiver will know how many DATA packets to expect and the length of the payload in each DATA packet.
    3. It should send 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 should send back an IACK packet to the transmitter each time a correct INIT packet is received.

B. Data transfer phase:
    1. The receiver should receive 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 and 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 should be printed on the screen.

Measurements

The sender protocol should measure the round trip time (RTT) for each acknowledged DATA packet. Before sending each DATA packet, the sender should record the current time using Java's `System.currentTimeMillis()` method. This method returns a `long` value, which is the number of milliseconds elapsed since midnight, January 1, 1970 UTC. After receiving the DACK packet for the DATA packet, the sender should again record the current time and determine the RTT (in milliseconds) by calculating the difference between the two recorded time values. The RTT measurement should only be valid for first-time DACK packets. If timeout occurs and the DATA packet is retransmitted, the RTT measurement process will have to start again (that is, retransmissions should not be included in the RTT measurements). At the end of the data transmission phase, the sender should display the RTT values for all DATA packets, their average, maximum and minimum.

The receiver should measure the overall data rate during the data transmission phase. When receiving the first DATA packet, it should record the current time as described above. When the receiver receives the last DATA packet, it should again record the current time, and calculate the total time it took to receive all DATA packets. At the end of the data transmission phase (after sending the last ACK), it should print 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.

You will need to repeat the following measurement session in three cases: 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; and c) when the sender and the receiver are connected via WiFi connection. For b) and c) cases, you will also need to manually configure the IP addresses of both computers. Please make sure that you place both network interfaces on the same subnet.

For c), you will need to set up an ad-hoc wireless network on your WiFi (wireless) network interfaces between the two computers. On Windows 7, you will need to go to Control Panel/Network and Internet/Network and Sharing Center and select "Set up a new connection or network". Then, select "Set up a wireless as hoc (computer to computer) network". Configure a network name, security protocol and a key if needed (for this experiment, you can just use no authentication as anyone trying to monitor your wireless network can only see random data exchanged over UDP/IP). For other operating systems, please figure out how to set up a wireless ad hoc network between the two communicating computers. Once you have finished the experiments, please compare the obtained RTT and data rate measurement values in your project report, preferably in the form of graphs.

Deliverables and Deadline

Each goup needs to submit the Java source code implementing the above described protocol. Please only send me .java source files, and do not send me .class or any other compiled or executable files. In addition, each group should write a project report (10-15 pages) discussing the problem, their design and solution (flow charts, block diagrams, UML class diagrams, state transition diagrams, etc.), and the application's output, demonstrating the implemented functionality. The ourput should show some cases where the protocol works without any errors, and also some cases where error occurs (e.g. timeout, checksum error, invalid packet type, etc.)

The perfect solution will receive 30 points (30% of the final grade). All materials shuld be submitted electronically as email attachment (to zsafar@umd.edu), one per group, by December 4, 4:00pm in a single email. You are encouraged to send me a single zipped file containing both the Java source code files and the project report.

Late submissions will receive reduced points as follows:

- One day late (before December 5, 4:00pm): 50% reduction – maximum 15 points
- Two days late (before December 6, 4:00pm): 20% reduction – maximum 6 points
- More than two days late: 0 points

Please note that in addition to correctness and functionality, the Java code will also be evaluated for coding style. Thus, you should pay attention to software design/engineering issues: code modularization, class design, code block organization, variable naming, comments, etc.

The last note is on academic integrity: Each group is welcome to discuss the project in general terms with other groups or students. However, when it comes to a particular solution to a particular problem arising during the project work, it must be a result of only the work of the group members. **Copying code from other groups or sharing code with other groups are strictly forbidden and will not be tolerated.**