CPSC 441: Computer Networks
Assignment 3
due March 24, 2017

Important Notes

- This an individual assignment. You should submit your own work.
- The assignment is due 11:59pm on March 24, 2017
- Start the assignment early and avoid procrastination.
- This is a programming assignment in Java

Objective

The objective of the assignment is to practice UDP socket programming and implement a reliable file transfer application protocol which is based on *Selective Repeat* technique. Hence, you can also consider this assignment as an improvement over the previous one by replacing rdt3.0 with a much efficient protocol, Selective Repeat.

Note: The description of this assignment is bit lengthy. This is to help you with the implementation. Please read the description very carefully. Also, terms "segment" and "packet" are used interchangeably.

Overview

You need to implement the client side of a file transfer application protocol which uses UDP as the transport protocol for data transfer. The server side is provided to you, `FastServer.jar`. Since UDP does not provide reliability, this feature has to be incorporated in the application protocol. Specifically, *Selective Repeat* technique will be used to ensure reliable transfer of both binary and text files from client to the server.

The client and server use **TCP connection to exchange control information about the file transfer**. Before the file transfer begins, control messages are exchanged between client and the server over the TCP connection. Then, the client uses a UDP socket to transmit the actual file content to the server. The server uses the same port number for both its TCP and UDP sockets. The client may also use same port number for its TCP and UDP sockets (but, this is not mandatory). Once the file content is fully transfered to the server, client sends end-of-transmission control message to the server over the *same* TCP connection which it has used at the beginning and terminates. Hence, the TCP connection created at the beginning should be kept open until the end. The Figure 1 gives an overview of the data and control message exchanges between the client and server during a file transfer.

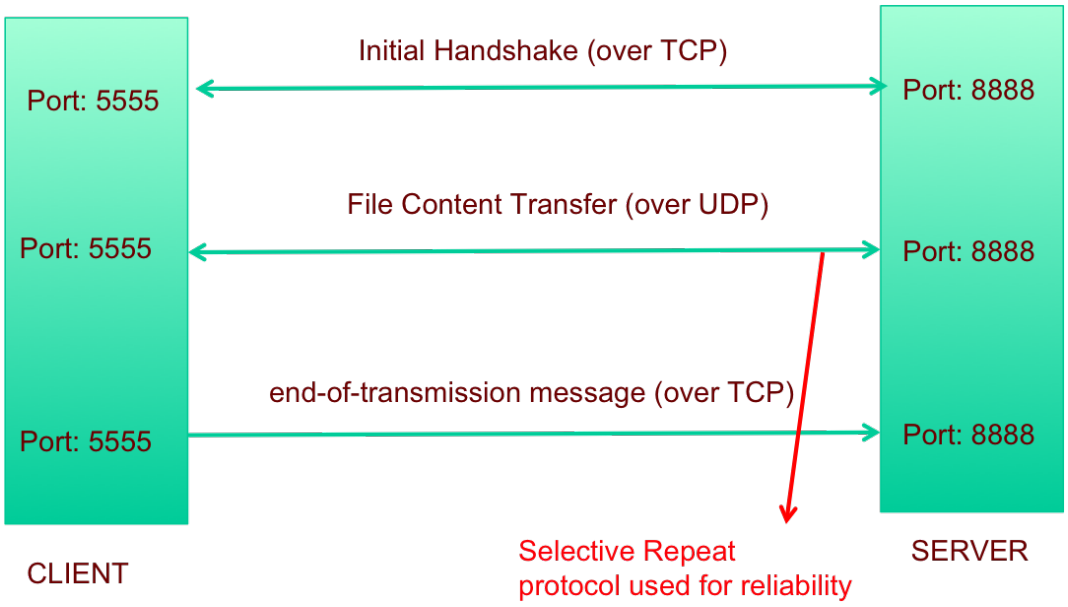

Figure 1: Overview

Protocol Details and Implementation Logic

- Initial handshake: This is done over TCP connection. A TCP connection needs to be established with the server first. All communication over TCP is in binary format. This means, you should use `DataInputStream` and `DataOutputStream` as I/O streams. After TCP connection is established, client sends filename using `writeUTF()` method of `DataOutputStream`. The server will respond back. To read the server response, use `readByte()` of `DataInputStream`. If response is 0, it means server is ready to accept file content. Any other value indicates error and client should terminate immediately.

- Data transfer:

  - **Send data segments**: For sending the data, read the file chunk by chunk, encapsulate each chunk in one segment and send the segment to the server. The maximum size of chunks is given by the constant $MAX\_PAYLOAD\_SIZE$ in class Segment. **The initial sequence number should be** 0 **and sequence number is always incremented after sending each segment**. This means, fist segment has sequence number 0, second has 1, third has 2 and so on. Segment class provided defines the structure of a segment that is transmitted between the sender (i.e., client) and receiver (i.e., server). Read the Javadoc documentation of the class on how to use it. Note that both data packets and ACKs are of type Segment. Segments that go from client (sender) to the server (receiver) carry data, while segments that come from the server (receiver) are ACKs that do not carry any data. In order to simulate packet drop, server drops packets in a ran-

dom fashion based on the drop probability given as run time argument. Unlike in assignment-2 where client waits for each segment to be acknowledged before sending the next segment, since pipelined protocol is used in this assignment, **client sends *window size* number of segments without waiting for acknowledgments**. When a segment is transmitted, it is also added to the transmission queue. Appropriate status may be set for the segment to identify that the segment is only sent and not yet acknowledged. The size of the transmission queue is set as *window size*, which is given as run time argument. If transmission queue is full, then your program has to wait until space becomes available.

- **Receive ACK segments**: Your program should listen for arriving ACK segments from the server. Recall that in the Selective Repeat protocol, if client receives ACK with sequence number n, it means server has received segment with sequence number n. When an ACK is received for a segment within the current sender window, you may consider changing the status of the segment in the transmission queue to reflect that acknowledgement has received (unless it is a duplicate ACK). Also, move forward the current sender window by removing segments from the head of the transmission queue until an unacknowledged segment is found at the head of the queue. If the ACK is for a segment that does not belong to the current sender window, ignore it.
- **Time-outs**: For handling the packet drop, you need to implement a count down timer. Whenever a segment is sent, a count down timer should be set with a time-out value which is 100 ms for this assignment. In Selective Repeat, a timer is set as soon as segment is sent (i.e., a timer for each segment). Upon time-out (i.e when timer goes off), resend the segment and restart the timer for that segment.

  Note: Above three operations, sending segments, receiving acknowledgments and handling retransmissions, should be implemented in parallel. For example, while client is sending segments, it should be able to receive ACKs and handle time-outs simultaneously. Hence, multi-threading is a MUST for this assignment.

- End-of-transmission: Once all segments are sent and corresponding acknowledgments are received, client sends an end-of-transmission message. For this, use `writeByte(0)` method of `DataOutputStream`. After sending this message, client has to terminate itself after the cleanup operation. The cleanup operation includes canceling the timer, closing the sockets and other I/O streams.

Above details are summarized in algorithm-1:

Assumptions

- No error checking needs to be implemented at the application layer (for any bit errors). This will be taken care by underlying protocols including UDP.
- Infinite sequence number space is available (i.e., no wrap around of sequence number)

Algorithm 1 Algorithm FastClient

1. Open TCP connection with the server
2. Initial handshake over TCP connection - send file name and receive response
3. Send data over UDP - as segments, handle acknowledgment and time-out
4. Send end-of-transmission message over TCP connection
5. Clean up - cancel timer, close socket and I/O streams
4. Terminate the program

Program Interface and Source Codes

The skeleton code, `FastClient.java`, is provided to you.

- `public FastClient(String server_name, int server_port, int window, int timeout)`: Constructor for class `FastClient` to do necessary initialization.
- `public void send(String file_name)`: Method that is called by the client to send the file. Once this method returns, the entire file is successfully transfered to the server end.

Also, read comments in the skeleton code. Your task is to complete the `FastClient.java` class. You may introduce additional classes and methods as needed, but must keep the signature of existing methods unchanged.

Class Segment (`Segment.java`) is provided to use in your implementation. Read Javadoc documentation of the class on how to use it. You are not allowed to edit this class.

Your implementation should include appropriate exception handling code to deal with various exceptions that could be thrown in the program. You can run both client and server in the same machine, but in different paths (or folders). **Also, please make sure that window size provided as run time argument at the client and server end are the same**.

Your client should be run as:
`java FastClient <server-ip> <server-port> <file-name> <window-size>`

where, `<server-ip> = localhost` (if you run client and server in the same machine), `<server-port>` is the port at which server is running, `<file-name>` is the name of the file to be transfered and `<window-size>` is the window size.

The file to be transmitted should be in the same path (or folder) where client runs. The server (provided) would save the file transmitted by the client in the path (or folder) where it runs.

Note: For easiness and to save your time, transmission queue implementation is provided, `TxQueue.java` *and* `TxQueueNode.java`*. Read Javadoc documentation of the class on how to use it. The queue is synchronized across its various operations to avoid data integrity problems that may arise when multiple threads are accessing the same queue. You are allowed to edit* `TxQueue.java` *and* `TxQueueNode.java`*. You may also discard this queue implementation and develop your own. If you choose to have your own implementation, please make sure data integrity problems due to multiple thread access on the same queue is taken care of.*

How to run the Server

The server can be run as (on terminal)

```
java -jar FastServer.jar <serverport> <windowsize> <loss>
```

Example:

java -jar FastServer.jar 8888 10 0.1, where 8888 is the port number at which you want the server to listen, window size is 10 and 0.1 is the packet loss probability. You need to test your program with different values for window size and loss to make sure that your implementation is correct. Please see marking guide for more details.

Every possible care is taken to ensure that server is bug free. However, if you encounter any bug, please let the instructor know about it.

Restrictions

- You are not allowed to change the signature of the methods provided in `FastClient.java`. However, you can define additional methods and classes.
- You are not allowed to modify class `Segment`
- Ask the instructor if you are in doubt about any specific Java classes that you want to use in your program