CPSC 441: Computer Networks
Assignment 2
due Feb 17, 2017

Important Notes

- This an individual assignment. You should submit your own work.
- The assignment is due 11:59pm on February 17, 2017 (if you wish to submit the written part as hard copy, you may do so by making use of the physical dropbox in Maths Science building, 1st floor. However, deadline in this case will be 4pm on February 17, 2017)
- Start the assignment early and avoid procrastination.
- Assignment has got two parts, programming and written part. The programming part has to be implemented in Java.

Objective (Programming Part - 85 Marks)

The objective of the programming part is to practice UDP socket programming and implement a file transfer application protocol which is based on alternating-bit stop-and-wait (rdt3.0) technique.

Note: The description of this assignment is bit lengthy. This is to help you with the implementation. Please read the description very carefully.

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, `FTPServer.jar`. Since UDP does not provide reliability, this feature has to be incorporated in the application protocol. Specifically, *alternating-bit stop-and-wait 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 one port number for its TCP and UDP sockets. 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.

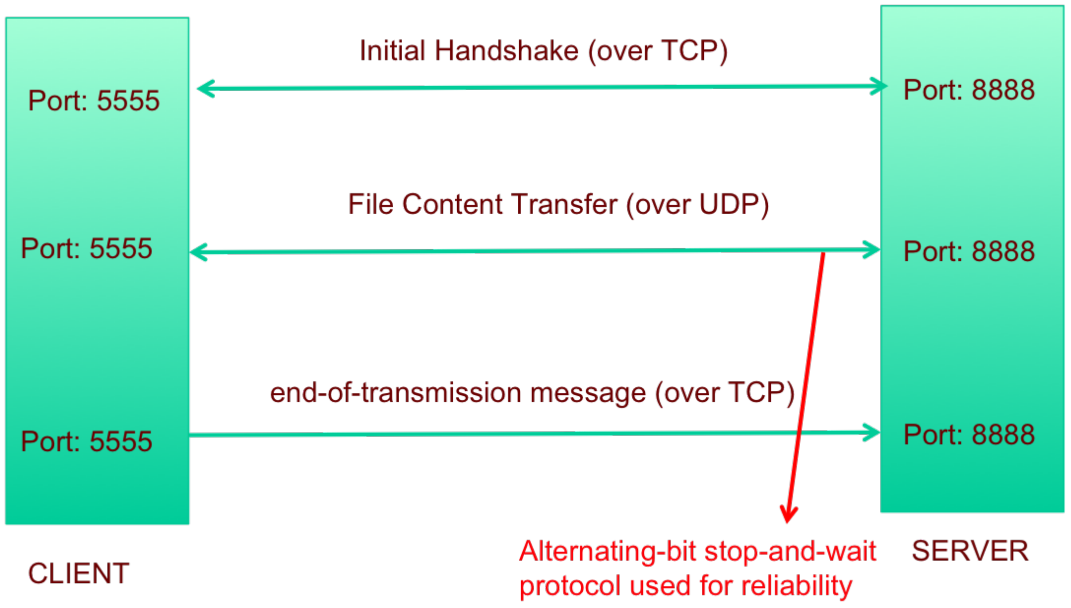
Initial Handshake (over TCP)
Port: 5555
Port: 8888
File Content Transfer (over UDP)
Port: 5555
Port: 8888
end-of-transmission message (over TCP)
Port: 5555
Port: 8888
CLIENT
Alternating-bit stop-and-wait protocol used for reliability
SERVER

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`. 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.

- Data transfer:

  - 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. Since alternating-bit protocol is used, sequence number toggles between 0 and 1. This means, fist segment has sequence number 0, second has 1, third has 0 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 random fashion based on the drop probability given as run argument. The client has to wait for each segment to be acknowledged before sending the next segment, as in stop-and-wait technique. Recall that in the alternating-bit stop-and-wait protocol, for packet with sequence number n correctly received at the server, server sends back acknowledgment with sequence number n. **Server expects sequence number of first segment to be** 0.

  – For handling the packet drop, you need to implement a count down timer. Whenever a packet is send, a count down timer should be set with appropriate time-out value. You can use the `Timer` class defined in java for this. The time-out interval is set as 50 ms for this assignment. Upon time-out (i.e when timer goes off), resend the segment and restart the timer.

  – The client alternate between send and receive operation. Hence, only a single thread of execution is needed to implement both send and receive operation. On receiving an ACK, check whether the ACK sequence number is same as the expected one. If yes, stop the timer, send the next segment and start the timer again. Otherwise, it is a duplicate ACK which should be ignored.

- End-of-transmission: Once the ACK for the final segment is 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 the following algorithm:

Algorithm 1 Algorithm FTPClient

```
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
```

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.

Program Structure and Source Codes

The skeleton code, `FTPClient.java`, is provided to you. Please read comments in the skeleton code.

In addition, class Segment (`Segment.java`) is provided to use in your implementation. Your task is to complete the `FTPClient.java` class. You may introduce additional classes and methods as needed, but must keep the signature of existing methods unchanged.

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).

Your client should be run as:

```
java FTPClient <server-IP> <server-port> <file-name>
```

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

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.

How to run the Server

The server can be run as (on terminal)

```
java -jar FTPServer.jar 8888 0.5
```

Example:

java -jar FTPServer.jar 8888 0.5, where 8888 is the port number at which you want the server to listen and 0.5 is the packet loss probability. You need to test your program with different packet loss probabilities (except 1) to verify that your implementation is correct.

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 to you. 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

Objective (Written Part - 15 Marks)

The objective of the written part is to get familiarized with *Wireshark*, a protocol analyzer, and do a simple packet trace analysis.

Overview

All questions are based on the packet trace (`cpsc441-assignment-2-trace`) provided as part of the assignment. The trace contains HTTP message exchanges between a client and web server along with other packets. Marks for each question is given in square brackets.

Questions

1. How many GET and Conditional GET requests are present in the trace? [2]
2. What is the IP address of HTTP server? On which port is web server application running? [2]
3. What is the HTTP version that is used? [2]
4. List last modified time of object from all Conditional GET request(s) in the trace? [3]
5. List status code and corresponding status message in response to all Conditional GET request(s) present in the trace? [3]
6. Is persistent or non-persistent connection used for exchanging HTTP requests and responses? [3]