Design Notes
Assignment 4

Majid Ghaderi

Disclaimer

These notes are based on my own implementation. I do not claim that my implementation is the simplest or the best. Feel free to use or disregard any of these suggestions as you wish.

General Tips

- It may be helpful to implement a basic UDP client and server on your own before working on this assignment.
- Make sure to test with small files that have just one segment (*e.g.*, file size is 1 byte).
- During testing/debugging, you can change the client and server input parameters, for example, to set the packet loss probability to zero.
- Add a lot of debug messages to your code. You can remove/disable them later when not needed. Do not reinvent the wheel here. Look at the Java debugging facilities specifically the `Logger` class.

Working with `FtpSegment` Class

Client and server exchange segments of type `FtpSegment` with each other: segments containing data are transmitted from the client to the server and in turn segments containing ACKs are transmitted from the server to the client. Each segment has a few header fields and a payload. The maximum size of the payload is given by the constant `FtpSegment.MAX_PAYLOAD_SIZE`. Segments are transmitted using UDP. Each segment is encapsulated in a UDP datagram for transmission. I have defined several helper methods in class `FtpSegment` to facilitate segment encapsulation and de-encapsulation. Feel free to use any of these methods.

1. To create a segment from raw data:

```
// size is the number of bytes in buff
FtpSegment segment = new FtpSegment(seqNum, buff, size);
```

2. To encapsulate a segment in a UDP datagram:

```
// segment is an object of type FtpSegment
DatagramPacket packet = FtpSegment.makePacket(segment, serverIP, serverPort);
```

3. To de-encapsulate a segment from a received UDP datagram:

```
// packet is an object of type DatagramPacket
FtpSegment segment = new FtpSegment(packet);
```

Setting up a Timer

Class `Timer` can be used to schedule a recurring timer. To use the `Timer` class, you need to define a timer task class as well. A timer task is similar to a `Thread` class with a `run()` method. The only difference is that it extends class `TimerTask`. Here is an example of a timer task class:

```
class TimeoutHandler extends TimerTask {

  // define the constructor

  // the run method
  public void run() {
    // process re-transmission of the pending segment
  }
}
```

To start the timer, I used method `Timer.scheduleAtFixedRate()` to create a recurring timer task. In this way, after each retransmission, I do not need to manually restart the timer again. Alternatively, you can just use method `Timer.schedule()` and then re-start the timer manually in `TimerTask.run()`.