Assignment 3: Simplified TCP

Due date: *Friday, December 22, 2017 at 22:00 CEST*

This is an individual assignment. You may discuss it with others, but your code and documentation must be written on your own.

You must implement a simplified variant of the Transmission Control Protocol (TCP), which we call *Simplified Transmission Control Protocol*, or *S-TCP*. S-TCP incorporates the reliability features of TCP plus the congestion-control feature. That is, *S-TCP* provides applications with a *unidirectional, reliable* stream. In practice, you must implement two Java classes, *STCPSender* and *STCPReceiver*, that implement the two sides of the transport layer. Your classes must be integrated with the *transport* library that you can download from the course web page. In fact, *STCPSender* and *STCPReceiver* must extend the *transport.Sender* and *transport.Receiver* base (abstract) classes, respectively. On the course web page you can also find two simple examples showing the use of the *transport* library and how to override its methods.

Details for this assignment are given below. Read them carefully, and make sure that you have understood every requirement before implementing your solution.

S-TCP Specification

S-TCP must allow an application to establish a connection between a sender component and a receiver component. Once the connection is established, S-TCP must reliably transfer a stream of data from the sender to the receiver. Moreover, S-TCP must try to improve the link utilization by increasing the window size.

The sender initiates the connection with a handshake similar to the three-way handshake of TCP. In the case of S-TCP, the handshake must only establish the state of one stream (sender to receiver). Therefore, a simple two-way handshake is sufficient (SYN, SYN-ACK). Similarly, S-TCP must allow the sender to explicitly shut down the connection with a closing handshake (FIN, FIN-ACK).

The basic reliability features of S-TCP are similar to TCP. In particular, the stream transmission is controlled through sequence numbers that identify *bytes* in the stream. Also similar to TCP, acknowledgment numbers indicate the first sequence number not yet seen by the receiver. Also similar to TCP, the SYN or FIN segments are considered to be 1 byte long, and therefore increment the sequence number by 1.

The basic congestion-control features of S-TCP are also the same as TCP. You should adjust the window size based on the congestion observed in the communication link. In particular, the sender should increase the window size by one Maximum Segment Size (MSS) upon receiving an in-order ACK. Consistently, upon receiving a cumulative ACK, the sender should increase the window size corresponding to the number of accumulated ACKs. Moreover, the sender should cut the window size in half or down to $1MSS$ when observing a triple duplicate-ACKs or a timeout expiration, respectively. You do not have to implement other features such as Slow-Start. The upper bound for the sender window size is $W = 64KB$; $MSS = 1KB$ and the initial value for window size is $W = 1MSS$.

S-TCP uses an adaptable timeout for reliability. In particular, S-TCP adapts this timeout by estimating the round-trip time (RTT) and its variability exactly like TCP. Therefore, for every correct acknowledgment A, the sender measures the interval between the time it sent the last segment acknowledged by A and the arrival time of A. The last segment acknowledged by A is the segment with the maximal sequence number lower than the acknowledgment number of A. This interval is the measured round-trip time ($RTT_{measured}$), then using these measurements, the sender maintains an estimate of the round-trip time ($\overline{RTT}$) and of its variability ($\overline{DevRTT}$) using the following equations also used in TCP:

$$\overline{RTT}_{new} = (1-\alpha)\overline{RTT}_{old} + \alpha RTT_{measured}$$
$$\overline{DevRTT}_{new} = (1-\beta)\overline{DevRTT}_{old} + \beta|\overline{RTT}_{old} - RTT_{measured}|$$
$$T = \overline{RTT}_{new} + 4\overline{DevRTT}_{new}$$

where $\alpha = 0.125$, $\beta = 0.25$, and the initial values are connection, $\overline{RTT}_{old} = 1s$, $\overline{DevRTT}_{old} = 0.25s$.

S-TCP then uses the following acknowledgment mechanisms:

- *Delayed ACK* (receiver): upon arrival of an in-order segment with expected sequence number, if all segments up to the expected sequence number have already been acknowledged, wait 500ms for another in-order segment. If that does not arrive, send the ACK.
- *Cumulative ACK* (receiver): upon arrival of an in-order segment with expected sequence number, if another in-order segment is waiting to be acknowledged (case above) immediately send a cumulative ACK for both segments.
- *Duplicate ACK* (receiver): upon arrival of an out-of-order segment with higher-than-expected sequence number (indicating a gap), immediately resend the last sent ACK.
- *Immediate ACK* (receiver): upon arrival of a segment that fills a gap in the received data (partially or completely), immediately send an ACK if the packet starts at the lower end of the gap, so that the sender can update its sliding window.
- *Fast retransmission* (sender): if a given segment is acknowledged three times in a row, the sender immediately sends it again.
- *Timeout reset* (sender): the sender resets its timeout whenever it receives an ACK. When the timeout expires, the sender re-sends all pending packets.

Both sender and receiver must log to the standard output every sent and received packet. The log must include the sequence number and flags (e.g. ACK) of the packet. The sender must also indicate the window size in the log. Below is a sample execution of the sender:

```
SENT seq: 0, payload-len: 0, flag: SYN, window: 1024
RECEIVED ack: 1, flag: SYN-ACK, window: 2048
SENT seq: 1, payload-len: 1000, flag: ACK, window: 2048
SENT seq: 1001, payload-len: 1000, len flag: ACK, window: 2048
RECEIVED ack: 2001 , flag: SYN-ACK, window: 4096
...
```

You may define the format you prefer for S-TCP segments. Since S-TCP handles only unidirectional transmissions, sender-side and receiver-side segments can be different. However, you must *clearly* and *precisely* define them in a separate README file.

Submission Instructions

You must submit a single tar.gz or zip archive containing only three files: *STCPSender.java*, *STCPReceiver .java* and *README.txt*. The *README.txt* file should contain a brief description of your implementation, possibly a list of limitations or errors you are aware of but that you were not able to fix, and clear references to any and all external material you might have used, including discussions with or help from other students. In addition, *README.txt* must contain a *precise* and *clear* description of the used S-TCP segments. Describe clearly how segments are structured, what is the meaning of each field, how many bytes they're long, etc.

Do not include any other file or folder. In particular, you may use the text editor or the IDE of your choice, but do not include project files and folders. However, make sure your solution does not depend on any non-standard library, and that it compiles cleanly with the standard command-line compiler. Name your archive file `assign03-`*`lastname-firstname`*`.tar.gz`.

Submit the tar.gz or zip archive through the iCorsi system.