Computer Networks Practicum

Dimo Stoyanov & Plamen Dimitrov
dsv200 & pdv200
Vrije Universiteit Amsterdam

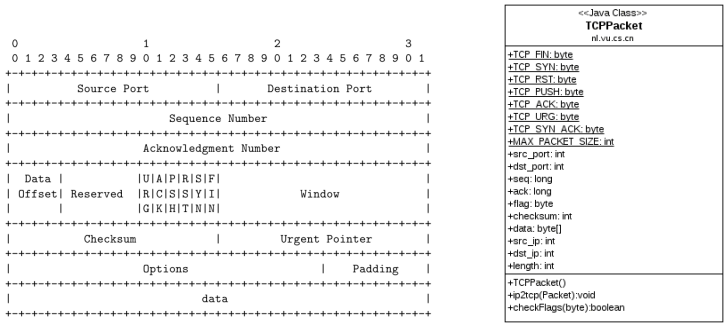
June 30, 2013

1 Building a TCP stack

Our TCP implementation follows strictly the RFC0793 except for the differences explicitly mentioned in the course requirements. A full class diagram of the implementation is

1.1 TCP Packet

The TCPPacket class (See 1b) was used to wrap a TCP packet. As can be seen on Figure 1, the class implementation matches exactly the header. The class also has a method for extracting the relevant information from the payload of an IP Packet.



(a) TCP header (b) TCPPacket class

Figure 1: The TCP header and the class used to implement it.

The types of the TCPPacket's attributes were chosen to adequately represent their ranges and compensate for the lack of the appropriate unsigned types in Java.

1.2 TCPControlBlock class

The TCPControlBlock (see Figure 2) is used to maintain all the information which is needed for the Socket functioning. The state of the socket is stored in the control block. Figure 2 shows all the states in our implementation; we will not discuss them in details since they are used exactly in the way RFC0793 defines them. A socket is uniquely defined based on the control block information.

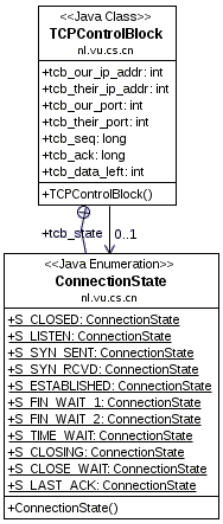


Figure 2: TCPControlBlock class

1.3 Socket class

The actual heavy lifting is done in the Socket(see Figure 3) class. As suggested in the assignment, two methods (e.g. *send_tcp_packet()* and *recv_tcp_packet()*) added to the original ones. They carry out the communication between a socket and the underlying IP layer. The *send_tcp_packet()* encodes a TCPPacket into a byte array, sets the required flags (i.e. sets the PSH flag to on all packets), computes checksums, and transmits the packet. That method can perform both blocking and non-blocking send. Its non-blocking version is latter used for time out detection. The *recv_tcp_packet()* method does exactly the opposite to the *send_tcp_packet()*: it receives a packet from the IP layer, checks if the checksum is correct and passes it to the socket as an instance of the TCPPacket class.

Another helper method implemented in the Socket class is the *add_uints()*, used for addition of unsigned 32-bit integers. These integers are represented as longs (64-bit in Java) and the method prevents the addition/subtraction to cause overflow or underflow of the 32-bit unsigned integer range.

The main methods in the Socket class are *connect(), accept(), read(), write(), close()*. They have one main feature in common - the way they deal with errors. To accomplish that, we implemented the state machine from RFC0793. All the methods require acknowledgement by the other side of every octet they send. Our implementation does not support acknowledgement packets carrying data.

The *connect()* method, as required, is non-blocking and connects a client socket to a server on given *host* and *port*.

2 The Chat Application

A minimalistic chat application was developed on top of the TCP stack described in the previous chapter. It consists of a single window used by both a "client" and a "server". The "server" waits for a connection (e.g. accepts) and the "client" connects to the "server". Because of the lack of multiplexing capabilities of the TCP implementation, the "server" and the "client" maintain two

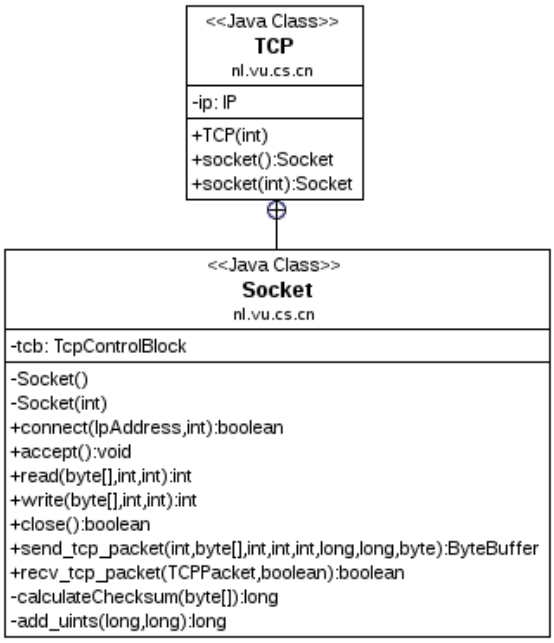


Figure 3: The TCP and Socket classes

TCP stacks each. They use one stack for writing data and one for reading data. The reading is constantly done in separate threads and updates the GUI upon data arrival.