Practical Exercises in Computer Networks

UDP service model check, Datagram loss experiment (WIP)

One of the most important properties of the IP service model represents the fact that the IP network can lose IP packets. The TCP transport protocol adds a multiplexing level to IP, in unicast IP we communicate a host with a host, with TCP we *can* communicate an application running on a host with another application running on another host thereby extending the basic host-to-host communication to a process-to-process model (In fact, in most modern operating systems this is conducive to a thread-to-thread communication). In this case, the promoted TCP-over-IP thread-to-thread model does overcome the packet loss problem and others (Errors, duplication, out-of-order delivery). In this experiment we want to check practically that TCP segments if lost, due to a variety of causes, cause the transmitter to become aware of this negative situation and, normally, retransmit the lost segment.

TCP adds a *namespace* for processes in a host

When a Java application opens a TCP socket associated with a TCP port, a bidirectional association is created between the Java thread and the Socket descriptor within the operating system. From that moment on, incoming segments at that TCP port will be delivered to the TCP socket incoming queue. The Java thread consumes bytes from the incoming queue by invoking methods of Socket. When a TCP connection is created due to an invocation of the Java Socket.connect() method, the involved threads obtain the same level of guarantees they get whne working with file descrpitors and system calls such as open/read/write etc.

What happens if the thread does not consume packets for some arbitrary amount of time? In this case, the TCP protocol will make sure that no information is lost, the transmitter will be notified by the receiver that its transmission rate is too high and it will, accordingly, reduce that rate even to 0, *i.e.,* quit any transmission at all.

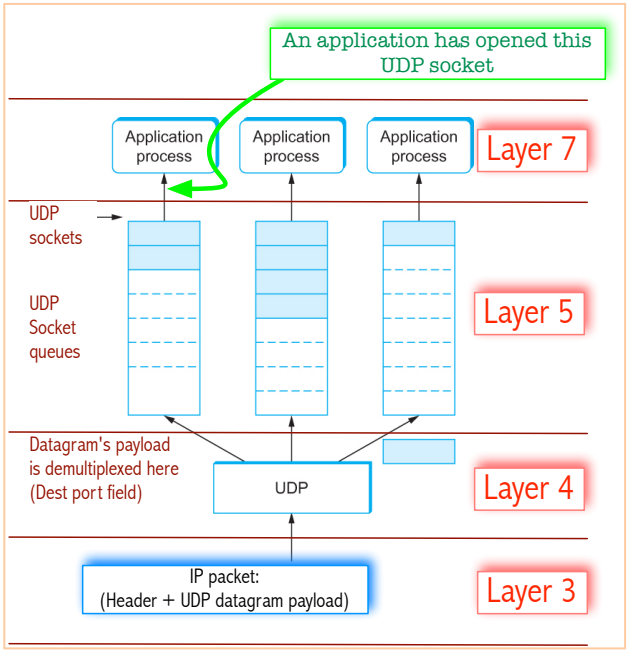

Fig. 1. TCP Socket queues

In this experiment, we will write a Java C/S pair of programs for proving this in a practical way. Unlike in the the previous CN exercise, TCP is a reliable transport protocol and, in this exercise we want to test two outstanding features of the sliding window algorithm in TCP (An advanced form of ARQ that we explained in Ch. 5). You are given the source code of a java Socket server and a counterpart client program, you will have to complete the server only. The client program sends a large amount of information to the server by means of a TCP connection as fast as it can and the server program has to read that information and somewhat process it, your task consists of completing the server program by following the instructions provided within the source code, particularly the loop where each text line of information is read over the socket connection.

Once a text line has been read, a delay is introduced so that the next iteration is effectively delayed, thereby emulating a hypothetical "information processing" function that would prevent the server from fetching (consuming) further lines until this processing finishes, meanwhile, more and more text lines will be delivered over the socket which will ultimately fill up the receive buffer, causing TCP to further close the receive window, eventually down to a size of 0.

You are asked to use Wireshark to check this receive window size reduction. Furthermore, we aim to check what is the next non-zero AWS (Advertised Window Size) size adevrtised after the receiver has a zero size.