EE 450

Solutions to HW 4

Dr. Walker

R3.4)

The choice of the transport protocol depends on the certain goal of that application. As an example consider Voice Over IP (VOIP). If you miss some milliseconds parts of some ones voice in a second you could still understand her. You may even not notice it. In this case reliable packet transfer is not a concern. But delay of a portion of second can be really annoying. TCP ensured the packet delivery in expense of reducing the transmission rate (congestion control) which means adding more delay. This is certainly what you don't want. While UDP does not limit your Transmission rate in expense of unreliable delivery. This is closer to what we want. Some other examples which are delay sensitive are online video conferences.

R3.5)

It is because of the fact that most of end users firewall's block UDP flows since UDP does not care for others and can easily push any network path to saturation. So most of the mentioned application has TCP backed the main UDP one, and in case of UDP blockage they use TCP.

R3.6)

Yes, That is possible. The developer can put a module in application layer which can guarantee reliable data transfer by checking on each packet and ack which has been sent and received. Pushing this guarantee to higher layer (from transport layer to application layer) makes it harder to implement.

R3.7)

Yes, both segments will be directed to the same socket. For each received segment, at the socket interface, the operating system will provide the process with the IP addresses to determine the origins of the individual segments

P3.1)

The client may use a range of port depends on the application which invokes telnet. However, on the sever side the telnet application is always hearing on port 23 and will respond back on the same port.

	source port numbers	destination port numbers
a) A→S	467	23
b) B→S	513	23
c) S→A	23	467
d) S→B	23	513

e) Yes. Two different host may choose the same port number, It is like two different tenants on different houses may have the same mailbox number

f) No. using the same analogy, it is not possible for two different tenant in the same house to have the same mailbox number.

P3.3)

First sum the first two (Wrap around when overflow)

$$\begin{array}{r} 0\ 1\ 0\ 1\ 0\ 0\ 1\ 1 \\ +\ 0\ 1\ 1\ 0\ 0\ 1\ 1\ 0 \\ \hline 1\ 0\ 1\ 1\ 1\ 0\ 0\ 1 \end{array}$$

Then sum the result with the third one:

$$\begin{array}{r} 1\ 0\ 1\ 1\ 1\ 0\ 0\ 1 \\ +\ 0\ 1\ 1\ 1\ 0\ 1\ 0\ 0 \\ \hline 0\ 0\ 1\ 0\ 1\ 1\ 1\ 0 \end{array}$$

Now taking one’s compliment:
One's complement = 1 1 0 1 0 0 0 1.

To detect errors, the receiver adds the four words (the three original words and the checksum). If the sum contains a zero, the receiver knows there has been an error. More detailed example is in the text 3.3.

All one-bit errors will be detected,

But for the case of two-bit errors can be undetected when two of the words have an error at the same bit position (e.g., if the last digit of the first word is converted to a 0 and the last digit of the second word is converted to a 1).

P3.4)

This is very like the previous problem, some simple adding.

a) Adding the two bytes gives 11000001. Taking the one’s complement gives 00111110.

b) Adding the two bytes gives 01000000; the one’s complement gives 10111111.

c) First byte = 01010100; second byte = 01101101.

P3.5)

No, the receiver cannot be absolutely certain that no bit errors have occurred. This is because of the manner in which the checksum for the packet is calculated. If the corresponding bits (that would be added together) of two 16-bit words in the packet were 0 and 1 then even if these get flipped to 1 and 0 respectively, the sum still remains the same. Hence, the 1s complement the receiver calculates will also be the same. This means the checksum will verify even if there was transmission error.

P3.12)

The protocol would work since the corrupted packet would be retransmitted (just like the ack has been lost) and from the receiver end point it would never know which one (lost ack or corrupted packet) has accrued.

To get at the more subtle issue behind this question, one has to allow for premature timeouts to occur. In this case, if each extra copy of the packet is ACKed and each received extra ACK causes another extra copy of the current packet to be sent, the number of times packet n is sent will increase without bound as n approaches infinity.

P3.15)

It takes 12 microseconds (or 0.012 milliseconds) to send a packet, as $1500*8/10^9=12$ microseconds.

In order for the sender to be busy 98 percent of the time, we must have the utilization as:

$$util = 0.98 = (0.012n)/30.012$$

Solving for n the approximate number of packets will be: 2451 packets.