# CMPT 371: Solutions to Homework 2

March 3, 2007

**Problem 3.4:** The sender starts by sending a packet with sequence number 0. The receiver receives it and sends back an ACK. That ACK gets corrupted along the way. The sender will resend the 0 packet, but the receiver has transitioned into the "wait for 1" state, so it returns a NAK for the resent 0 packet. This continues forever.

**Problem 3.8:** Obviously RDT 3.0 will handle this situation, but we should be able to design a simpler protocol because we will know for certain when a packet is lost. Hence our goal will be to make as few changes to RDT 2.1 as possible. Call the states of the sender in RDT 2.1 (figure 3.11) $S_0, S_1, S_2, S_3$, going clockwise from the start state. Let $RTT_{max}$ be the maximum delay; that is, the longest amount of time that can elapse between when a packet is sent by the sender and an ACK or NAK is received in return. We will set the time to the value $RTT_{max}$. On the transition from $S_0$ to $S_1$ and from $S_2$ to $S_3$, start the timer. Add a transition from $S_1$ to itself and from $S_3$ to itself with the label $\frac{timeout}{udt\_send(sndpkt)}$. What's the difference between this and 3.0? We don't have to worry about receiving an ACK/NAK in states $S_0$ and $S_2$.

**Problem 3.13:** The protocol will be based on GoBackN in that the sender will have a window of size N and when it discovers a loss it will resend everything starting with that lost packet. The timer will be on the receiver side now and the receiver's default behavior will be to periodically send NAKs unless it receives a packet. The sequence number of the NAK is meant to indicate the next packet it is expecting.

More precisely, the sender will start sending and will keep going until it has sent N packets, as long as there is a supply of packets and it hasn't received a NAK. If it receives a NAK i for some i in the past, it will start resending everything starting with packet i. After N packets, it will stop and wait for NAK i where i-1 is the last packet sent. If it gets such a NAK it will advance the sending window by one and send packet i (when there is one).

In general, the receiver is expecting the next packet, say sequence number i. It starts a timer and waits. If it receives packet i, it restarts the timer expecting packet i+1. If it doesn't receive packet i and there is a timeout, it sends NAK i and restarts the timer. If it receives a previous packet, it does nothing. If it receives a future packet, it sends NAK i and restarts the timer.

When the sender doesn't have much to send, the receiver might not discover a lost packet until its timer times out. When the sender has a lot to send, the receiver will discover a lost packet more quickly because it will receive a future packet (i.e. a packet after the one it is expecting).

**Problem 3.18:** The sequence numbers go from 0 to $k-1$. If $N \leq m = \lfloor k/2 \rfloor$, then the sender cannot send a second pkt 0 without getting an ACK for pkt $m$. Otherwise, it can.

**Problem 3.20:**

(a) Recall that TCP sequence numbers refer to the number of bytes in the bytestream that have already been sent, not to the number of packets. Therefore, if $L > 2^{32}$, or 4 GB, then TCP will need to rollover its sequence numbers.

(b) The number of segments will be $2^{32}/1460$. Each segment will have size $1460 + 66 = 1526$ bytes or $1526 \times 8$ bits. Therefore, the total time will be

$$\frac{2^{32}}{1460}\frac{1526 \times 8}{10 \times 2^{20}} \approx 3,422 \text{ seconds.}$$

If you used $10^6$ instead of $2^{20}$, that's fine.

**Problem 3.21:**

(a) Assume we initialize EstimatedRTT with 0. Then, after the four samples,

$$\begin{aligned}\text{EstimatedRTT} &= (0.9)^3(0.1)\text{SampleRTT}_4 + (0.9)^2(0.1)\text{SampleRTT}_3 \\ &\quad +(0.9)(0.1)\text{SampleRTT}_2 + (0.1)\text{SampleRTT}_1.\end{aligned}$$

(b)

$$\text{EstimatedRTT} = \sum_{i=1}^{n}(0.9)^{i-1}(0.1)\text{SampleRTT}_i.$$

(c) The influence of a given sample decreases exponentially in the number of samples we have obtained since then.

**Problem 2.13:**

(a) A whois database contains the names and IP addresses of various hosts within a given domain. In particular, it lists the authoritative name servers of a domain. The various whois databases are maintained by the registrars of domain names.

(b) See attached transcript.

(c) See attached transcript. We use the dig command instead of nslookup because it is more thorough and up-to-date.

(d) See attached transcript. `<www.sfu.ca>` does not have multiple IP addresses, but `<www.google.com>` does.

(e) See attached transcript.

(f) An attacker could use a whois database to identify the authoritative name servers of an institution. The attacker can then use nslookup to identify the actual names of web and mail servers, etc. He or she can then, for instance, try to overwhelm them with mass logins.

(g) While whois databases can be used for attack or fraud, they can also be used for authentication. For instance, they might allow one to verify that a given webpage or email is really from the institution it claims to be from.