**ECE 465/565 Computer Networks and Protocols**
**Fall 05**

Dr. Thinh Nguyen

**Solutions**
**Homework 2**

**Problem 11.**

In a NAK only protocol, the loss of packet *x* is only detected by the receiver when packet *x+1* is received. That is, the receivers receives *x-1* and then *x+1,* only when *x+1 is* received does the receiver realize that *x* was missed.

If there is a long delay between the transmission of *x* and the transmission of *x+1,* then it will be a long time until *x* can be recovered, under a NAK only protocol. Therefore, it is preferable to use a protocol that uses ACK's.

On the other hand, if data is being sent often, then recovery under a NAK-only scheme could happen quickly. Moreover, if errors are infrequent, then NAKs are only occasionally sent (when needed), and ACK are never sent. Therefore, the NAK-only protocol will significantly reduce feedback compared to the ACK-only protocol.

**Problem 12.**

It takes 8 microseconds (or 0.008 milliseconds) to send a packet.

In order for the sender to be busy 90 percent of the time, we must have

$$util = 0.9 = (0.008n)/30.008$$

or $n$ approximately equal to 3376 packets.

**Problem 18.**

In order to avoid the scenario of Figure 3.26, we want to avoid having the leading edge of the receiver's window (i.e., the one with the "highest" sequence number) wrap around in the sequence number space and overlap with the trailing edge (the one with the "lowest" sequence number in the sender's window). That is, the sequence number space must be large enough to fit the entire receiver window and the entire sender window without this overlap condition. So - we need to determine how large a range of sequence numbers can be covered at any given time by the receiver and sender windows (see Problem 13).

Suppose that the lowest-sequence number that the receiver is waiting for is packet m. In this case, it's window is [m,m+w-1] and it has received (and ACKed) packet m-1 and the

w-1 packets before that, where w is the size of the window. If none of those w ACKs have been yet received by the sender, then ACK messages with values of [m-w,m-1] may still be propagating back. If no ACKs with these ACK numbers have been received by the sender, then the sender's window would be [m-w,m-1].

Thus, the lower edge of the sender's window is m-w, and the leading edge of the receivers window is m+w-1. In order for the leading edge of the receiver's window to not overlap with the trailing edge of the sender's window, the sequence number space must thus be big enough to accommodate 2w sequence numbers. That is, the sequence number space must be at least twice as large as the window size, $k \geq 2w$.

**Problem 31**

**a)** The loss rate, *L*, is the ratio of the number of packets lost over the number of packets sent. In a cycle, 1 packet is lost. The number of packets sent in a cycle is

$$\frac{W}{2} + \left(\frac{W}{2} + 1\right) + \cdots + W = \sum_{n=0}^{W/2} \left(\frac{W}{2} + n\right)$$

$$= \left(\frac{W}{2} + 1\right)\frac{W}{2} + \sum_{n=0}^{W/2} n$$

$$= \left(\frac{W}{2} + 1\right)\frac{W}{2} + \frac{W/2(W/2+1)}{2}$$

$$= \frac{W^2}{4} + \frac{W}{2} + \frac{W^2}{8} + \frac{W}{4}$$

$$= \frac{3}{8}W^2 + \frac{3}{4}W$$

Thus the loss rate is

$$L = \frac{1}{\frac{3}{8}W^2 + \frac{3}{4}W}$$

**b)** For $W$ large, $\frac{3}{8}W^2 >> \frac{3}{4}W$. Thus $L \approx 8/3W^2$ or $W \approx \sqrt{\frac{8}{3L}}$. From the text, we therefore have

$$\text{average throughput} = \frac{3}{4}\sqrt{\frac{8}{3L}} \cdot \frac{MSS}{RTT}$$

$$= \frac{1.22 \cdot MSS}{RTT \cdot \sqrt{L}}$$

Programming Assignment

```
/*
 * Created on 11/ 05/ 2004
 *
 */
import java.io.BufferedReader;
import java.io.DataOutputStream;
import java.io.InputStreamReader;
import java.net.Socket;
import java.util.StringTokenizer;
import java.io.InterruptedIOException;

/**
 * @author Thinh Nguyen
 *
 * RDT Sender
 *
 */
public class RdtSender {

	final static String CRLF = "\r\n";

	public static void main(String[] argv) throws Exception{
		// Get the server name and port number from the command
line.
		String server = argv[0];
		int port = (new Integer(argv[1])).intValue();

		// Connect to the server
		Socket socket = new Socket(server, port);
		socket.setSoTimeout(1000);
		DataOutputStream toServer = new
DataOutputStream(socket.getOutputStream());
		BufferedReader fromServer = new BufferedReader(new
InputStreamReader(socket.getInputStream()));

		int seqno = 0;


		while (true)
		{
			try
			{

				// send packet
				String packet = seqno + "\n";
```

```
                        toServer.writeBytes(packet);
                        // wait for ACK
                        String ACK = fromServer.readLine();

                        StringTokenizer tokens = new
StringTokenizer(ACK);
//                      String sequence = tokens.nextToken();
                        String flag = tokens.nextToken();

                        System.out.println("packet: " + seqno + "
flag:" + flag);

                        if (flag.equals("1"))
                        {
                              seqno = seqno + 1;
                              System.out.println("ACK");
                        }
                        else
                        {
                              System.out.println("NACK");
                        }

                        if (seqno == 10)
                        {
                              break;
                        }
                  }

                  catch (InterruptedIOException iioe)
                  {
                        System.out.println ("Timeout occurred");

                  }

            }



            // Close the connection and exit
            socket.close();
      }
}
```