Final Programming Assignment-Module 2
(Module ‘2’ Due Thursday, 4th Nov’04)

In this module you will be required to implement some reliability mechanisms on top of the (unreliable) STFTP aware client and server you have already implemented in module1. You will also make this protocol more efficient by implementing a form of the sliding window in this protocol. Details of this are given below.

**What needs to be done?**

When ever a data packet is sent from either client to server or vice versa, the receiver should reply with an ACK packet. After sending a data packet, a timer should be started so that if the ACK is not received for a given period of time, then the packet needs to be re-transmitted. If an ACK is received, then it is sufficient for all the packets before and including this ACK number. For example if the sliding window is awaiting ACK 5 and it receives ACK 7, then the window moves by 3 and now the next awaited ACK is 8.

On the other hand, the receiver should keep track of all the packets arriving, and if there is a duplicate packet within the receive window then it needs to simply discard the second copy.

In this module, you will implement simple selective repeat protocol over STFTP (module1). The window size for both, sender window and receiver window, is 4 and the sequence numbers of packets would range from 0 to 7. The **sender** will start sending packets from the start of its window, and it will not slide the window, until it has received the ACK for the first packet in the window. As soon as the sender receives and ACK it will slide the window. For example if the sender window starts from 3, then on receiving ACK4, the window will slide by one (implying that the first packet was received correctly). Sender will transmit new packets until it has exhausted the window or one of the previously sent packets time out, in which case you will retransmit them. If the sender receives a NACK, it will retransmit the required packet and will continue its transmission from where it was before. For example if the next packet in line for transmission was packet number 7, but it received a NACK for 4, so now it will transmit packet number 4 and then continue transmission form 7 onwards (remember it is selective repeat).

**Receiver**’s window is also of size 4. When the receiver receives a packet for the first location in the window, it will send an ACK with the next expected packet, and will also slide the window. However if the packet received is not the first expected packet (not the first location in the window), but for any other location in the window, then it will keep the packet, but will send a NACK for the first packet expected (note: it is not sending an ACK in this case). For example, if the receiver was expecting packet number 3 (start of window), but it gets packet number 5, then it will keep this packet but will send a NACK for 3. In this case it will not slide the window until it gets the first expected packet. So now we know that the receiver responds with either an ACK or a NACK when it receives a data packet for the first time, and it simply ignores the duplicate packets when ever they arrive (if they do).
In module one, there was a field for block#, and this field will now be used for sequence numbers. In module one, the 2 byte block# could have represented anything from 0 to 65535 (2^16bits), but from now onwards, they will only range from 0 to 7 and then wrap around. You will use this field as the sequence number for sliding window, as described above.

**ACK and NACK packet**

<table>
<thead>
<tr>
<th>SOURCE IP</th>
<th>0</th>
<th>DEST IP</th>
<th>0</th>
<th>SOURCE Port</th>
<th>0</th>
<th>DEST Port</th>
<th>0</th>
<th>OPCODE</th>
<th>0</th>
<th>BLOCK #</th>
</tr>
</thead>
</table>

As you can see above, there is a field for OPCODE in the acknowledgment packet. You will use this field to determine whether this packet is an ACK or a NACK. For ACK, the OPCODE will be ‘04’ and for NACK it will be ‘14’.

To implement sliding window, you will need to use timers. We will provide a pseudo code for how an iterative timer works and how you can use it in your assignment. If anyone is able to implement multi-threaded timers, then he/she will be given extra credit, however we caution you that it won’t be easy to do so.

Another piece of advice is that take a good look at the selective repeat protocol and then go through the details given above before you start coding, because previously students have misunderstood the problem statement and implemented something different.

That’s it for today; you can expect another instructive email soon.

THE HELPFUL TAs (IAQ, NA, QK)