```
/**
 * A go-back n type sliding window protocol
 */

public class GoBackN
    extends datalink.Protocol
{
    int nextBufferToSend;        // buffer to be sent when channel is idle
    int firstFreeBufferIndex;    // buffer in which to store next packet
    int nextSequenceNumberExpected;  // sequence number expected
    int firstUnAcknowledged;     // last unacknowledged frame
    final int maximumSequenceNumber;
    int numberOfPacketsStored;
    final int windowSize;

    datalink.Packet[] buffer;
    double timer;

    public GoBackN(int windowSize, double timer)
    {
        super( windowSize, timer);
        numberOfPacketsStored = 0;
        nextBufferToSend = 0;
        firstFreeBufferIndex= 0;
        nextSequenceNumberExpected = 0;
        firstUnAcknowledged = 0;
        maximumSequenceNumber = windowSize;
        this.windowSize = windowSize;
        this.timer = timer;
        buffer = new datalink.Packet[windowSize+1];
    }

    public void FrameArrival( Object frame)
    {
        DataFrame f = (DataFrame) frame;
        // a frame has arrived from the physical layer

        // check that it is the one that is expected
        if (f.sequenceNumber == nextSequenceNumberExpected)
            {
                sendPacket(f.info); // valid frame, so send it
                // to the network layer
                nextSequenceNumberExpected = inc( nextSequenceNumberExpected);
            }
        // if frame n is ACKed then that implies n-1, n-2 etc
        //  have also been
        // ACKed, so stop associated timers.
        while ( between( firstUnAcknowledged,
                         f.acknowledgment,
                         nextBufferToSend) )
            {
                numberOfPacketsStored--;
                stopTimer(firstUnAcknowledged);
                firstUnAcknowledged = inc( firstUnAcknowledged);
            }
        if ( numberOfPacketsStored < windowSize )
            enableNetworkLayer();
    }

    public void PacketArrival( datalink.Packet p)
    {
        buffer[firstFreeBufferIndex] = p;
        numberOfPacketsStored++;                // buffer packet
        if ( numberOfPacketsStored >= windowSize )
```

```
            disableNetworkLayer();

        if ( isChannelIdle() )
            {
                transmit_frame( nextBufferToSend);
                nextBufferToSend = inc( nextBufferToSend);
            }
        firstFreeBufferIndex = inc( firstFreeBufferIndex);
    }

    public void TimeOut( int code)
    {
        int seq = inc(code);

        while ( seq != nextBufferToSend )
            { // cancel outstanding timers
                stopTimer(seq);
                seq = inc( seq);
            }

        nextBufferToSend = firstUnAcknowledged;
        if ( isChannelIdle() )
            {
                transmit_frame( nextBufferToSend);
                nextBufferToSend = inc( nextBufferToSend);
            }
    }

    public void CheckSumError()
    {
    }

    public void ChannelIdle()
    {
        if ( nextBufferToSend != firstFreeBufferIndex )
            {
                transmit_frame( nextBufferToSend);
                nextBufferToSend = inc( nextBufferToSend);
            }
    }

    private boolean between( int a, int b, int c)
    {  // calculate if a<=b<c circularly
        if(((a<=b) && (b<c))
           || ((c<a) && (a<=b))
           || ((b<c) && (c<a)))
            return true;
        else
            return false;
    }

    private int inc ( int a)
    {  // increment modulo maximum_sequence_number + 1
        a++;
        a %= maximumSequenceNumber+1;
        return a;
    }

    private void transmit_frame( int sequenceNumber)
    {
        int acknowledgement;
        // piggyback acknowledge of last frame receieved
        acknowledgement = (nextSequenceNumberExpected+maximumSequenceNumber)
            % (maximumSequenceNumber+1);
        // send it to physical layer
```

```
        sendFrame( new DataFrame( sequenceNumber,
                                  acknowledgement,
                                  buffer[sequenceNumber]));
        startTimer( sequenceNumber, timer);
    }
}

class DataFrame
{ /* frame structure */
    int sequenceNumber;
    int acknowledgment;
    datalink.Packet info;

    DataFrame ( int s, int a, datalink.Packet p)
    {
        info = p;
        sequenceNumber = s;
        acknowledgment = a;
    }

}
```