Matthew Garmon
Walter Conway

A look at a Brief Report of a System Architecture
System Architecture Discussed: Selective Repeat

Description:
The system architecture that was implemented was the Selective Repeat(SR) architecture so there could be reliable transport of information through the network. This architecture is what is called a sliding-window protocol. The SR architecture attempts to re-transmit only those packets that are actually lost that are due to errors.

Design:
The fragment header we decided on was 18 bytes of the 128 which left us 110 bytes to send data reliably. Inside the header we place a check-sum section, sequence number, end of fragment indicator.
The check-sum we used to verify the fragment was transmitted correctly was MD5, the digest size was 128 bits,which is 16 bytes. This was place in the first section of our header. We choose to include the sequence number and the last fragment indicator and lastly the data that that was in the last 110 bytes to generate a MD5 hash for verification.
The sequence number was used to determine which array element we should place the incoming fragment in if it was acceptable. We choose not to use an int primitive since all we needed was at least 32 significant symbols to determine the placement of the incoming fragment. Since an int primitive would have taken up was 32 bits, which would have been 4 bytes of data, which would have been too much. Since byte in java is has a minimum value of -128 and a maximum value of 127 (inclusive). This fit our needs. Since byte was the lowest primitive that we could use with ease we decided to use it as well for the last fragment indication flag.

The timing data structure that we used was based off the Article that was being referenced in the book that was corresponding to how we should use one hardware clock and other additional software timers. This article was titled “Hashed and Hierarchical Timing Wheels: Efficient Data Structures for Implementing A Timer Facility”. To incorporate this timing mechanism into our architecture we had an array list that held the time and sequence number that would expire in ms if the fragment was lost. We used a long running thread to decrement each member of the list. If one of these times reached was zero it would then send the single fragment back to the client. If the packet was received correctly then it would stop the timer by deleting it from the list. Once a Fragment was sent it created a new timer for that sent fragment and store it into the list in the appropriate place.

The gremlin takes in the datagram that was received from the socket and then randomly generates a double. The formula used to determine the range of each case to either loose, damage, or corrupt the inputted datagram was loose Packet = 1-(X+Y), where X is the probability of corruption and Y is the probability of losing the fragment from this we base the ranges off the loose packet result
so if the random generated double was from (0 – (1-(X+Y))) it would pass, if the double was from [(1-(X+Y)),(1-(X+Y)) + X) it would corrupt, [(1-(X+Y))+X, 1) it would lose the packet.
If the double that was randomly generated falls into the range of corruption we have to randomly generate another double to decide by how much should the gremlin corrupt the data in the datagram. So from 0 to .5 was one bit .5 to .5+.3 two bit from .8 to 1 was three bits. When it chooses to corrupt the datagram it extracts the data from the datagram and randomly selects a byte from 128 then from there randomly chooses one bit and XOR that position with a 1 which then flips the bit.

Implementation issues:
A few issues we came across while implementing this architecture was not closing the streams after we opened them, which caused a lot of toiling over why the program only worked when we used System.out.print… which it would flush…Another was the timer data structure which required us to have to do some digging into the textbook to locate their source. The implementation we used for the timing data structure is not the most efficient, but it was the easiest to write as well since there are only 4 timers so efficiency wasn’t the major concern. Also, only allowing packets in the bounds was another issue using mod arithmetic.