CA3 Report:

A(A, and B represent two entities sending packets) creates a Packet object is created with a SEQ(sequence number) (Indicating how many packets have been sent, an ACK(acknowledgment number) (Indicating the last packet successfully received), the data from the message.

A checksum is then created from these parts of the packet. The SEQ and ACK are then added to together, and then the data is then converted into int from Java's hashCode method and added to them. How Java's hashCode works is that it takes the array of char's, takes the value of the char (When char's are in expressions they provide their ASCII equivalent) adds it to the previous hash (starting at 0), then multiplies it by 31[1]. This allows B to check whether the packet has been corrupted. Since the checksum stored on the packet is an int(32 bit) there is the chance that a corrupted packet's stored checksum, and the checksum generated from the packet can be the same, or that two seperate packets have different data, and pass the check[2]. The chances however are very small.

When B receives a packet and it hasn't been corrupted. B sends a small ACK packet to tell A that it has received this packet, and ready to take the next packet. If the packet is corrupted however, a NACK(negative acknowledgment) to tell A that B needs it to send that packet again. A NACK is created in the same way as an ACK, but with a different sequence number so that A knows the packet it sent has been corrupted.

A does the same checks as B to make sure the ACK packets haven't been corrupted. It also checks if the packet it is been acknowledged for is the same as it sent. If either of those checks fail the previous packet is sent again.

Once a packet is sent a timer of 1000 milliseconds is started. If an ACK hasn't been received since that time it is resent.

1:
http://docs.oracle.com/javase/7/docs/api/java/lang/String.html#hashCode%28%29
2: http://en.wikipedia.org/wiki/Pigeonhole_principle