Main topics: The lecture included a formal proof for correctness of ARQ stop and wait, an overview as well as proof of Go-Back-n (GBN) ARQ, and an overview of Selective Repeat ARQ. This corresponds to sections 2.4.1, 2.4.2, and 2.4.3 of our text.

I. ARQ Stop-and-wait.

Summary of algorithms for reliable transmission:

At the sending end, A:

1. Transmit packet SN
2. If packet RN such that RN>SN is received, then set SN=RN
3. If a timeout occurs, go to step 1

At the receiving end, B:

1. If packet SN such that SN=RN is received without errors, then increment RN, RN=RN+1
2. Occasionally (i.e. within a finite delay), acknowledge receipt of the packet by sending an ACK along with RN.

Proof of correctness of Stop-and-wait ARQ

The proof of algorithms of this type consists of two parts:

a. The algorithm is safe: this means that the algorithm never produces an incorrect result.

b. The algorithm is live: which means that the algorithm can never enter a deadlock condition from which no further progress is possible.

For stop-and-wait ARQ, the above conditions translate into the following:

a. It is safe if B releases packets in order to the higher layer.

b. It is live if A can forever accept packets and release them at B.

The first proof is easy in our case. Suppose RN=i, this means that packets 0, 1…i-1 have been released and the next packet accepted will be packet i. That is, packet i is the only packet that it can next accept and release.

The second proof is trickier and we will present the proof from the book as well as an alternate view. First, let us think of sequence numbering for a moment, for stop-and-wait we can number packets using a modulo 2 sequence. Why? We will illustrate this with what follows.
Let RN(t) be the value of the variable RN at node B and SN(t) be the corresponding value of SN at node A. At time t, RN(t)=i+1. This means that all packets 0,…,i have been received, so SN is at least i. Moreover, assuming that B already ACKed the packet i, SN will be at most i+1. Therefore, SN(t) ≤ i+1. In other words, SN(t) can never be greater than or equal to i+2 because B never ACKed i+2. Thus, either SN=i, or SN=i+1. Therefore, we can map SN/RN = i with SN/RN=0 and SN/RN = i+1 with SN/RN=1 and we can construct the following state machine:

![Finite State Machine for Stop-and-wait ARQ](image)

This state machines, then, illustrates that the stop-and-wait ARQ algorithm never reaches a deadlock state.

An alternative way of illustrating the liveness of this algorithm is the following. A packet sent is received error-free with a nonzero probability q. This implies that A sends a packet SN with finite delay (if this were not the case, then q would be zero). Also, an acknowledgement will reach A from B with finite delay.

Notice, however, the inefficiency of this algorithm form A’s point of view. A could be transmitting useful packets while waiting for the acknowledgements from B. This technique is the subject of our next discussion.

II. Go Back n (GBN) ARQ

The difference between this technique and the previous lies in the algorithm for the sender, A. In GBN ARQ, A can race ahead n-1 packets without receiving an ACK from B. However, A cannot send packet i+n without packet i being ACKed first. We can think of a window being formed in this ARQ protocol, its length is n and it moves accordingly with the ACKs being receives. The situation is illustrated in the following figure:
In general, the range of the window at any given time is \([\text{SNmin}, \text{SNmin} + n]\). Or, as a function as \(\text{SNmin}\) and \(\text{SNmax}\), \([\text{SNmin}, \text{SNmax} - 1]\) where \(\text{SNmin}\) is the lowest packet not yet acknowledged from B and \(\text{SNmax}\) is the next packet from the higher layer.

The rules for the algorithm at the sending A are the following:

1. Set \(\text{SNmin} = \text{SNmax} = 0\)
2. Do steps 3, 4, 5 repeatedly in any order
3. If \(\text{SNmax} < \text{SNmin} + n\) then accept packet from higher layer. Increment \(\text{SNmax}\) (\(\text{SNmax} = \text{SNmax} + 1\))
4. If packet is received error-free from B with \(\text{RN} > \text{SNmin}\) then increase \(\text{SNmin}\) to \(\text{RN}\)
5. If \(\text{SNmin} < \text{SNmax}\) and no frame is being sent then send frame \(\text{SN}\) for some \(\text{SN}\) s.t. \(\text{SNmin} \leq \text{SN} < \text{SNmax}\). A bounded delay must exist between the sending of the successive \(\text{SNmin}\) packets.

The rules for the receiving side, B, are identical from that of the stop-and-wait ARQ.

Proof of correctness of GBN ARQ.

a. Safety: The safety property depends on B, thus the proof is the same from the one provided in stop-and-wait ARQ.
b. The liveness property lies on the fact that at some point in time, $SN_{\text{min}} = RN$ and eventually B will receive the packet $SN_{\text{min}}$.

Proof of liveness:

Let $RN = i$.
At time $t_1$, A receives $RN = i$ and sets $SN_{\text{min}} = RN = i$.
At time $t_2$, B receives $SN = i$ without errors and sets $RN = i+1$.
At time $t_3$, A receives $RN$ which is greater than $SN_{\text{min}}$. It then sets $SN_{\text{min}} = RN$.
If we assume, as we did in the stop-and-wait ARQ, that each packet is received error-free with a nonzero probability $q>0$, then we know that the time intervals between $t_1$, $t_2$, and $t_3$ are finite. This is sufficient to demonstrate liveness, since using this argument for each successive value of $SN_{\text{min}}$ shows that each packet is transmitted with finite delay.

Selective repeat ARQ.

The main feature of this technique is that the sender avoids unnecessary retransmission of good packets, as opposed to GBN protocols where the sender must transmit at least one round-trip-delay worth of frames when a single error occurs in an awaited packet.

The basic idea for a selective-repeat ARQ for data on a link from A to B is to accept out-of-order packets and to request retransmissions from A only for those packets that are not correctly received. Notice, however, that for B to relay all the packets in order to the higher layer B must provide storage space for all packets received following a lost packet (i.e. if packet $i$ is lost, B must store all packets $i+1,...,n$ that A sends before retransmitting packet $i$).