ARQ Protocols

Handouts: Lecture notes for Lecture 2; Problem Set 2 out. Problem Set 1 in.

Lecture Outline:

- Automatic retransmission protocols:
  - Stop and wait protocol.
  - Go back n protocol.
  - Selective repeat transmission.
- Error recovery in the transport layer.

1 ARQ protocols

We will study three ARQ strategies: Stop and wait, Go back n, and Selective repeat.

1.1 Stop and wait ARQ

The basic idea behind the Stop and wait ARQ is that the sender ensures that each frame is received correctly before sending the next frame. One possible strategy is that the sender sends a packet in frame with CRC, then waits for an *ack* (acknowledgment) or *nak* (negative acknowledgment), which is also sent CRC-coded. If the sender receives a nak or an erroneous ack/nak, it resends the packet.

It is easy to see that the above protocol has potential problems. For example, what happens if an ack/nak gets lost (or delayed). If the sender resends after a timeout, how will the receiver distinguish the received packet from the sending of a new packet. The receiver should not look at the internals, because the actual data is meaningless to DLC.

Idea 1: Use sequence numbers in packet frame. As Figure 1 indicates, this is not enough.

Idea 2: The receiver indicates which packet it needs by attaching the requested sequence number in the request packet. This idea makes the protocol work. We now formalize the protocol.

The sender maintains a variable SN that indicates the sequence number of the packet being sent, and the receiver maintains a variable RN indicating the sequence number of the packet it is waiting for. More formally, the sender process is as follows:

1. $SN \Leftarrow 0$.
2. Accept new packet from the network layer if available, wait otherwise. Packet is numbered SN.
3. Transmit packet with SN in frame.
4. If error free frame received, containing $RN > SN$, set SN to RN and go to step 2. If no such frame is received within some finite delay, go to step 3.

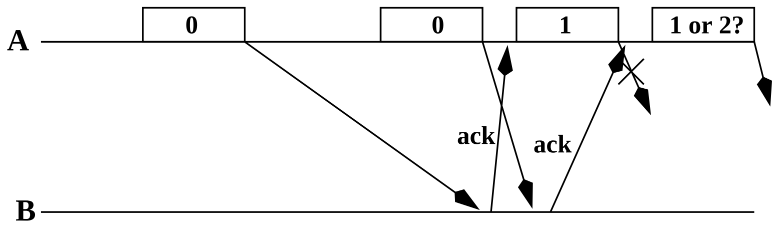

Figure 1: If sequence numbers alone are used and acks are not numbered, then the sender may send the wrong packet.

The receiver process is as follows.

1. Set $RN \Leftarrow 0$. Repeat steps 2 and 3 forever.
2. Whenever an error-free frame is received, containing $SN = RN$, release packet up to the network layer and set $RN \Leftarrow RN + 1$.
3. Within bounded delay of receiving any error-free data, transmit request frame with RN.

We note that the receiver ack/nak/request packets could be piggybacked on the return stream, if there is one from B to A.

Proof of correctness for the stop and wait protocol.

The correctness proof requires us to prove that the packets are delivered to the network layer of the receiver DLC in order, and without deletion or repetition. We establish the correctness by showing that the protocol is *safe* and *live*. Safety means that the protocol never produces an incorrect result. Liveness means that the protocol will continue forever to produce results.

The proof of safety is easy. Node B releases packets to the network layer in order. It increments RN by 1 only when packet with sequence number RN is received. Therefore safety is established.

We now consider the liveness property of the protocol. Assume A transmits a frame with sequence number i at time t_1. Node B returns $RN > i$ only after i has been received. Until that happens, A will continue to send i. Let t_2 be the time this packet is received by B and released up to the network layer. If this never happens, then we set $t_2 = \infty$. Let t_3 be the time the sequence number at A is increased to $i + 1$. If this never occurs, then $t_3 = \infty$.

Claim 1 *We have $t_1 < t_2 < t_3$ and $t_3 < \infty$.*

Proof: Let $RN(t)$ and $SN(t)$ denote the values of RN and SN, respectively, at time t. We note that both are non-decreasing functions of t. Furthermore, $SN(t) \leq RN(t)$ for all t. Since i has never been transmitted before t_1, we have $RN(t_1) \leq i$. But $SN(t_1) = i$; therefore, $RN(t_1) = i$. We also have $RN(t_2) = i + 1; SN(t_3) = i + 1$. It thus follows that $t_2 < t_3$ since $SN(t) \leq RN(t)$.

From time t_1, A keeps retransmitting packet i periodically, until time t_3. Since the probability of a successful transmission is $q > 0$, $t_2 - t_1$ is finite. After t_2, B sets RN to $i + 1$, keeps sending frame with $RN = i + 1$, within bounded delay of receiving an error-free frame from A. Since this happens continuously until t_3, one of the request frames is successfully transmitted in finite time. We thus have $t_3 - t_2 < \infty$. ■

This completeness the liveness proof. We note that the assumption of in-order delivery is not used in the proof. Indeed, the stop and wait protocol works correctly even if packets are delivered out of order.

1.2 Stop and wait using modulus 2

One of the drawbacks of the stop and wait protocol, as described, is that the sequence number grows without bound. If packets are delivered in order, however, it turns out that the sequence numbers can be sent modulo 2!

To see this, we note that between t_1 and t_2, RN and SN both equal i. Between t_2 and t_3, A has $SN = i$, while B has $RN = i + 1$. Therefore, when $RN = i + 1$, B receives packets with sequence number i or $i + 1$ only. Sending the sequence number modulo 2 is enough since the receiver can distinguish the two cases.

Similarly when $SN = i$, receiver sends $RN = i$ or $RN = i + 1$. So RN modulo 2 is sufficient. Since sequence numbers are being transmitted modulo 2 only, both A and B can just maintain SN and RN modulo 2 only. The stop and wait protocol with modulus 2 can now be described by a simple 4-state transition diagram as in Figure 2. The value of SN (mod 2) yields the state of A, while the value of RN (mod 2) yields state of B. This protocol is also referred to as the *Alternating Bit Protocol.*

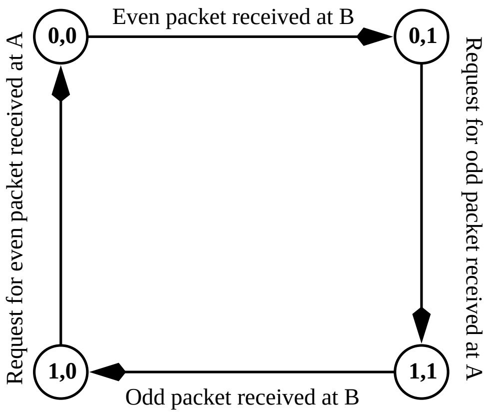

Figure 2: The stop and wait protocol with modulus 2. The pair (a, b) indicates that A is in state a and B is in state b.

2 The go back n protocol

The stop and wait ARQ protocol makes inefficient use of the underlying link, since it waits for an acknowledgement for a packet before sending the next. We now study the go back n protocol, in which the sender may send several successive packets without waiting for the next packet to be requested. Go back n ARQ is the most widely used ARQ protocol; it appears in various standard DLC protocols, such as HDLC, SDLC, and LAPB.

The go back n protocol shares the basic features of the stop and wait protocol. Incoming packets are numbered sequentially, and every frame contains the sequence number of the packet being carried. A receiver maintains a variable RN, periodically sends request frames requesting packet RN, and increments RN whenever it receives a frame with sequence number RN. Furthermore, as in the stop and wait protocol, a frame with sequence number RN acknowledges all packets prior to RN.

The main difference between the stop and wait protocol and the go back n protocol is that the sender may send several successive packets before a particular packet is acknowledged. The number of successive packets that the sender can send without receiving a request for a new packet is the parameter n. This parameter is also referred to as the size of the "window" of frames that can be in flight at any time. Go back n protocols and its variants are also called *sliding window* protocols.

Consider the example in Figure 3, which illustrates go back n with $n = 4$. The sender A sends packets 0 through 3 without receiving a request for 1, 2, or 3. When packet 0 is received by the receiver B, B sends a request frame to A with $RN = 1$. The sender then "slides" the window $[0, 3]$ one frame to the right to get $[1, 4]$ and sends out frame 4.

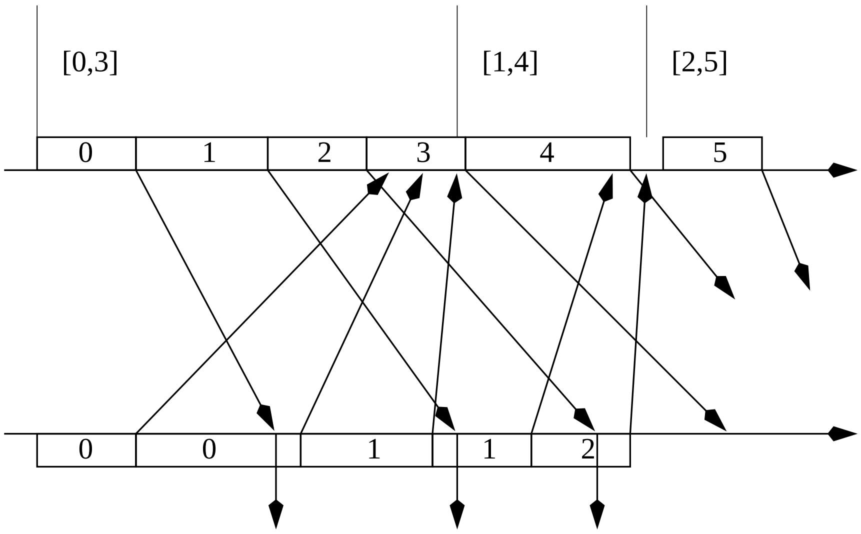

Figure 3: The go back n protocol.

Figure 4 illustrates the go back n protocol in the presence of errors. In this example, n is again set to 4. Sender A sets the initial window to $[0, 3]$ and sends out the first 4 packets, of which 1 is lost. When A receives a request frame for packet 1, it sets the window to $[1, 4]$ and sends out packet 4. It, however, times out when it does not receive a request frame with $RN = 2$, and retransmits the frames in the window.

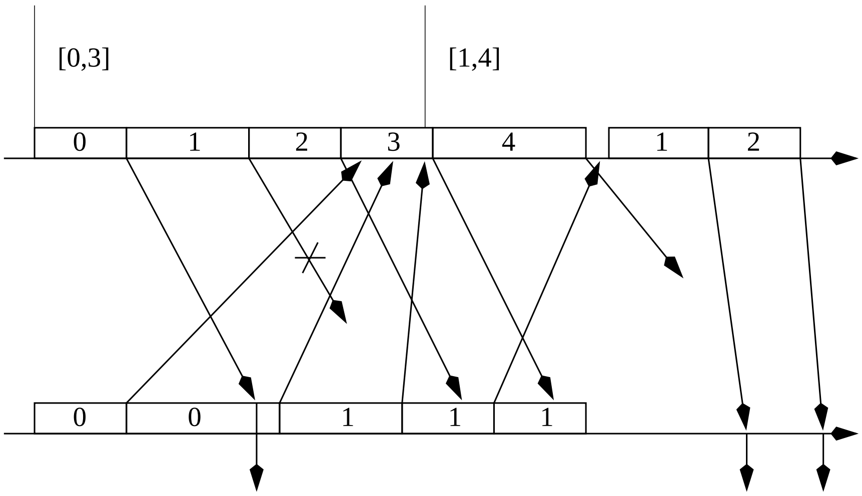

Figure 4: The go back n protocol in the presence of errors.

We now consider the sender algorithm in the go back n protocol in greater detail. The sender maintains two variables $SN_{\min}$ and $SN_{\max}$, which together determine the current window. The variable $SN_{\min}$ is the smallest numbered packet not yet acknowledged, while $SN_{\max}$ is the number that will be assigned to the next packet accepted from the network layer. Thus, the sender can transmit packets $SN_{\min}$ through $SN_{\max} - 1$; the current window is $[SN_{\min}, SN_{\max} - 1]$.

1. $SN_{\min} \leftarrow SN_{\max} \leftarrow 0$.

2. Do steps 2, 3, and 4 in any order. There can be an arbitrary but bounded delay between the time the conditions are satisfied to the time when the step is executed.

3. **Accepting packets from the network layer:** If $SN_{\max} < SN_{\min} + n$, and if a packet is available from the network layer, then accept the new packet, assign it sequence number $SN_{\max}$ and increment $SN_{\max}$.

4. **Receipt of an error-free frame:** If an error-free frame is received containing $RN > SN_{\min}$, then increment $SN_{\min}$ until it equals RN.

5. **Transmission of a frame:** If $SN_{\min} < SN_{\max}$, choose SN such that $SN_{\min} \leq SN < SN_{\max}$; transmit packet number SN. At most a bounded delay is allowed between successive transmissions of $SN_{\min}$ over intervals when $SN_{\min}$ does not change.

The receiver algorithm in the go back n protocol is identical to the algorithm in the stop and wait protocol.

The proof of correctness of go back n is along the same lines as that of the stop and wait protocol. We refer to the Bertsekas and Gallager text for details. We note that the correctness proof does not require the in-order delivery assumption.

2.1 Go back n with modulus $m > n$

Similar to the stop and wait protocol, the sequence numbers in the go back n protocol can be set such that they do not grow beyond bound under the important assumption that packets are delivered in order. In particular, the go back n protocol works correctly if the sender and receiver exchange sequence numbers modulo m, as long as $m > n$. We now show why this is true.

Suppose the sender A sends a frame with sequence number SN at time t_1 and this frame is received by B at time t_2 (see Figure 5). By the definition of the sender we have:

$$SN_{\min}(t_1) \leq SN \leq SN_{\min}(t_1) + n - 1. \tag{1}$$

Let the last time $SN_{\min}$ was updated before t_1 be due to a frame sent by B at time t_0. Clearly at time t_0, RN equals $SN_{\min}(t_1)$. Since packets are being received in order, any packet received by B in the time interval $[t_0, t_2]$ must have been sent by A when $SN_{\min} \leq SN_{\min}(t_1)$. (Here, we use the fact that $SN_{\min}(t)$ is a nondecreasing function of t.) Thus, B does not receive any packet with sequence number at least $SN_{\min}(t_1) + n$, which implies that $RN(t_2) \leq SN_{\min}(t_1) + n$. We have

$$SN_{\min}(t_1) \leq RN(t_2) \leq SN_{\min}(t_1) + n. \tag{2}$$

From Equations 1 and 2, we obtain $|RN(t_2) - SN| \leq n$. Therefore, since the receiver only needs to check whether a given packet satisfies $SN = RN$, this can be done even if the check is performed modulo $m > n$. Thus, the sender need only send SN modulo m as long as $m > n$.

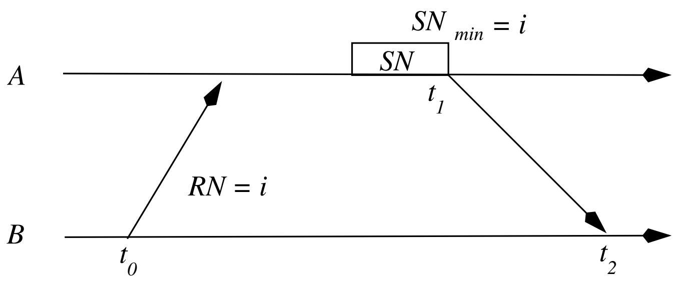

Figure 5: The relationship between $RN(t_2)$ and SN is given by $|RN(t_2) - SN| \leq n$. Therefore, it suffices to send $SN \bmod m$, where $m > n$, for the receiver to function correctly.

We now consider the frames received by the sender. It can be shown using an argument similar to the above that when the sender receives a request frame with sequence number RN at time t, the following inequality holds:

$$SN_{\min}(t) \leq RN \leq SN_{\min}(t) + n.$$

The sender needs to (i) check whether $RN = SN_{\min}$, (ii) distinguish between the different values of RN, and (iii) if $RN \neq SN_{\min}$ increase $SN_{\min}$ to equal RN. Since RN always lies in the interval $[SN_{\min}(t), SN_{\min}(t)+n]$, it is sufficient that both RN and $SN_{\min}$ are maintained modulo $m > n$.

2.2 Efficiency of go back n

Retransmissions or delays in the go back n protocol occur due to three reasons: (i) errors in the forward direction; (ii) errors in the backward direction; (iii) larger frames in the backward than in the forward direction. Errors in the backward direction or larger frames in the reverse direction imply that the acknowledgments take longer to arrive. While it is easy to see why errors in the backward direction pose a problem, the impact of larger frames in the reverse direction is not immediately clear. This is illustrated in Figure 6.

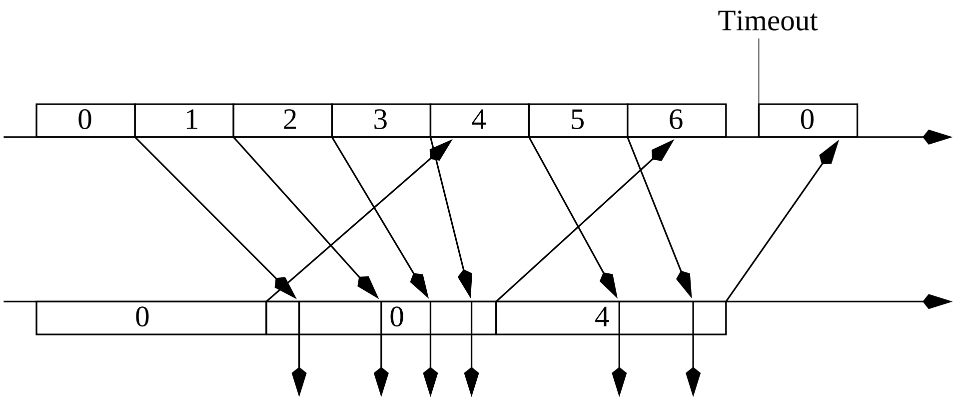

Figure 6: When frames in the reverse direction are larger than those in the forward direction, the sender may not receive an acknowledgment for the first packet in a window even when the entire window has been transmitted. This may cause the sender to timeout and retransmit frames even though no frames have been lost or incurred errors.

The problem of lost, erroneous, or delayed acknowledgments can be combatted by setting n sufficiently large. If n is large, however, errors in the forward direction may cause a large number of retransmissions, since the sender will have to retransmit a window-length of frames when a particular frame is lost. One possible solution is to set the sender timeout early. Of course, the timeout should be set at least larger than the round-trip time and other overhead involving the transmission of a packet frame and a request frame. Errors in the forward direction can also be addressed by having the receiver send short supervisory frames upon receiving a frame in error.

3 Selective repeat ARQ

One potential weakness of the go back n ARQ protocol is that when one packet in a window is dropped by the link, subsequent packets are not accepted by the receiver. A natural idea is to have the receiver accept the packets that are received out of order (in a certain range), buffer them, but not release them to the network layer until the "lost" packet is received. As in go back n, the receiver can send request frames with RN being the sequence number of the smallest packet that it has not received. Furthermore, the receiver may also send information to the sender about which packets ahead of RN have been received, so that the sender does not send them again.

The sender algorithm is the same as in go back n. Like the sender, the receiver in selective repeat ARQ also maintains a window of n sequence numbers. This window, which equals $[RN, RN+n-1]$, specifies the range of packets that the receiver accepts. When the packet with sequence number RN is received, RN is incremented to the smallest packet that has not been received.

We now consider whether the sequence numbers can be sent modulus m for a suitably large m. Equations 1 and 2, established for go back n, continue to hold. However, while the receiver in the go back n protocol only needs to determine whether SN of the incoming packet equals RN, the receiver in the selective repeat protocol has to distinguish between packets with sequence numbers in the range $[RN, RN+n-1]$ since the receiver buffers frames with sequence numbers higher than RN. Since we have

$$RN - n \leq SN \leq RN + n - 1,$$

we can (i) check whether $SN = RN$, and (ii) distinguish between the different packets in $[RN, RN+n-1]$, using modulo values if and only if the modulus $m \geq 2n$.

4 Error recovery at the transport layer

The transport layer is to the network layer what the data link layer is to the physical layer. The transport layer provides an error recovery mechanism using the same techniques as those used for the data link layer. It is important to understand the similarities and the differences in the problems of error recovery at the transport and data link layers.

Data link layer error recovery involves two nodes that are connected by a single link. Transport layer error-recovery (also referred to as end-to-end error recovery) involves two end hosts with a subnet or collection of networks connecting them. Both kinds of error recovery mechanisms use sequence numbers. Whereas in the data link layer, all the packets that traverse the link are numbered one after another, the numbering in the transport layer is done per session. Frames sent on a link may get lost, have errors, or incur a variable amount of delay. Similarly, packets sent from one host to another may get lost, have errors, and could certainly have a widely variable delay. The only real difference is that frames must stay in order on a link, whereas packets might arrive out of order on an end-to-end connection.

The potential of out-of-order delivery at the transport layer implies that while sliding window protocols such as go back n and selective repeat could be used, their variants with a fixed modulus may not work correctly. There are three possible ways out of this problem: (i) Require that the network deliver the packets in order. This can be done by setting up a fixed path between the source and destination and then using virtual circuit routing. Even with virtual circuit routing, however, out-of-order delivery cannot be eliminated in a straightforward manner. For example, when the particular path that has been selected goes down, packets need to be sent along a new path. If the old path comes back up, some packets that are still present on the old path may arrive at the destination. (ii) Use a very large modulus so that any resultant errors are acceptably unlikely. (iii) Create a mechanism whereby packets are destroyed within a maximum lifetime, and use a large enough modulus so that a wraparound does not take place during the lifetime of a packet. This mechanism is used in the transport protocol TCP.