Go-Back-N (GBN)

In a Go-Back-N (GBN) protocol, the sender is allowed to transmit several packets (when available) without waiting for an acknowledgment, but is constrained to have no more than some maximum allowable number, N, of unacknowledged packets in the pipeline. We describe the GBN protocol in some detail in this section. Figure 2 shows the sender's view of the range of sequence numbers in a GBN protocol. If we describe base to be the sequence number of the oldest unacknowledged packet and nextseqnum to be the smallest unused sequence number (that is, the sequence number of the next packet to be sent), then four intervals in the range of sequence numbers can be identified. Sequence numbers in the interval [0, base-1] correspond to packets that have already been transmitted and acknowledged. The interval [base, nextseqnum-1] corresponds to packets that have been sent but not yet acknowledged. Sequence numbers in the interval [nextseqnum, base+N-1] can be used for packets that can be sent immediately, should data arrive from the upper layer. In the end, sequence numbers greater than or equal to base+N cannot be used until an unacknowledged packet currently in the pipeline (particularly, the packet with sequence number base) has been acknowledge.

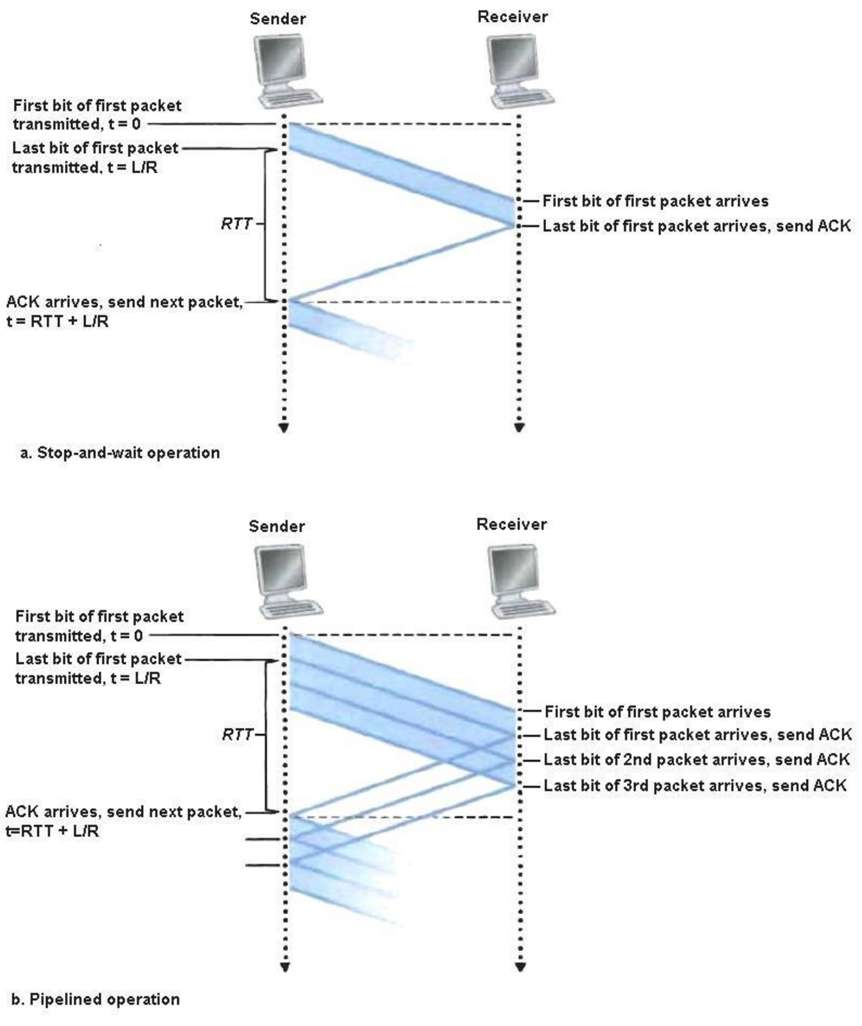
Sender
Receiver
First bit of first packet transmitted, t = 0
Last bit of first packet transmitted, t = L/R
First bit of first packet arrives
Last bit of first packet arrives, send ACK
RTT
ACK arrives, send next packet, t = RTT + L/R
a. Stop-and-wait operation
Sender
Receiver
First bit of first packet transmitted, t = 0
Last bit of first packet transmitted, t = L/R
First bit of first packet arrives
Last bit of first packet arrives, send ACK
Last bit of 2nd packet arrives, send ACK
Last bit of 3rd packet arrives, send ACK
RTT
ACK arrives, send next packet, t=RTT + L/R
b. Pipelined operation

Figure 1. Stop-and-wait and pipelined sending

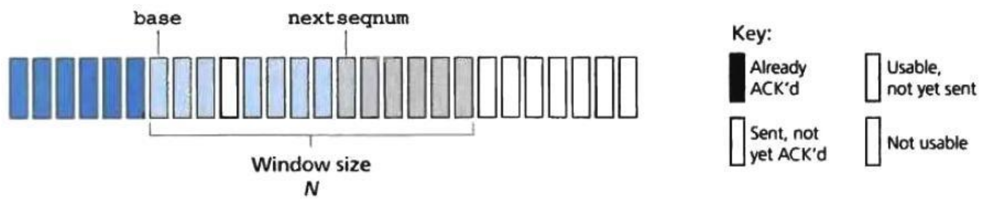

Figure 2. Sender's view of sequence numbers in Go-Back-N

As suggested by Figure 2, the range of permissible sequence numbers for transmitted but not yet acknowledged packets can be viewed as a window of size N over the range of sequence numbers. As the protocol operates, this window slides forward over the sequence number space. Therefore, N is often referred to as the window size and the GBN protocol itself as a sliding-window protocol. You might be wondering why we would even limit the number of outstanding, unacknowledged packets to a value of N in the first place. Why not allow an unlimited number of such packets? We'll see in "Connection-Oriented Transport: TCP" that flow control is one reason to impose a limit on the sender. Well look at another reason to do so in "TCP Congestion Control", when we study TCP congestion control.

In fact, a packets sequence number is carried in a fixed-length field in the packet header. If k is the number of bits in the packet sequence number field, the range of sequence numbers is thus [0,2k-1]. With a finite range of sequence numbers, all arithmetic involving sequence numbers must then be done using modulo 2karithmetic. (That is, the sequence number space can be thought of as ring of size 2k, where sequence number 2k-1 is immediately followed by sequence number 0.) Recall that rdt3.0 had a 1-bit sequence number and a range of sequence numbers of [0,1]. We will see in "Connection-Oriented Transport: TCP" that TCP has a 32-bit sequence number field, where TCP sequence numbers count bytes in the byte stream rather than packets.

Figures 3 and 4 give an extended FSM description of the sender and receiver sides of an ACK-based, NAK-free, GBN protocol. We refer to this FSM description as an extended FSM because we have added variables (similar to programming-language variables) for base and nextseqnum, and added operations on these variables and conditional actions involving these variables. Note that the extended FSM specification is now beginning to look somewhat like a programming-language specification. [Bochman 1984] provides an outstanding survey of additional extensions to FSM techniques as well as other programming-language-based techniques for specifying protocols.

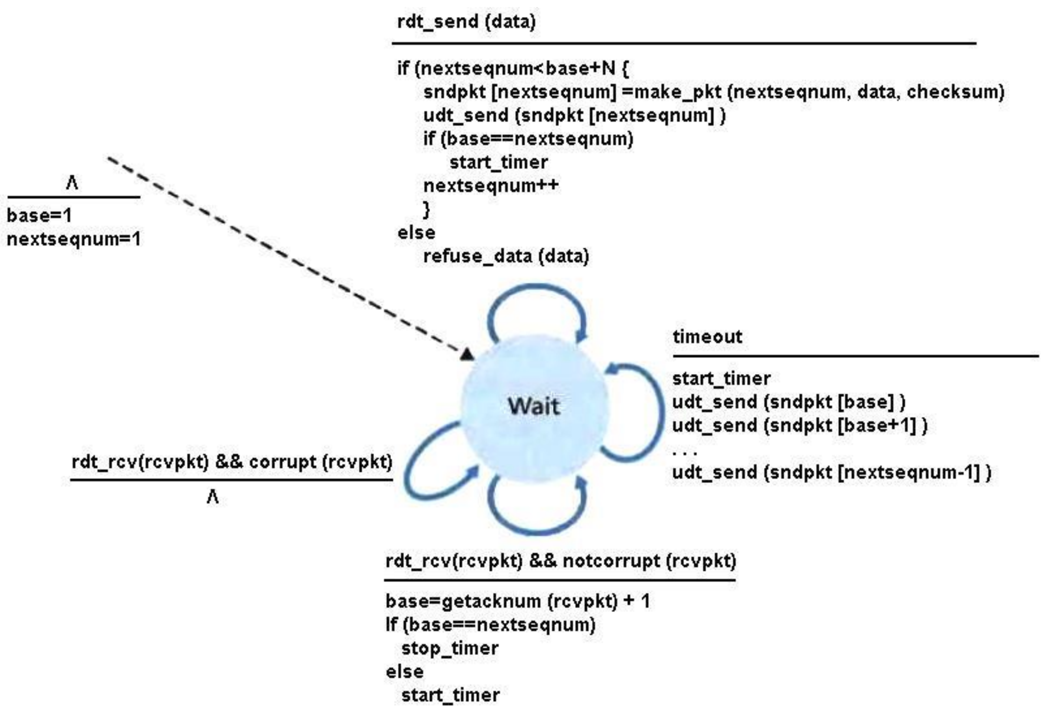
rdt_send (data)
if (nextseqnum<base+N {
    sndpkt [nextseqnum] =make_pkt (nextseqnum, data, checksum)
    udt_send (sndpkt [nextseqnum])
    if (base==nextseqnum)
        start_timer
    nextseqnum++
    }
else
    refuse_data (data)
Λ
base=1
nextseqnum=1
Wait
timeout
start_timer
udt_send (sndpkt [base])
udt_send (sndpkt [base+1])
. . .
udt_send (sndpkt [nextseqnum-1])
rdt_rcv(rcvpkt) && corrupt (rcvpkt)
Λ
rdt_rcv(rcvpkt) && notcorrupt (rcvpkt)
base=getacknum (rcvpkt) + 1
If (base==nextseqnum)
  stop_timer
else
  start_timer

Figure 3. Extended FSM description of GBN sender

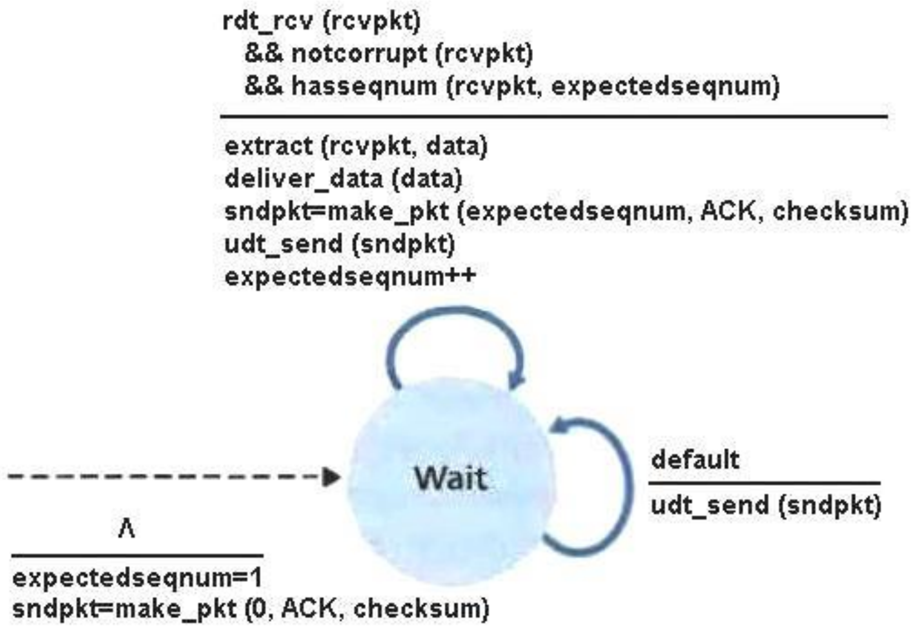

Figure 4. Extended FSM description of GBN receiver

The GBN sender must respond to three types of events:

- Invocation from above. When rdt_send ( ) is called from above, the sender first checks to see if the window is full, that is, whether there are N outstanding, unacknowledged packets. If the window is not full, a packet is created and sent, and variables are appropriately updated. If the window is full, the sender simply returns the data back to the upper layer, an implicit indication that the window is full. The upper layer would most likely then have to try again later. In a real implementation, the sender would presumably have either buffered (but not immediately sent) this data, or would have a synchronization mechanism (for instance, a semaphore or a flag) that would allow the upper layer to call rdt_send ( ) only when the window is not full.
- Receipt of an ACK. In our GBN protocol, an acknowledgment for a packet with sequence number n will be taken to be a cumulative acknowledgment, indicating that all packets with a sequence number up to and including n have been correctly received at the receiver. We'll come back to this problem soon when we study the receiver side of GBN.
- A timeout event. The protocol's name. "Go-Back-N", is derived from the sender's behavior in the presence of lost or overly delayed packets. As in the stop-and-wait protocol, a timer will again be used to recover from lost data or acknowledgment packets. If a timeout takes place, the sender resends all packets that have been previously sent but that have not yet been acknowledged. Our sender in Figure 3 uses only a single timer, which can be thought of as a timer for the oldest transmitted but not yet acknowledged packet. If an ACK is received but there are still additional transmitted but not yet acknowledged packets, the timer is restarted. If there are no outstanding, unacknowledged packets, the timer is stopped.

The receiver's actions in GBN are also simple. If a packet with sequence number n is received correctly and is in order (that is, the data last delivered to the upper layer came from a packet with sequence number n -1), the receiver sends an ACK for packet n and delivers the data portion of the packet to the upper layer. In all other cases, the receiver discards the packet and resends an ACK for the most recently received in-order packet. Note that since packets are delivered one at a time to the upper layer, if packet k has been received and delivered, then all packets with a sequence number lower than k have also been delivered. In this way, the use of cumulative acknowledgments is a natural choice for GBN.

In our GBN protocol, the receiver discards out-of-order packets. Although it may seem silly and wasteful to discard a correctly received (but out-of-order) packet, there is some justification for doing so. Recall that the receiver must deliver data in order to the upper layer. Suppose now that packet n is expected, but packet n + 1 arrives. Because data must be delivered in order, the receiver could buffer (save) packet n + 1 and then deliver this packet to the upper layer after it had later received and delivered packet n. However, if packet n is lost, both it and packet n + 1 will finally be retransmitted as a result of the GBN retransmission rule at the sender. Therefore, the receiver can simply discard packet n + 1. The advantage of this approach is the simplicity of receiver buffering - the receiver need not buffer any out-of-order packets. Therefore, while the sender must maintain the upper and lower bounds of its window and the position of nextseqnum within this window, the only piece of information the receiver need maintain is the sequence number of the next in-order packet. This value is held in the variable expectedseqnum, shown in the receiver FSM in Figure 4. Of course, the disadvantage of throwing away a correctly received packet is that the subsequent retransmission of that packet might be lost or garbled and thus even more retransmissions would be required.

"Selective Repeat (SR)" Figure 1 shows the operation of the GBN protocol for the case of a window size of four packets. Because of this window size limitation, the sender sends packets 0 through 3 but then must wait for one or more of these packets to be acknowledged before proceeding. As each successive ACK (for example, ACK0 and ACK1) is received, the window slides forward and the sender can transmit one new packet (pkt4 and pkt5, respectively). On the receiver side, packet 2 is lost and thus packets 3, 4 and 5 are found to be out of order and are discarded.

Before closing our discussion of GBN, it is worth noting that an implementation of this protocol in a protocol stack would likely have a structure similar to that of the extended FSM in Figure 3. The implementation would also likely be in the form of various procedures that implement the actions to be taken in response to the various events that can occur. In such event-based programming, the various procedures are called (invoked) either by other procedures in the protocol stack, or as the result of an interrupt. In the sender, these events would be (1) a call from the upper-layer entity to invoke rdt_send( ), (2) a timer interrupt, and (3) a call from the lower layer to invoke rdt_rcv( ) when a packet arrives.

We note here that the GBN protocol integrates almost all of the techniques that we will encounter when we learn the reliable data transfer components of TCP in "Connection-Oriented Transport: TCP". These techniques contain the use of sequence numbers, cumulative acknowledgments, checksums, and a timeout/retransmit operation.