CIS 457 - Data Communications

Nathan Bowman

Images taken from Kurose and Ross book

Selective Repeat (SR)

Go-Back-N is more efficient than stop-and-wait, but it can result in many packets being re-sent

Recall that any time a packet is lost, all subsequent packets will also be sent again

With a deep pipeline (large N), losing a packet from early in the pipeline can be costly

An alternative protocol, **selective repeat** (SR), tries to avoid this by keeping track of each packet individually

In SR, sender still has window of size N, meaning it can send no more than N packets after the last ACK'd packet

However, receiver will now accept and *buffer* packets delivered too early

If packet 1 is lost, receiver will still accept packets 2, ..., N and hold on to them

To keep sender from re-sending everything, receiver must ACK individual packets

This is one of the key differences between GBN and SR --

in GBN, an ACK for a packet is also an ACK for every packet up to that point (called *cumulative ACK*)

in SR, and ACK for a packet applies *only to that packet*

This complicates implementation for sender as well, because now sender must keep track of whether each packet in window has been ACK'd

Receiver cannot deliver to application until packets have arrived in order

Assume packets 2, 3, and 4 arrived -- they will be held in a buffer (and ACKs will be sent)

Once packet 1 arrives, all packets 1 - 4 can be delivered to application at once

If repeat packet, meaning packet that receiver already ACK'd, is delivered, receiver must ACK again

Consider: if packet was sent again, it may be because ACK was lost first time, so sender cannot proceed unless ACK sent again

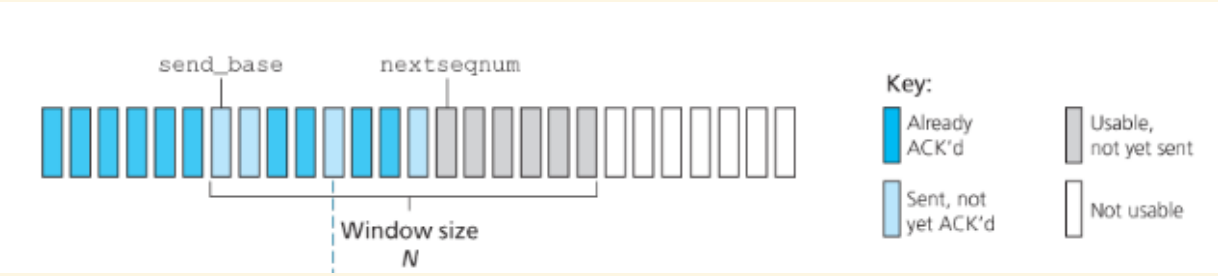
In short, SR may be more "natural" protocol if you were designing it yourself:

- sender sends at most N messages
- receiver ACKs and holds on to each received message
- sender tracks each ACK so it knows which messages may need to be re-sent after a timeout
- when sender receives ACK for oldest message in window, it can move window forward and send another

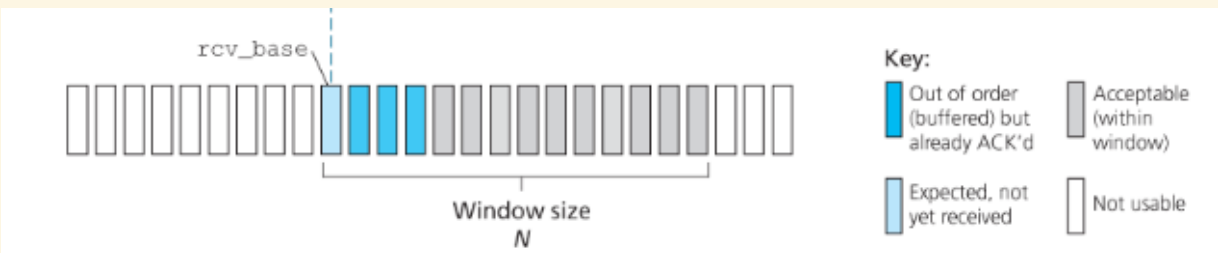
Important to note that sender and receiver may have different view of situation

Until ACK received by sender, for example, receiver may know packet was correctly delivered, but sender still believes it is in transit

Sender


send_base
nextseqnum
Window size
N
Key:
Already ACK'd
Sent, not yet ACK'd
Usable, not yet sent
Not usable

Receiver

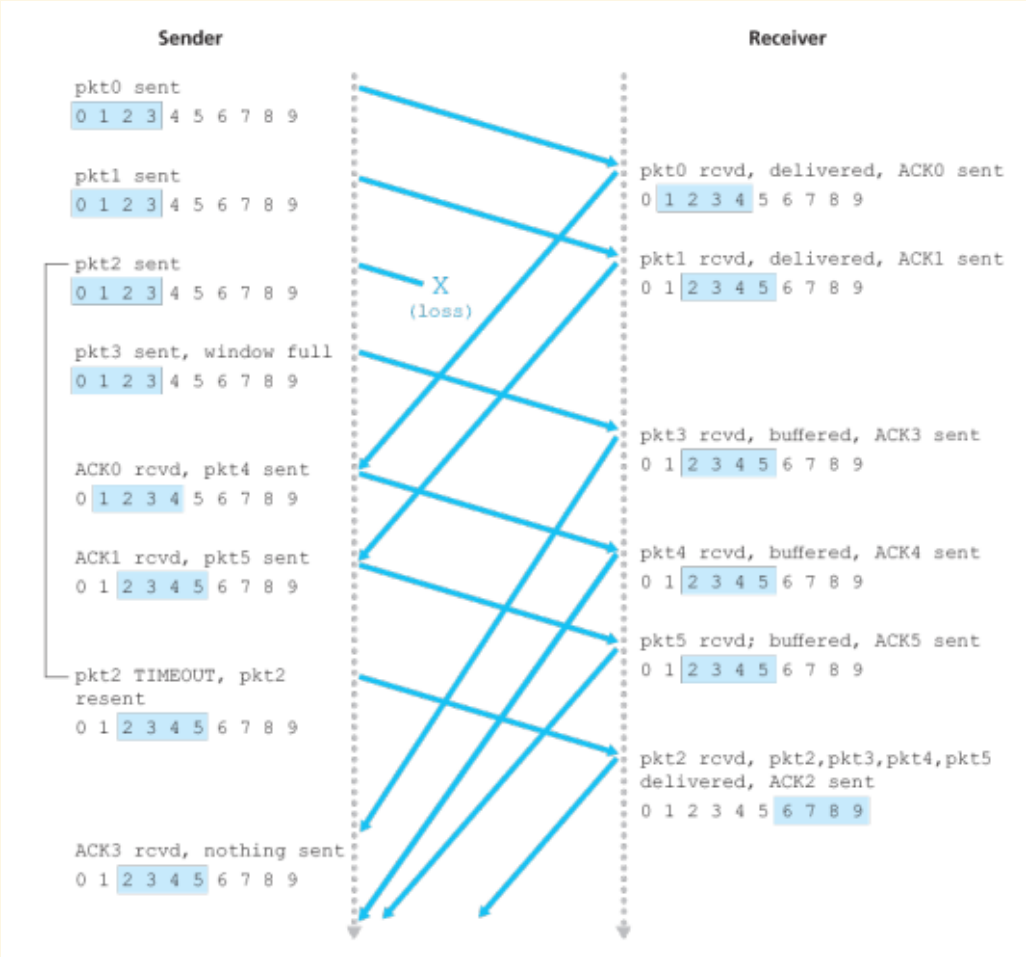

rcv_base
Window size
N
Key:
Out of order (buffered) but already ACK'd
Acceptable (within window)
Expected, not yet received
Not usable

Sender Description

1. *Data received from above.* When data is received from above, the SR sender checks the next available sequence number for the packet. If the sequence number is within the sender's window, the data is packetized and sent; otherwise it is either buffered or returned to the upper layer for later transmission, as in GBN.
2. *Timeout*. Timers are again used to protect against lost packets. However, each packet must now have its own logical timer, since only a single packet will be transmitted on timeout. A single hardware timer can be used to mimic the operation of multiple logical timers [Varghese 1997].
3. *ACK received*. If an ACK is received, the SR sender marks that packet as having been received, provided it is in the window. If the packet's sequence number is equal to `send_base`, the window base is moved forward to the unacknowledged packet with the smallest sequence number. If the window moves and there are untransmitted packets with sequence numbers that now fall within the window, these packets are transmitted.

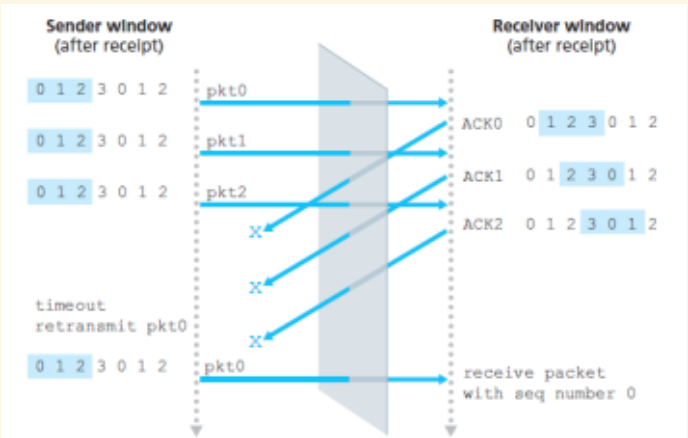
Receiver Description

1. *Packet with sequence number in* `[rcv_base, rcv_base+N-1]` *is correctly received.* In this case, the received packet falls within the receiver's window and a selective ACK packet is returned to the sender. If the packet was not previously received, it is buffered. If this packet has a sequence number equal to the base of the receive window (`rcv_base` in Figure 3.22), then this packet, and any previously buffered and consecutively numbered (beginning with `rcv_base`) packets are delivered to the upper layer. The receive window is then moved forward by the number of packets delivered to the upper layer. As an example, consider Figure 3.26. When a packet with a sequence number of `rcv_base=2` is received, it and packets 3, 4, and 5 can be delivered to the upper layer.
2. *Packet with sequence number in* `[rcv_base-N, rcv_base-1]` *is correctly received.* In this case, an ACK must be generated, even though this is a packet that the receiver has previously acknowledged.
3. *Otherwise.* Ignore the packet.

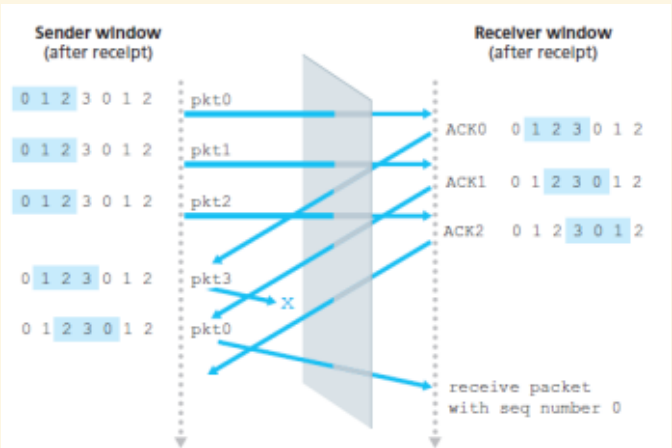

Sender
Receiver
pkt0 sent
0 1 2 3 4 5 6 7 8 9
pkt1 sent
0 1 2 3 4 5 6 7 8 9
pkt2 sent
0 1 2 3 4 5 6 7 8 9
X
(loss)
pkt3 sent, window full
0 1 2 3 4 5 6 7 8 9
ACK0 rcvd, pkt4 sent
0 1 2 3 4 5 6 7 8 9
ACK1 rcvd, pkt5 sent
0 1 2 3 4 5 6 7 8 9
pkt2 TIMEOUT, pkt2
resent
0 1 2 3 4 5 6 7 8 9
ACK3 rcvd, nothing sent
0 1 2 3 4 5 6 7 8 9
pkt0 rcvd, delivered, ACK0 sent
0 1 2 3 4 5 6 7 8 9
pkt1 rcvd, delivered, ACK1 sent
0 1 2 3 4 5 6 7 8 9
pkt3 rcvd, buffered, ACK3 sent
0 1 2 3 4 5 6 7 8 9
pkt4 rcvd, buffered, ACK4 sent
0 1 2 3 4 5 6 7 8 9
pkt5 rcvd; buffered, ACK5 sent
0 1 2 3 4 5 6 7 8 9
pkt2 rcvd, pkt2,pkt3,pkt4,pkt5
delivered, ACK2 sent
0 1 2 3 4 5 6 7 8 9

Because sender and receiver may have different view of situation, need to be careful with sequence numbers

Consider example where window size N is just one less than number of available sequence numbers


Sender window
(after receipt)
0 1 2 3 0 1 2
0 1 2 3 0 1 2
0 1 2 3 0 1 2
timeout
retransmit pkt0
0 1 2 3 0 1 2
pkt0
pkt1
pkt2
X
X
X
pkt0
Receiver window
(after receipt)
ACK0 0 1 2 3 0 1 2
ACK1 0 1 2 3 0 1 2
ACK2 0 1 2 3 0 1 2
receive packet
with seq number 0

In case just shown, sender was re-sending packet 0


Sender window
(after receipt)
Receiver window
(after receipt)
0 1 2 3 0 1 2
0 1 2 3 0 1 2
0 1 2 3 0 1 2
0 1 2 3 0 1 2
0 1 2 3 0 1 2
pkt0
pkt1
pkt2
pkt3
X
pkt0
ACK0 0 1 2 3 0 1 2
ACK1 0 1 2 3 0 1 2
ACK2 0 1 2 3 0 1 2
receive packet
with seq number 0

In second case, sender was sending *new* packet 0

Note that both cases looked identical to receiver

Receiver has no way of knowing whether packet is new or repeated

Need to ensure enough sequence numbers available such that they will not wrap around too soon

For SR, if N <= (1/2) * [number of sequence numbers], protocol will not run into this issue

To understand why, consider worst case: if all packets received but all ACKs lost, receiver window will be shifted by N from sender window

As long as N + N <= [number sequence #s], receiver window cannot wrap around and become confused with sender window

What if packet is "lost" in network for a time and randomly spit out later?

It could have old sequence number that happens to be in use again

In practice, avoid this by assuming some maximum time packet can "live" in network and have enough sequence numbers to avoid reuse during that time

For internet, three minutes is generally enough time

Have yet to discuss actual mechanisms in TCP, but discussion so far has given us general tools for reliable data transfer:

- checksum
- timer
- sequence number

...continued...

- acknowledgement
- negative acknowledgement
- window/pipelining

We will see many of these tools when we look at actual TCP protocol