CIS 457 - Data Communications

Nathan Bowman

Images taken from Kurose and Ross book

Go-Back-N (GBN)

Need reliable transfer protocols to be efficient

To do so, must use pipelining

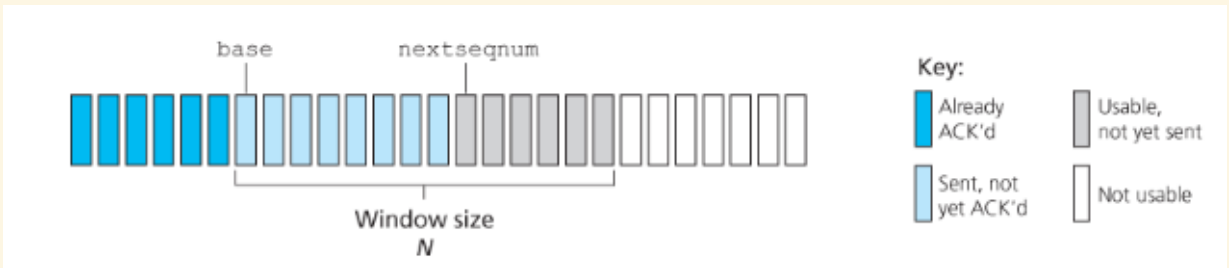
First pipelining protocol examined is **Go-Back-N** (GBN)

Can have up to N un-ACKed packets in pipeline at once

After that, sender must wait for ACKs to send out more

Receiver still accepts packets only in correct order

If packet received out of order (because previous packet was dropped), receiver simply discards


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

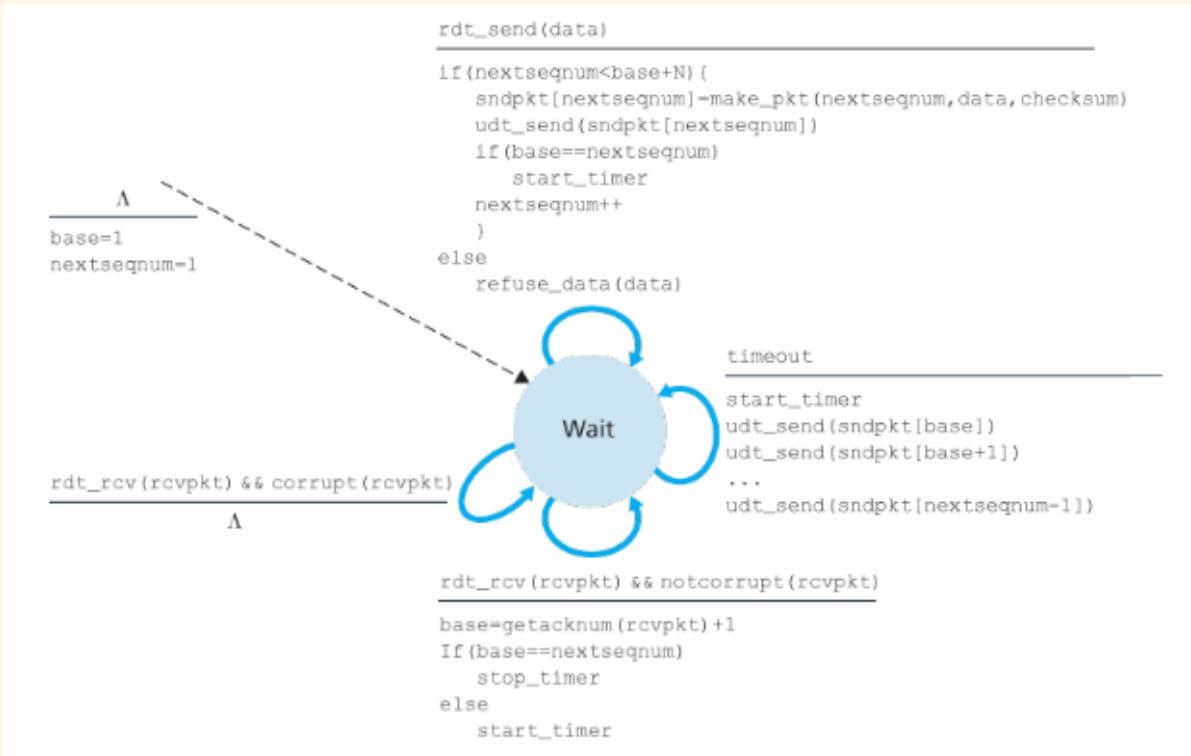
As seen in image, window of size N that slides forward as ACKs received

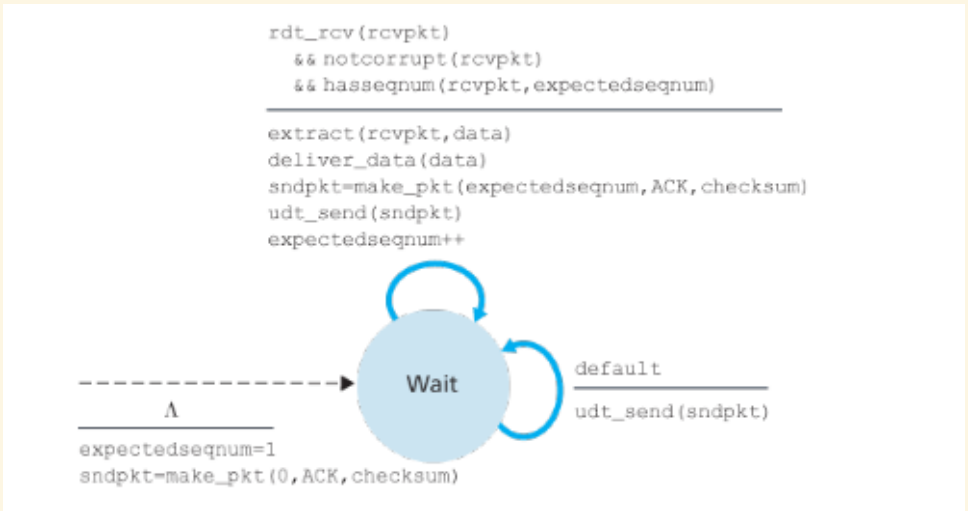
Hence, a **sliding-window** protocol

Because more packets are in channel at once, need more sequence numbers

Packet has fixed-length header of size k bits for storing sequence number

Possible numbers are [0, (2^k) - 1]


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


rdt_rcv(rcvpkt)
&& notcorrupt(rcvpkt)
&& hasseqnum(rcvpkt,expectedseqnum)
extract(rcvpkt,data)
deliver_data(data)
sndpkt=make_pkt(expectedseqnum,ACK,checksum)
udt_send(sndpkt)
expectedseqnum++
Wait
default
udt_send(sndpkt)
Λ
expectedseqnum=1
sndpkt=make_pkt(0,ACK,checksum)

Sender

Sender must respond to three types of event

1. Application calls send -- check whether window is full and either send or refuse

Sender

2. Receives ACK --

cumulative acknowledgment of all packets up to ACK'd one

For example, if packets 3, 4, and 5 delivered, but ACKs for 3 and 4 lost, ACK for 5 notifies sender that all were received correctly

Easy to keep track of which packets sender knows were delivered correctly

If ACK larger than previous base, slide window

Sender

3. Timeout -- no ACK received for current base packet, so sender suspects packet is lost

In this case, resent *all* packets

Since receiver accepts only in order, if oldest packet did not arrive, no packets were accepted by receiver

Only keeps one timer at a time, rather than one for every packet

Receiver

Receiver is very simple

If packet received is next in order, send ACK to sender and deliver packet to application layer

Otherwise, discard packet and send ACK for most recent successful packet

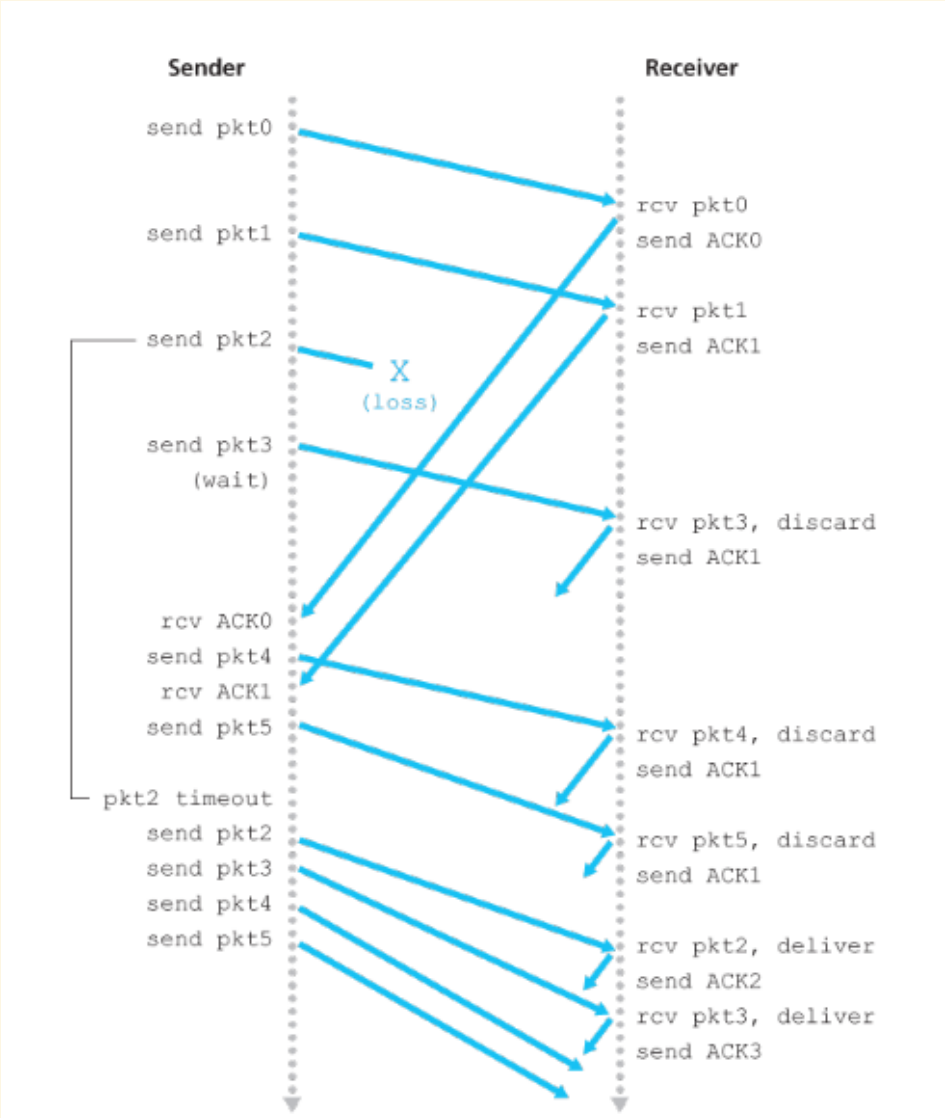
Receiver

Seems wasteful to have receiver not buffer packets, but sender is going to resend all packets in case of lost packet anyway

Why not change the sender then?

Receiver not buffering out-of-order packets keeps sender and receiver simple and means no buffer space required

Not without its tradeoffs


Sender
Receiver
send pkt0
send pkt1
send pkt2
X
(loss)
send pkt3
(wait)
rcv ACK0
send pkt4
rcv ACK1
send pkt5
pkt2 timeout
send pkt2
send pkt3
send pkt4
send pkt5
rcv pkt0
send ACK0
rcv pkt1
send ACK1
rcv pkt3, discard
send ACK1
rcv pkt4, discard
send ACK1
rcv pkt5, discard
send ACK1
rcv pkt2, deliver
send ACK2
rcv pkt3, deliver
send ACK3