Sender:
- seq # added to pkt
- two seq. #’s (0,1) will suffice. Why?
- must check if received ACK/NAK corrupted
- twice as many states
  - state must “remember” whether “current” pkt has 0 or 1 seq #

Receiver:
- must check if received packet is duplicate
  - state indicates whether 0 or 1 is expected pkt seq #
- note: receiver can not know if its last ACK/NAK received OK at sender
rdt2.2: a NAK-free protocol

• same functionality as rdt2.1, using ACKs only
• instead of NAK, receiver sends ACK for last pkt received OK
  – receiver must *explicitly* include seq # of pkt being ACKed
• duplicate ACK at sender results in same action as NAK: *retransmit current pkt*
**rdt2.2: sender, receiver fragments**

**sender FSM fragment**

- `rdt_send(data)`
  - `sndpkt = make_pkt(0, data, checksum)`
  - `udt_send(sndpkt)`
- `Wait for call 0 from above`
- `Wait for ACK 0`
- `rdt_rcv(rcvpkt) && (corrupt(rcvpkt) || isACK(rcvpkt,1))`
- `udt_send(sndpkt)`

**receiver FSM fragment**

- `rdt_rcv(rcvpkt) && notcorrupt(rcvpkt) && isACK(rcvpkt,0)`
- `Wait for 0 from below`
- `extract(rcvpkt, data)`
- `deliver_data(data)`
- `sndpkt = make_pkt(ACK1, checksum)`
- `udt_send(sndpkt)`
- `rdt_rcv(rcvpkt) && notcorrupt(rcvpkt) || has_seq1(rcvpkt)`
- `udt_send(sndpkt)`
**New assumption:**
underlying channel can also lose packets (data or ACKs)
- checksum, seq. #, ACKs, retransmissions will be of help, but not enough

**Approach:**
- sender waits “reasonable” amount of time for ACK
- retransmits if no ACK received in this time
- if pkt (or ACK) just delayed (not lost):
  - retransmission will be duplicate, but use of seq. #’s already handles this
  - receiver must specify seq # of pkt being ACKed
- requires countdown timer
rdt3.0 sender

rdt_send(data)

sendpkt = make_pkt(0, data, checksum)
udt_send(sndpkt)
start_timer

Wait for call 0 from above

Wait for ACK0

Stop timer

timeout
udt_send(sndpkt)
start_timer

Wait for call 1 from above

rdt_send(data)

sendpkt = make_pkt(1, data, checksum)
udt_send(sndpkt)
start_timer

rdt_rcv(rcvpkt) &&
( corrupt(rcvpkt) ||
isACK(rcvpkt,1) )
Λ

rdt_rcv(rcvpkt) &&
notcorrupt(rcvpkt)
&& isACK(rcvpkt,1)
stop_timer

rdt_rcv(rcvpkt) &&
notcorrupt(rcvpkt)
&& isACK(rcvpkt,0)
stop_timer

rdt_rcv(rcvpkt)
Λ
rdt3.0 in action

(a) operation with no loss

(b) lost packet
rdt3.0 in action

(c) lost ACK

(d) premature timeout
Performance of rdt3.0

- rdt3.0 works, but performance stinks
- ex: 1 Gbps link, 15 ms prop. delay, 8000 bit packet:

\[ d_{\text{trans}} = \frac{L}{R} = \frac{8000\text{bits}}{10^9 \text{bps}} = 8\text{microseconds} \]

\[ U_{\text{sender}} : \text{utilization} - \text{fraction of time sender busy sending} \]

\[ U_{\text{sender}} = \frac{L / R}{\text{RTT} + L / R} = \frac{.008}{30.008} = 0.00027 \]

- 1KB pkt every 30 msec -> 33kB/sec thruput over 1 Gbps link
- network protocol limits use of physical resources!
**rdt3.0: stop-and-wait operation**

First packet bit transmitted, \( t = 0 \)

Last packet bit transmitted, \( t = \frac{L}{R} \)

First packet bit arrives

Last packet bit arrives, send ACK

ACK arrives, send next packet, \( t = RTT + \frac{L}{R} \)

\[
U_{\text{sender}} = \frac{\frac{L}{R}}{RTT + \frac{L}{R}} = \frac{0.008}{30.008} = 0.00027
\]