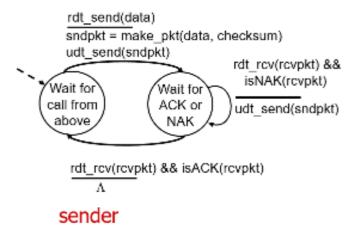
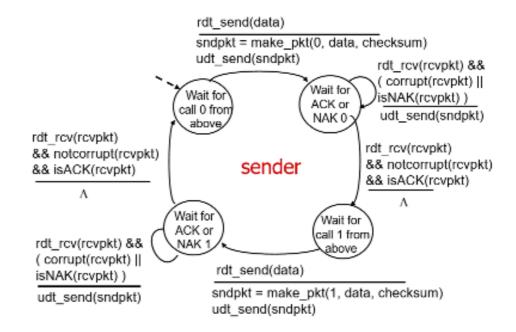
## rdt2.0 ⇒ rdt2.1 Senders Add Sequence Number

#### rdt2.0: FSM specification

#### rdt2.1: sender, handles garbled ACK/NAKs





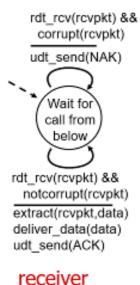
- data now has sequence number
- ACK/NAK can be corrupted

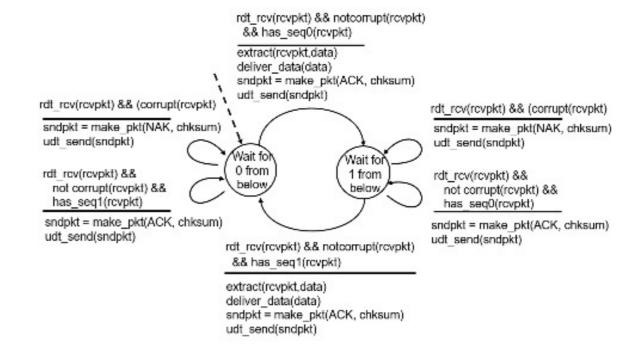


## rdt2.0 ⇒ rdt2.1 Receivers Add Sequence Number

#### rdt2.0: FSM

rdt2.1: receiver, handles garbled ACK/NAKs



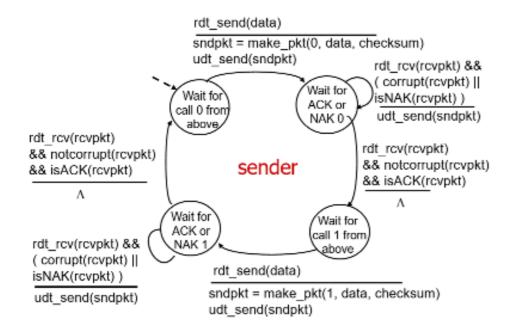


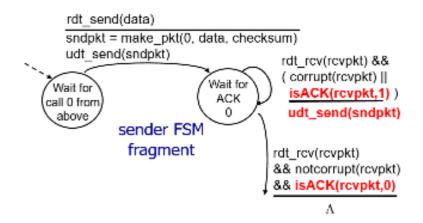
- data now has sequence number
  - need to detect duplicate data packet
- ACK/NAK packet now contains checksum



## rdt2.1 ⇒ rdt2.2 Senders Replace NAK with DupACK

rdt2.1: sender, handles garbled ACK/NAKs rdt2.2: sender





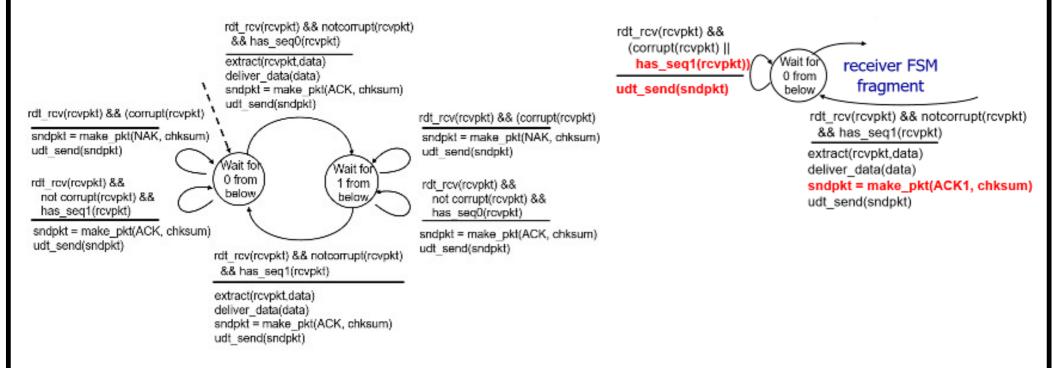
- ACK now has sequence number for last packet received
- NAK and DupACK are the same thing, can merge transitions
  - DupACK = ACK with the wrong sequence number



# rdt2.1 ⇒ rdt2.2 Receivers Replace NAK with DupACK

#### rdt2.1: receiver, handles garbled ACK/NAKs ro

rdt2.2: receiver



- ACK now has sequence number for last packet received
- NAK and DupACK are the same thing, can merge transitions
  - DupACK = ACK with the wrong sequence number

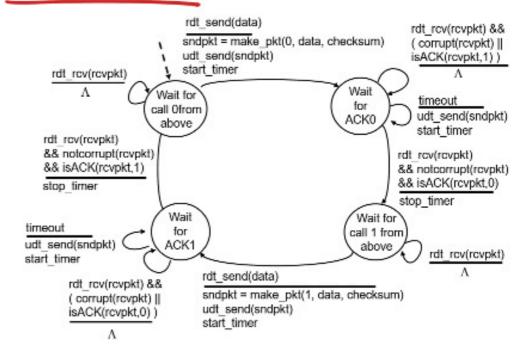


## rdt2.2 ⇒ rdt3.0 Senders Add Timeout

#### rdt2.2: sender

#### rdt\_send(data) sndpkt = make\_pkt(0, data, checksum) udt\_send(sndpkt) rdt\_rev(revpkt) && ( corrupt(rcvpkt) || Wait for Wait for isACK(rcvpkt,1)) ACK call 0 from udt send(sndpkt) above sender FSM fragment rdt rev(revpkt) && notcorrupt(rcvpkt) && isACK(rcvpkt,0)

#### rdt3.0 sender



- udt\_send(sndpkt) is followed by start\_timer
- when ACK is received, stop\_timer
- retransmit when timeout event occurs



### rdt3.0



- rdt3.0 is sometimes known as the alternating-bit protocol
- this is a working reliable data transfer protocol
- ingredients:
  - checksums
  - sequence numbers
  - timers
  - ACKs and NACKs

