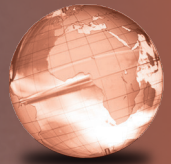


GLOBAL
EDITION



Computer Networking

A Top-Down Approach

EIGHTH EDITION

James F. Kurose • Keith W. Ross



DIGITAL RESOURCES FOR STUDENTS

Your new textbook provides 12-month access to digital resources that may include VideoNotes (illustrating key concepts from the text), interactive exercises, interactive animations, quizzes, and more. Refer to the preface in the textbook for a detailed list of resources.

Follow the instructions below to register for the Companion Website for James F. Kurose and Keith W. Ross's ***Computer Networking: A Top-Down Approach, Eighth Edition, Global Edition***.

- 1 Go to **www.pearsonglobaleditions.com**.
- 2 Enter the title of your textbook or browse by author name.
- 3 Click Companion Website.
- 4 Click Register and follow the on-screen instructions to create a login name and password.

ISSJKK-FROMM-DAIRY-CUPPA-PLUSH-POSES

Use the login name and password you created during registration to start using the digital resources that accompany your textbook.

For technical support go to **<https://support.pearson.com/getsupport>**

(the sequence number of the received packet has the same sequence number as the most recently received packet) or a new packet (the sequence number changes, moving “forward” in modulo-2 arithmetic). Since we are currently assuming a channel that does not lose packets, ACK and NAK packets do not themselves need to indicate the sequence number of the packet they are acknowledging. The sender knows that a received ACK or NAK packet (whether garbled or not) was generated in response to its most recently transmitted data packet.

Figures 3.11 and 3.12 show the FSM description for `rdt2.1`, our fixed version of `rdt2.0`. The `rdt2.1` sender and receiver FSMs each now have twice as many states as before. This is because the protocol state must now reflect whether the packet currently being sent (by the sender) or expected (at the receiver) should have a sequence number of 0 or 1. Note that the actions in those states where a 0-numbered packet is being sent or expected are mirror images of those where a 1-numbered packet is being sent or expected; the only differences have to do with the handling of the sequence number.

Protocol `rdt2.1` uses both positive and negative acknowledgments from the receiver to the sender. When an out-of-order packet is received, the receiver sends a positive acknowledgment for the packet it has received. When a corrupted packet

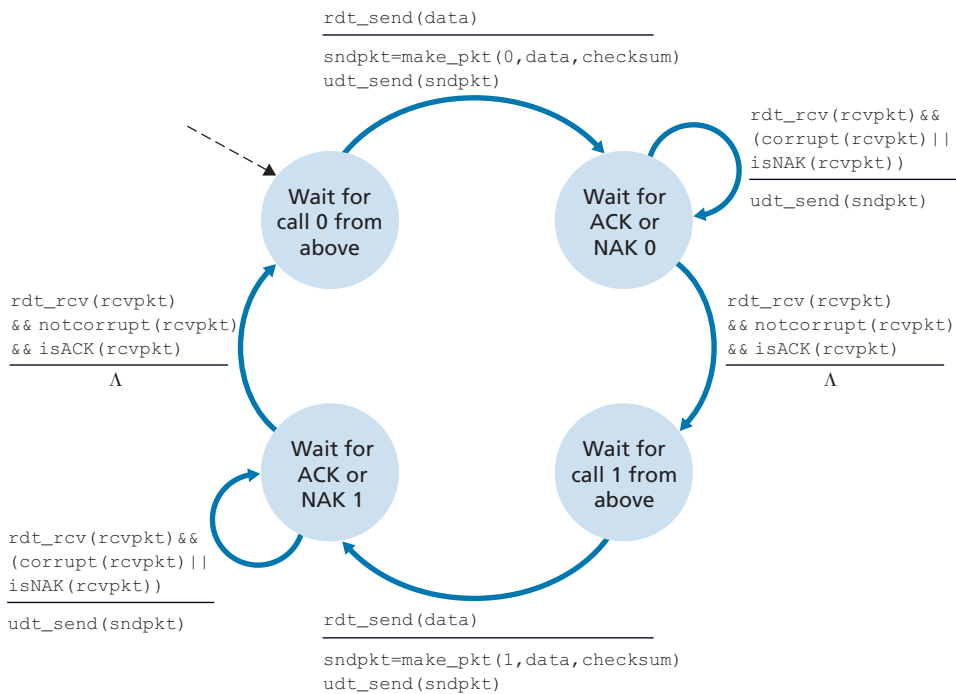


Figure 3.11 ♦ `rdt2.1` sender

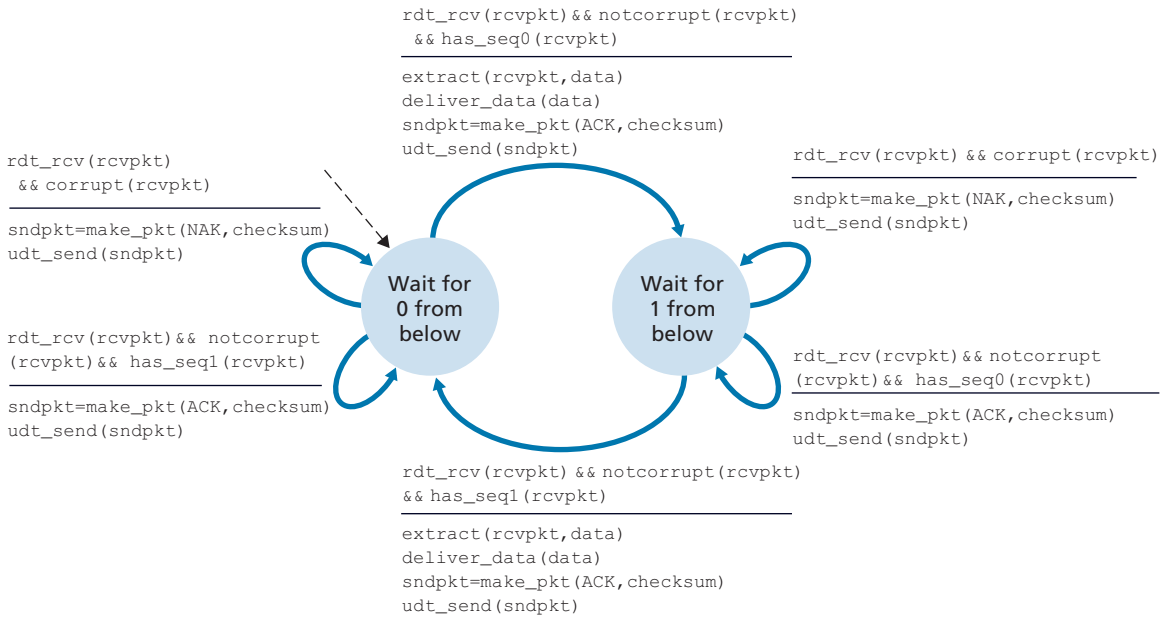


Figure 3.12 ♦ `rdt2.1` receiver

is received, the receiver sends a negative acknowledgment. We can accomplish the same effect as a NAK if, instead of sending a NAK, we send an ACK for the last correctly received packet. A sender that receives two ACKs for the same packet (that is, receives **duplicate ACKs**) knows that the receiver did not correctly receive the packet following the packet that is being ACKed twice. Our NAK-free reliable data transfer protocol for a channel with bit errors is `rdt2.2`, shown in Figures 3.13 and 3.14. One subtle change between `rdt2.1` and `rdt2.2` is that the receiver must now include the sequence number of the packet being acknowledged by an ACK message (this is done by including the `ACK, 0` or `ACK, 1` argument in `make_pkt()` in the receiver FSM), and the sender must now check the sequence number of the packet being acknowledged by a received ACK message (this is done by including the `0` or `1` argument in `isACK()` in the sender FSM).

Reliable Data Transfer over a Lossy Channel with Bit Errors: `rdt3.0`

Suppose now that in addition to corrupting bits, the underlying channel can *lose* packets as well, a not-uncommon event in today's computer networks (including the Internet). Two additional concerns must now be addressed by the protocol: how to detect packet loss and what to do when packet loss occurs. The use of checksumming, sequence numbers, ACK packets, and retransmissions—the techniques

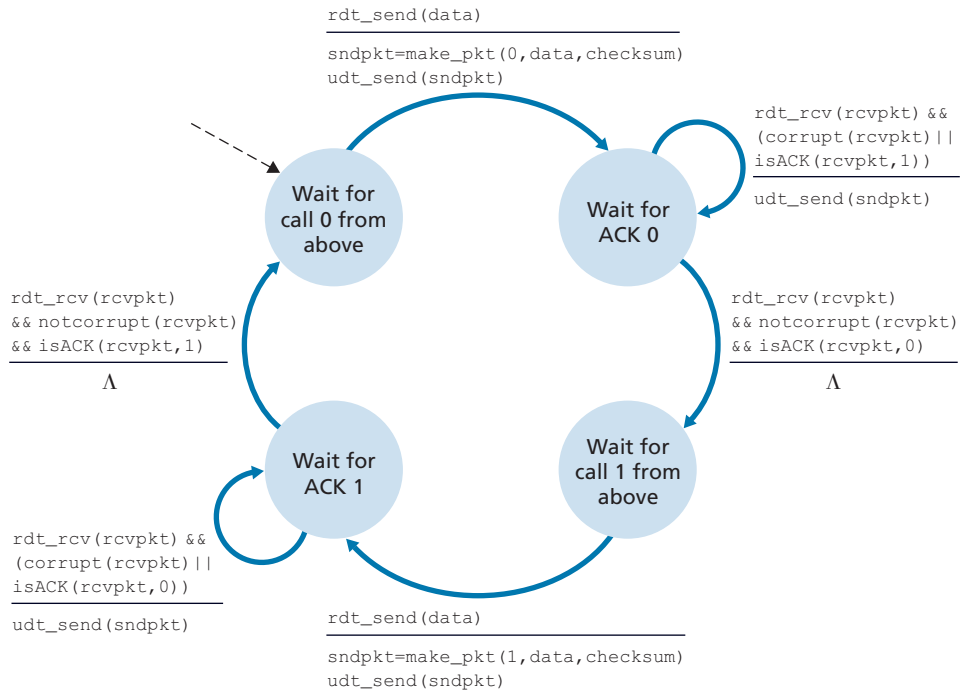


Figure 3.13 ♦ `rdt2.2` sender

already developed in `rdt2.2`—will allow us to answer the latter concern. Handling the first concern will require adding a new protocol mechanism.

There are many possible approaches toward dealing with packet loss (several more of which are explored in the exercises at the end of the chapter). Here, we'll put the burden of detecting and recovering from lost packets on the sender. Suppose that the sender transmits a data packet and either that packet, or the receiver's ACK of that packet, gets lost. In either case, no reply is forthcoming at the sender from the receiver. If the sender is willing to wait long enough so that it is *certain* that a packet has been lost, it can simply retransmit the data packet. You should convince yourself that this protocol does indeed work.

But how long must the sender wait to be certain that something has been lost? The sender must clearly wait at least as long as a round-trip delay between the sender and receiver (which may include buffering at intermediate routers) plus whatever amount of time is needed to process a packet at the receiver. In many networks, this worst-case maximum delay is very difficult even to estimate, much less know with certainty. Moreover, the protocol should ideally recover from packet loss as soon as possible; waiting for a worst-case delay could mean a long wait until error recovery

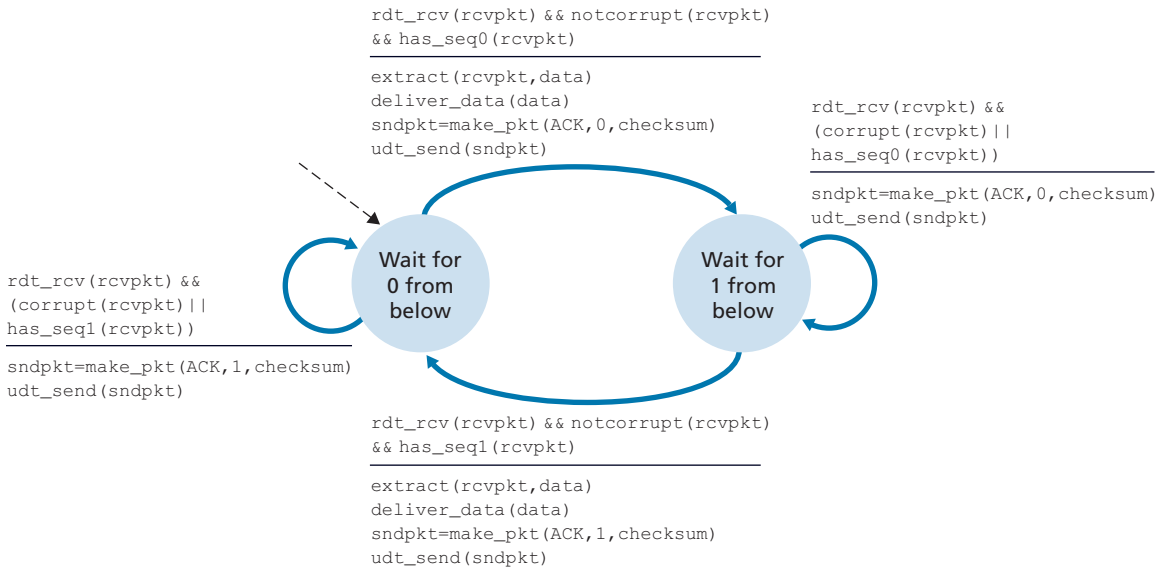


Figure 3.14 ♦ `rdt2.2` receiver

is initiated. The approach thus adopted in practice is for the sender to judiciously choose a time value such that packet loss is likely, although not guaranteed, to have happened. If an ACK is not received within this time, the packet is retransmitted. Note that if a packet experiences a particularly large delay, the sender may retransmit the packet even though neither the data packet nor its ACK have been lost. This introduces the possibility of **duplicate data packets** in the sender-to-receiver channel. Happily, protocol `rdt2.2` already has enough functionality (that is, sequence numbers) to handle the case of duplicate packets.

From the sender's viewpoint, retransmission is a panacea. The sender does not know whether a data packet was lost, an ACK was lost, or if the packet or ACK was simply overly delayed. In all cases, the action is the same: retransmit. Implementing a time-based retransmission mechanism requires a **countdown timer** that can interrupt the sender after a given amount of time has expired. The sender will thus need to be able to (1) start the timer each time a packet (either a first-time packet or a retransmission) is sent, (2) respond to a timer interrupt (taking appropriate actions), and (3) stop the timer.

Figure 3.15 shows the sender FSM for `rdt3.0`, a protocol that reliably transfers data over a channel that can corrupt or lose packets; in the homework problems, you'll be asked to provide the receiver FSM for `rdt3.0`. Figure 3.16 shows how the protocol operates with no lost or delayed packets and how it handles lost data packets. In Figure 3.16, time moves forward from the top of the diagram toward the bottom of the

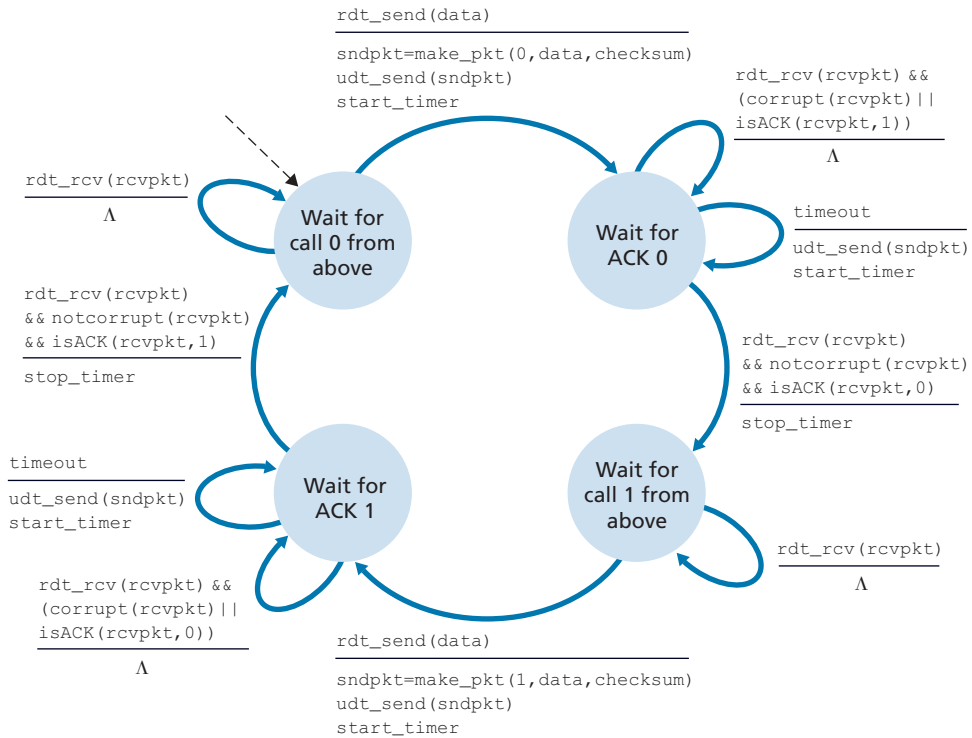


Figure 3.15 ♦ `rdt3.0` sender

diagram; note that a receive time for a packet is necessarily later than the send time for a packet as a result of transmission and propagation delays. In Figures 3.16(b)–(d), the send-side brackets indicate the times at which a timer is set and later times out. Several of the more subtle aspects of this protocol are explored in the exercises at the end of this chapter. Because packet sequence numbers alternate between 0 and 1, protocol `rdt3.0` is sometimes known as the **alternating-bit protocol**.

We have now assembled the key elements of a data transfer protocol. Checksums, sequence numbers, timers, and positive and negative acknowledgment packets each play a crucial and necessary role in the operation of the protocol. We now have a working reliable data transfer protocol!

3.4.2 Pipelined Reliable Data Transfer Protocols

Protocol `rdt3.0` is a functionally correct protocol, but it is unlikely that anyone would be happy with its performance, particularly in today's high-speed networks. At the heart of `rdt3.0`'s performance problem is the fact that it is a stop-and-wait protocol.

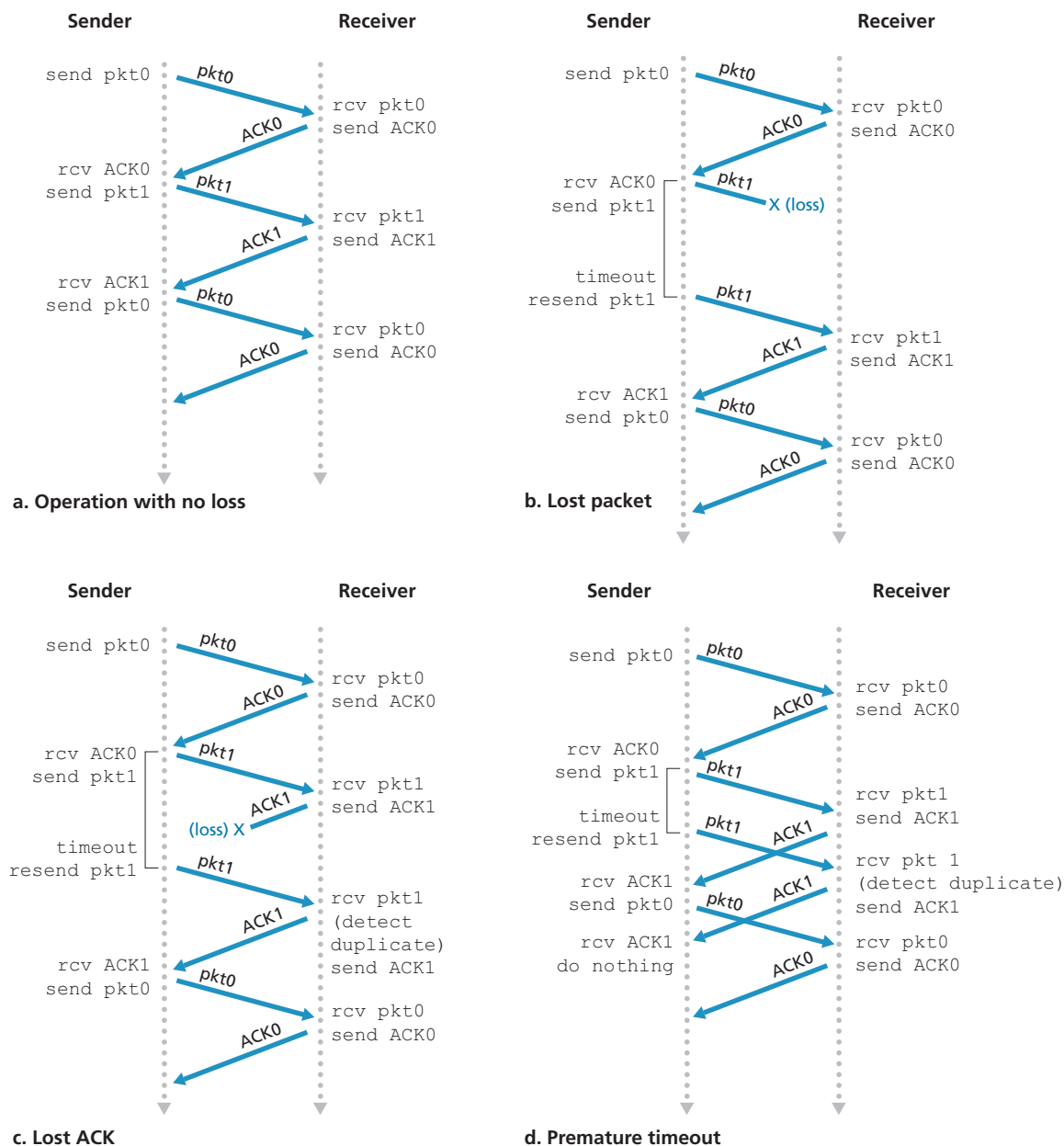


Figure 3.16 ♦ Operation of `rdt3.0`, the alternating-bit protocol