Computer Communication Networks

Transport Layer

Dr. Raja Vara Prasad
Assistant Professor
IIIT Sri City

Transport Layer

Transport Layer

how two entities can communicate reliably over a medium that may lose and corrupt data ?

controlling the transmission rate of transport-layer entities in order to avoid

Or

recover from, congestion within the network.

Transport Layer Services

- **logical communication**
- Transport-layer **segments**
- Transport-layer protocol provides logical communication between *processes* running on different hosts
- a network-layer protocol provides logical communication between *hosts*
- services that a transport protocol can provide are often constrained by the service model of the underlying network-layer protocol
- a transport protocol can offer reliable data transfer service to an application even when the underlying network protocol is unreliable
- can use encryption

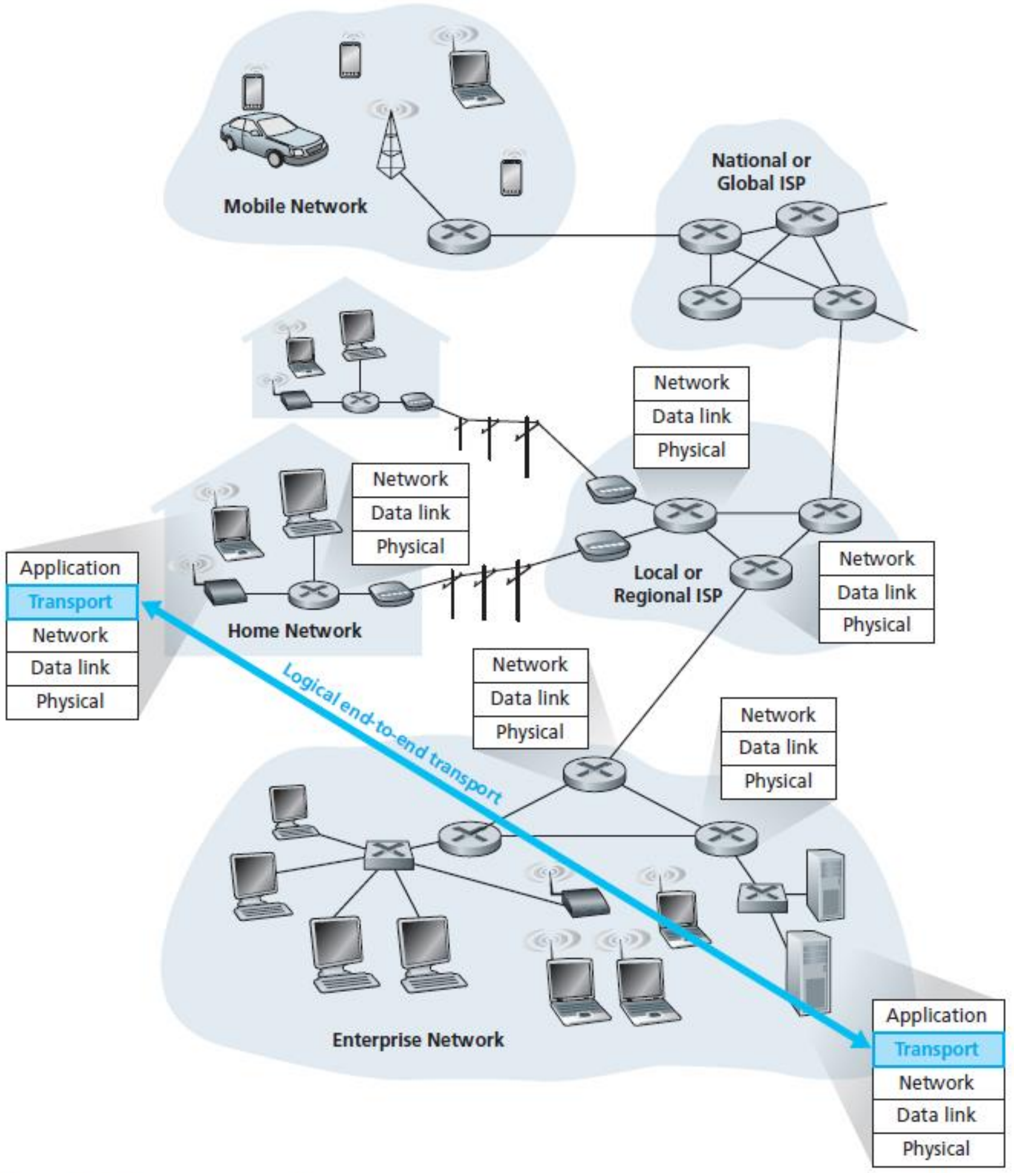


Figure 3.1 ♦ The transport layer provides logical rather than physical communication between application processes

Transport Layer in the Internet

- Internet Protocol. IP provides logical communication between hosts.
- IP service model is a **best-effort delivery service**
- “best effort” to deliver segments between communicating hosts → *makes no guarantees.*
- not guarantee segment delivery
- it does not guarantee orderly delivery of segments
- does not guarantee the integrity of the data in the segments

UDP Services:

process-to-process data delivery and error checking

TCP:

- reliable data transfer
- correct and in order → using flow control, sequence numbers, acknowledgments, and timers
- **congestion control**

Multiplexing and Demultiplexing

- host-to-host delivery service provided by the network layer
- process-to-process delivery service for applications running on the hosts – Transport Layer

- a process can have one or more **sockets**
- transport layer in the receiving host does not deliver data directly to a process → to an intermediary socket
- more than one socket in the receiving host → each socket → unique identifier
- Each transport-layer segment has a set of fields

Demultiplexing:

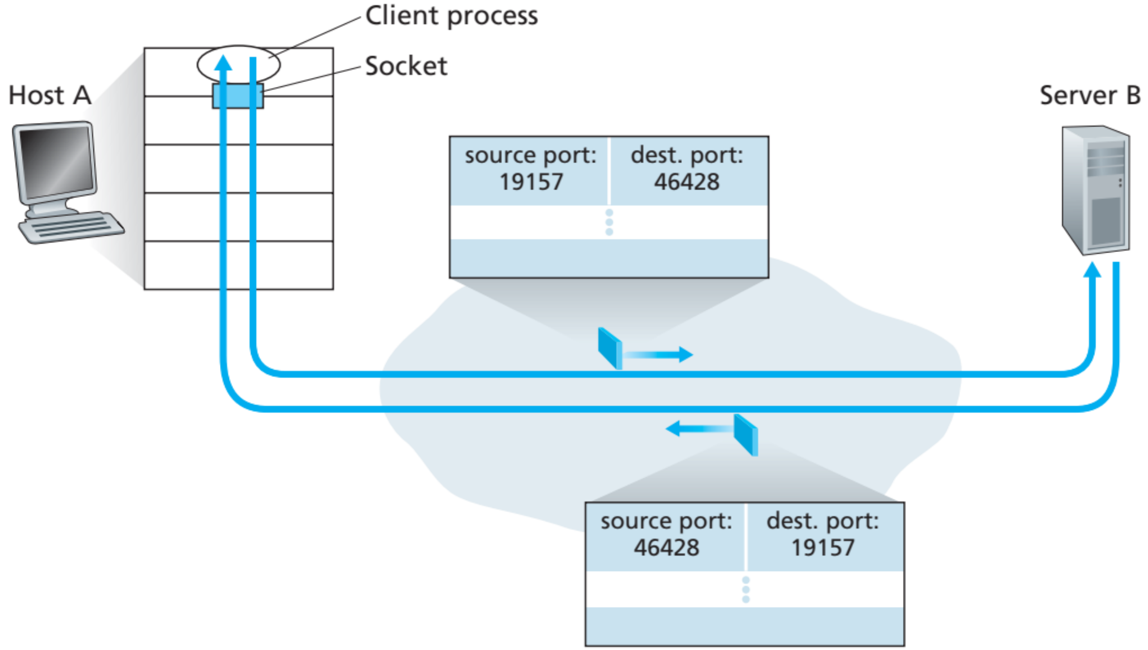
- receiving end → the transport layer examines these fields to identify the receiving socket
- directs the segment to that socket
- **Multiplexing**
  gathering data chunks at the source host from different sockets
- encapsulating each data chunk with header information to create segments
- passing the segments to the network layer is called

Connectionless Multiplexing and Demultiplexing

UDP socket:

- transport layer assigns a port number in the range 1024 to 65535 that is currently not being used by any other UDP port in the host

  Ex: A process in Host A, with UDP port 19157 → to send a chunk of application data to a process with UDP port 46428 in Host B.
- UDP socket: identified by a two-tuple → a destination IP address and a destination port number



if two UDP segments have different source IP addresses and/or source port numbers, but have the same *destination* IP address and *destination* port number ?

Connection Oriented Multiplexing and Demultiplexing

TCP socket:

- TCP socket is identified by a four-tuple
  source IP , source port number, destination IP, destination port number
- host uses all four values to direct the segment to the appropriate socket

server host may support many simultaneous TCP connection sockets, with
each socket attached to a process, and with each socket identified by its own four tuple.

Web Servers and TCP:

---all segments will have destination port 80.

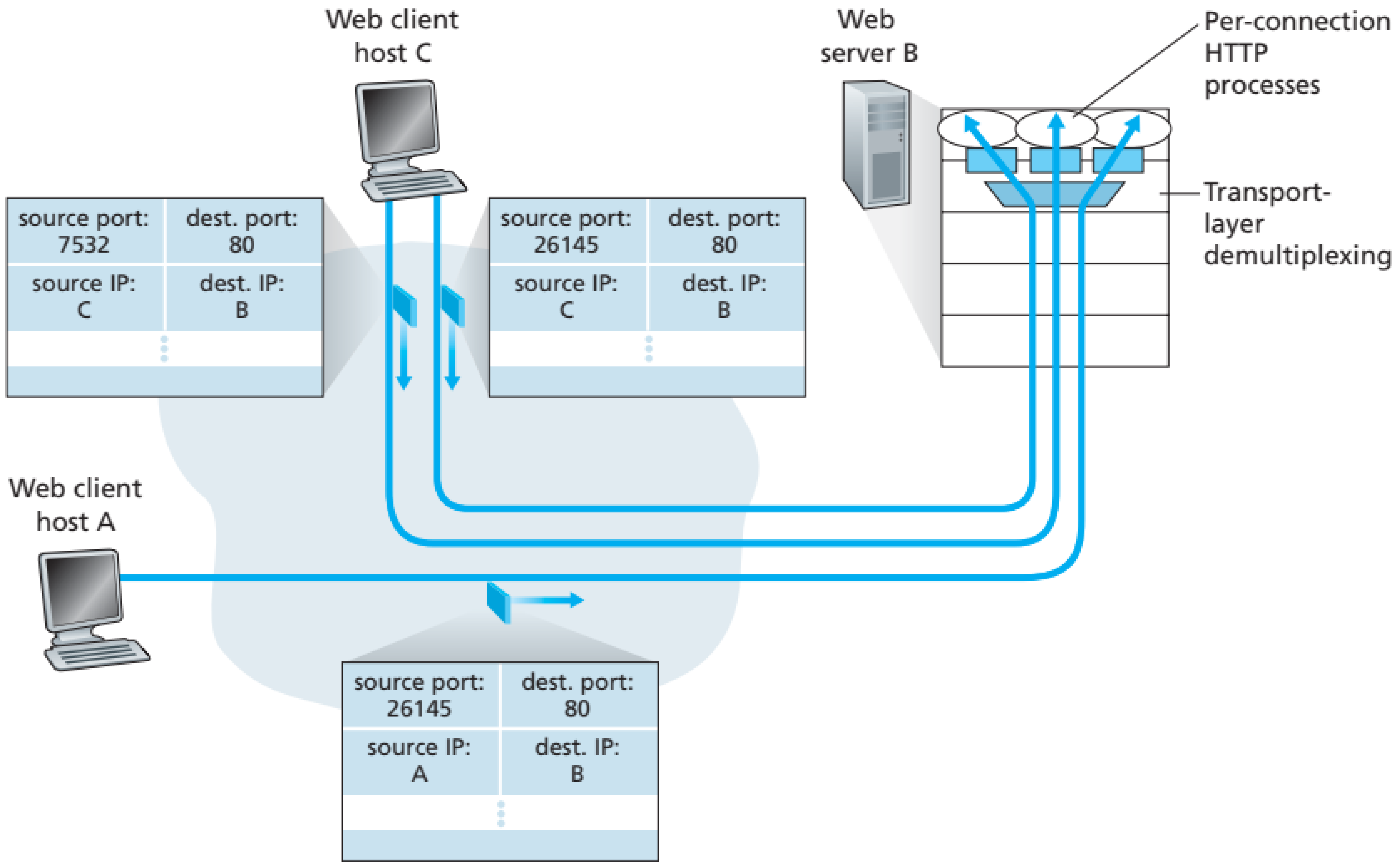
---Web servers often use only one process, and create a new thread with a new connection socket for each new client connection .

---client and server using persistent HTTP → same server socket

---non-persistent HTTP → a new TCP connection is created and closed for every request/response

---frequent creating and closing of sockets --- severely impact the performance of a busy Web server

Connection Oriented Multiplexing and Demultiplexing



Connectionless - UDP

- UDP → no handshaking between sending and receiving transport-layer →UDP is said to be *connectionless*
  Example: DNS → a query → DNS query message and passes the message to UDP

- many applications are better suited for UDP for the following reasons:
- ***Finer application-level control over what data is sent, and when***
  →TCP will also continue to resend a segment until the receipt of the segment has been acknowledged by the destination, regardless of how long reliable delivery takes → real-time applications
- ***No connection establishment***
- ***No connection state*** **: Connection state** includes receive and send buffers, congestion-control parameters, and sequence and acknowledgment number parameters
  "can typically support many more active clients when the application runs over UDP rather than TCP"
- ***Small packet header overhead*** : TCP segment has 20 bytes of header : UDP: 8 bytes

Connectionless - UDP

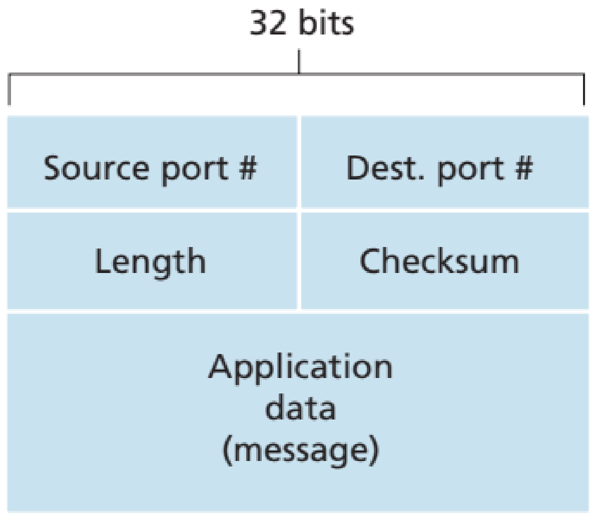
- UDP is used for RIP routing table updates
- carry network management data

Multimedia applications, such as Internet phone, real-time video conferencing, and streaming of stored audio and video.

- Internet phone and video conferencing, react very poorly to TCP's congestion control
- When packet loss rates are low and some organizations blocking UDP traffic for security reasons TCP becomes an increasingly attractive protocol for streaming media transport.
- lack of congestion control in UDP can result in high loss rates between a UDP sender and receiver, and the crowding out of TCP sessions

Application	Application-Layer Protocol	Underlying Transport Protocol
Electronic mail	SMTP	TCP
Remote terminal access	Telnet	TCP
Web	HTTP	TCP
File transfer	FTP	TCP
Remote file server	NFS	Typically UDP
Streaming multimedia	typically proprietary	UDP or TCP
Internet telephony	typically proprietary	UDP or TCP
Network management	SNMP	Typically UDP
Routing protocol	RIP	Typically UDP
Name translation	DNS	Typically UDP

Connectionless - UDP



- **UDP Segment Structure**
- **UDP Checksum**

  provides for error detection

0110011001100000
0101010101010101
1000111100001100

The sum of first two of these 16-bit words is

0110011001100000
<u>0101010101010101</u>
1011101110110101

Adding the third word to the above sum gives

1011101110110101
<u>1000111100001100</u>
0100101011000010

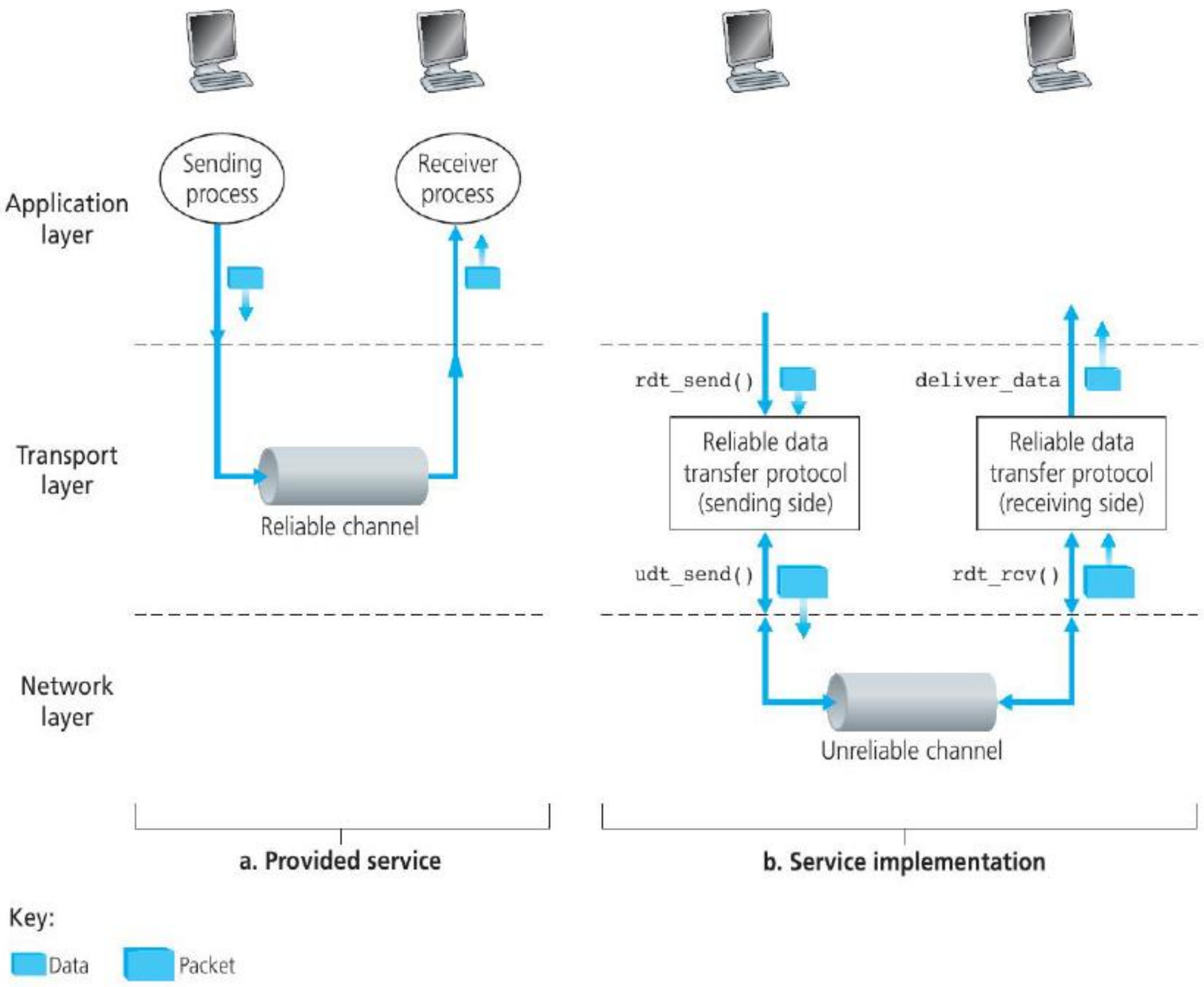
I's complement 1011010100111101

At the receiver, all four 16-bit words are added, including the checksum.
If no errors are introduced into the packet -- 1111111111111111
lin klayer protocols also provide error checking. The why UDP again ?

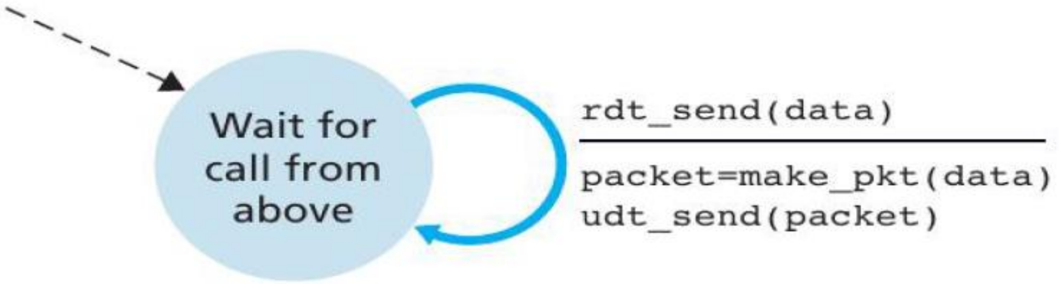
provides error checking, it does not do anything to recover from an error

Principles of Reliable Data Transfer

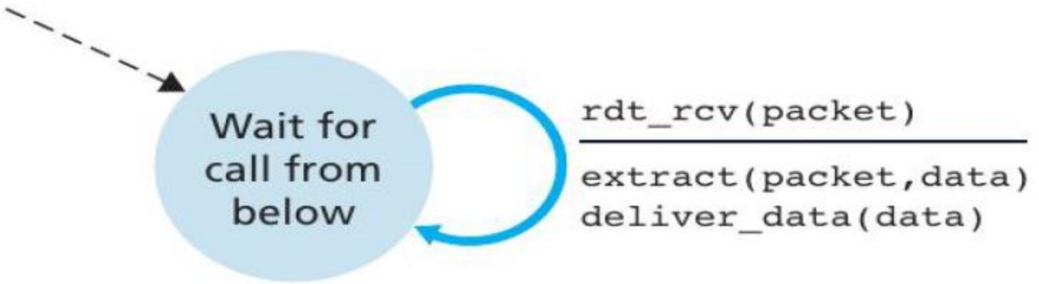
Reliable Data Transfer



RDT1.0: Perfectly Reliable Channel

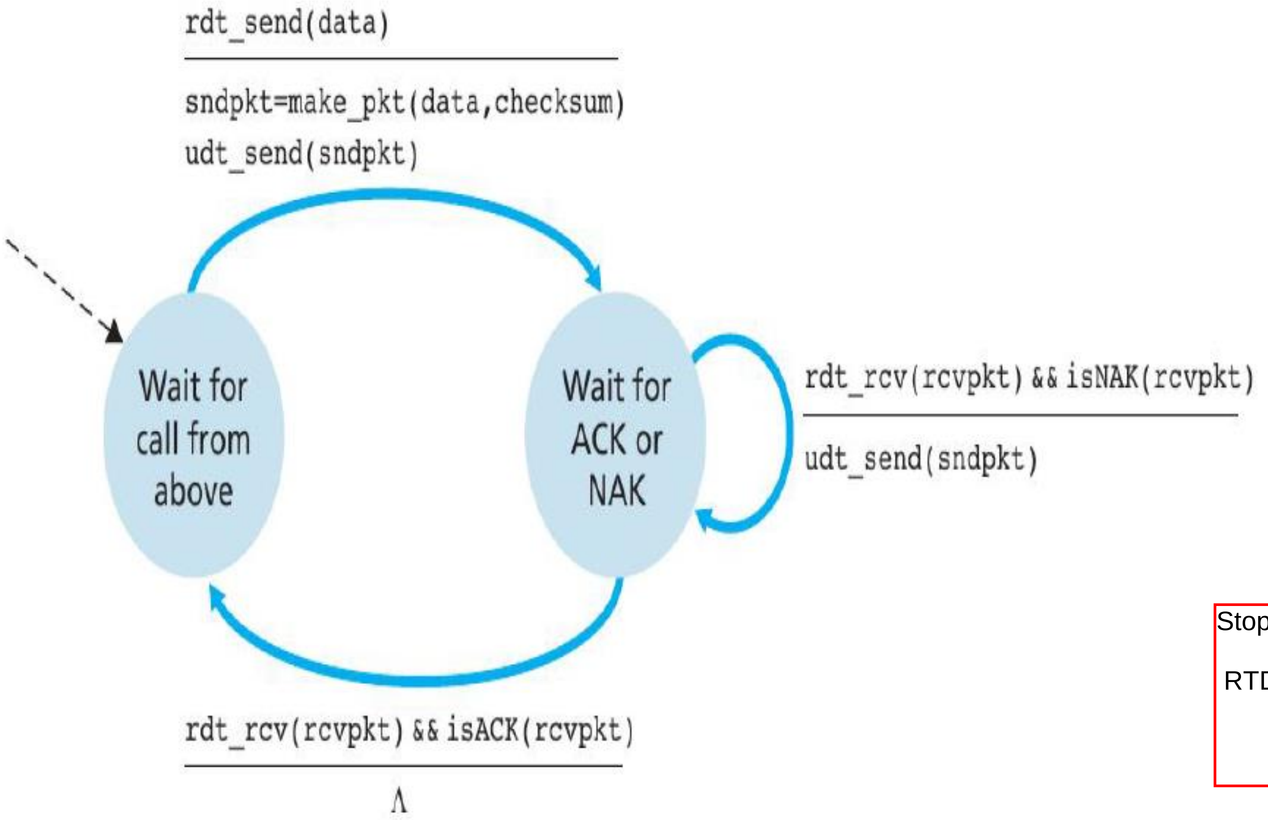


a. rdt1.0: sending side



b. rdt1.0: receiving side

RDT Over a Channel with Bit Errors: rdt 2.0 sender



a. rdt2.0: sending side

Positive Acknowledgements

Negative Acknowledgements

Automatic Repeat Request (ARQ)

Error Detection

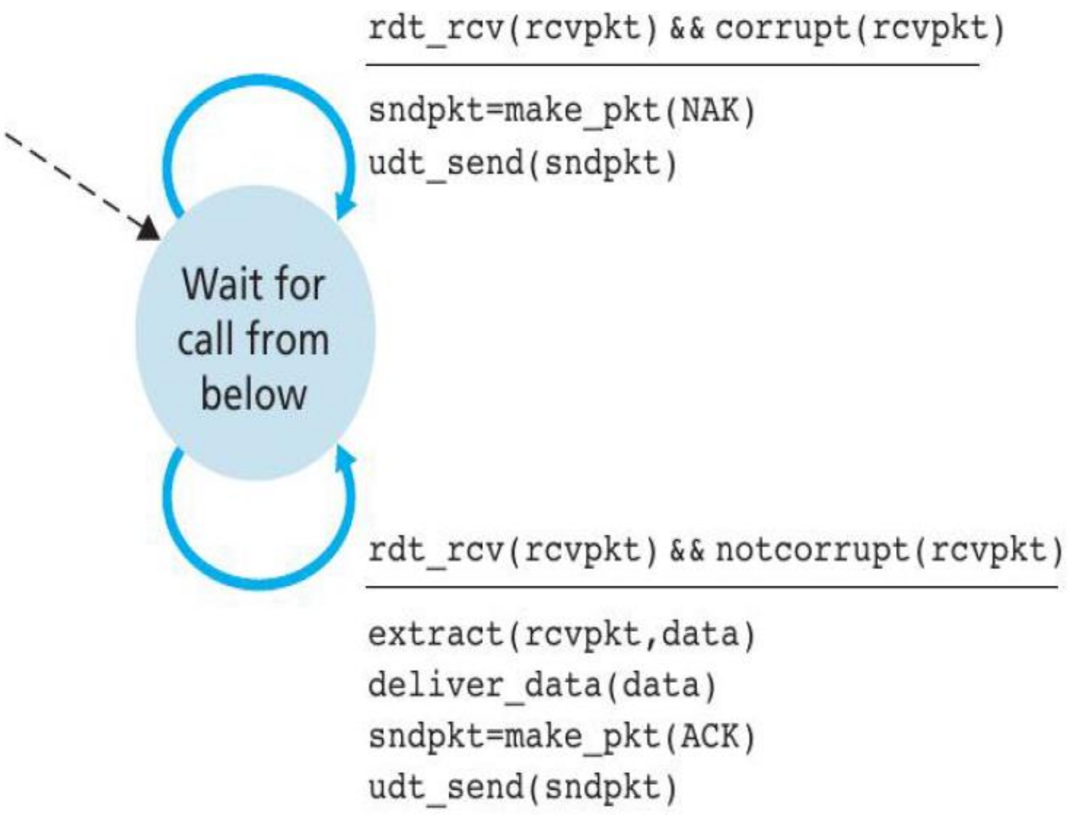
Receiver Feedback

Retransmission

Stop and Wait Protocols

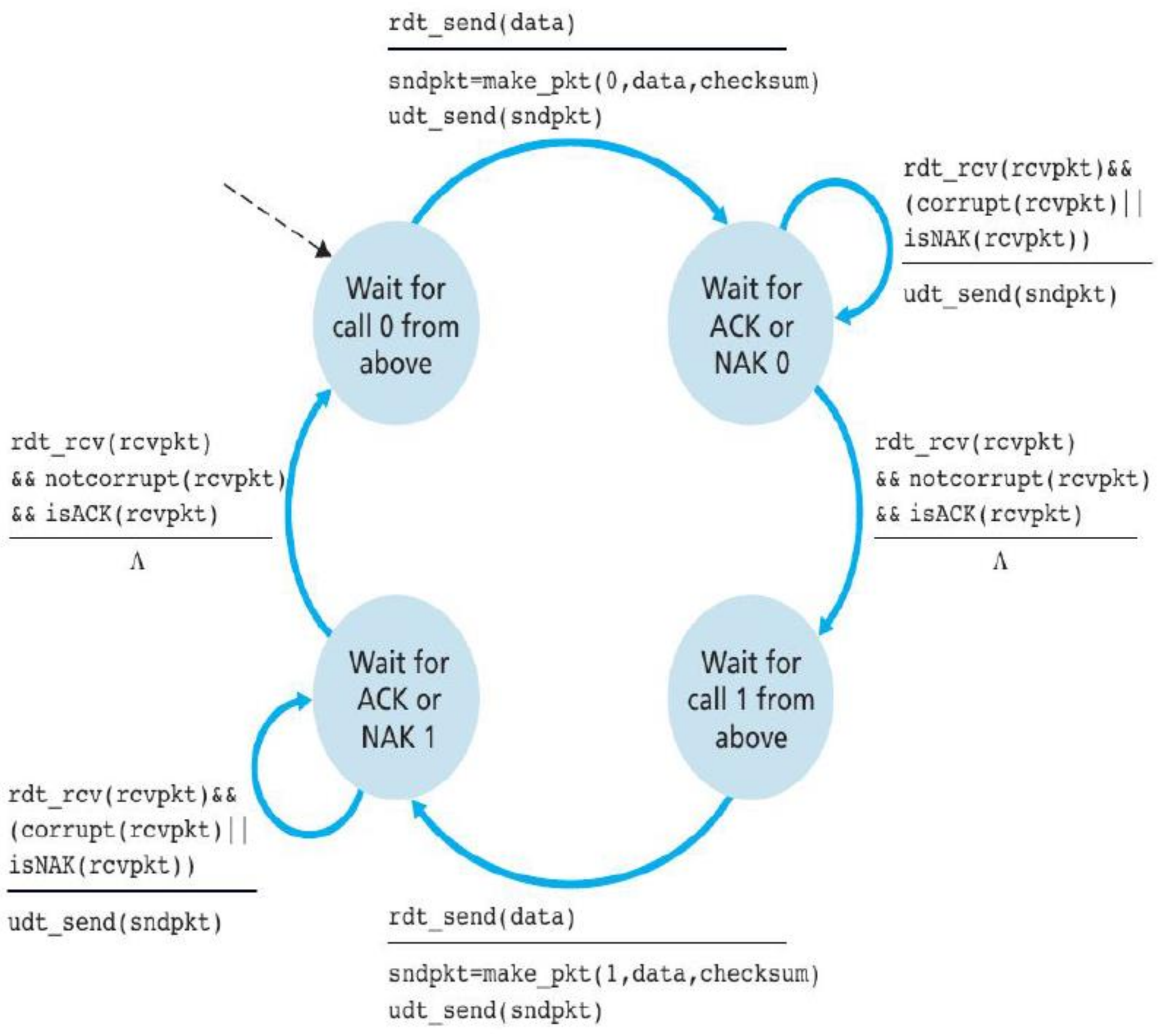
RTD 2.0 assumes -- No corruption in ACK and NAK
--- Consecutive ACK and NAK corruption
--- Checksum bits for ACK
--- Sender resend last data packet with corrupted ACK/NAK

RDT Over a Channel with Bit Errors: rdt 2.0 receiver

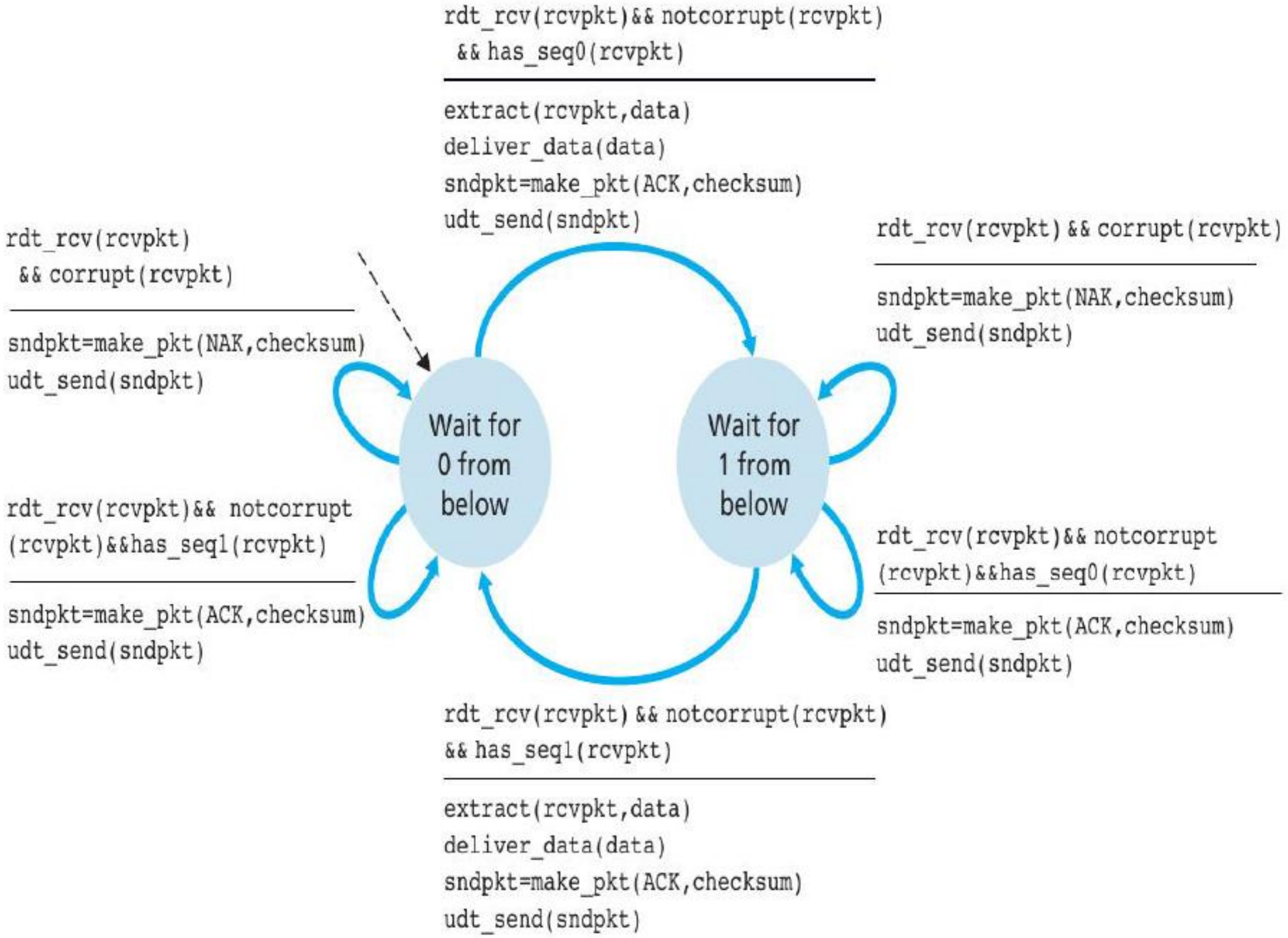


b. rdt2.0: receiving side

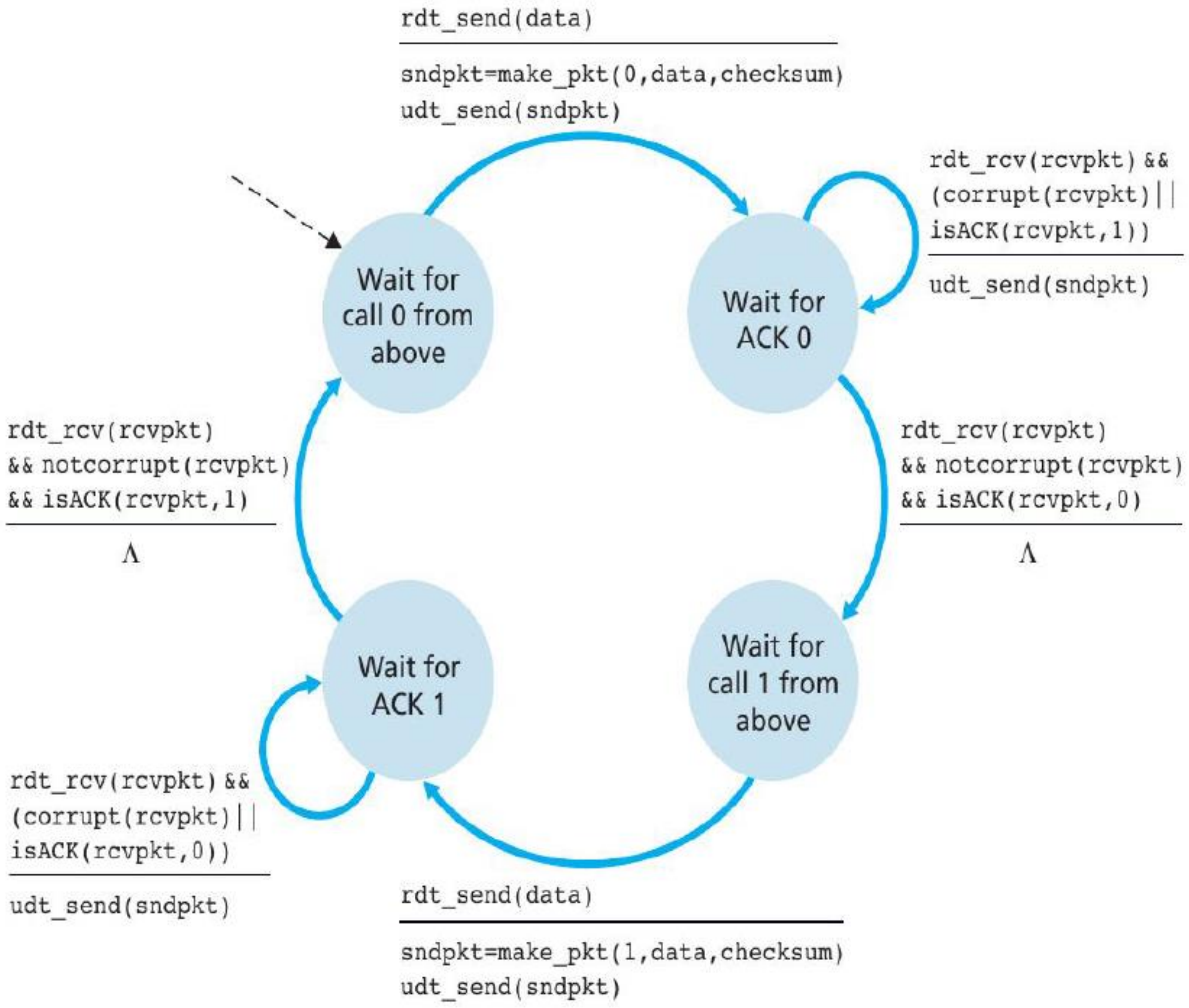
RDT 2.1 Sender



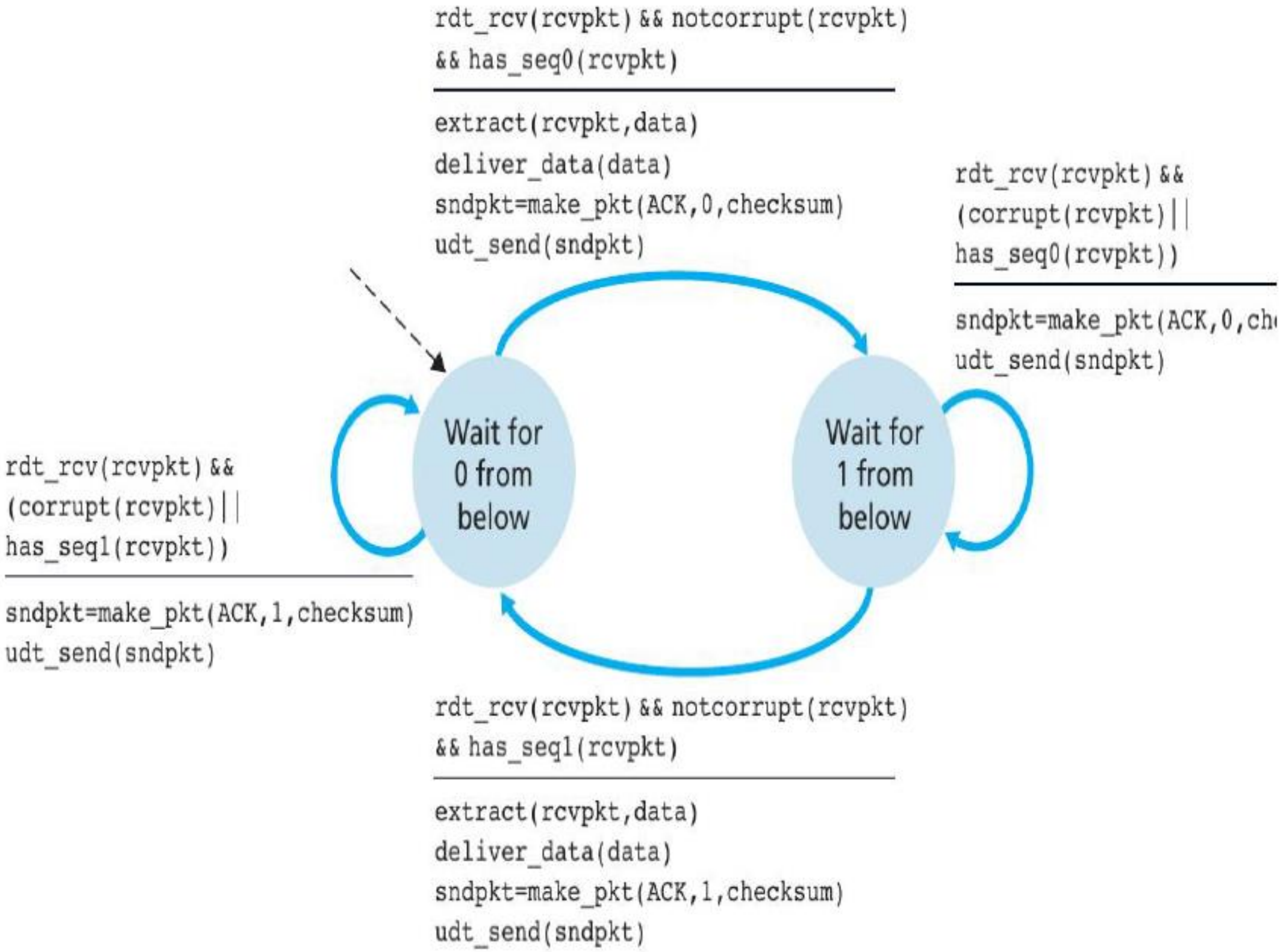
RDT 2.1 Receiver



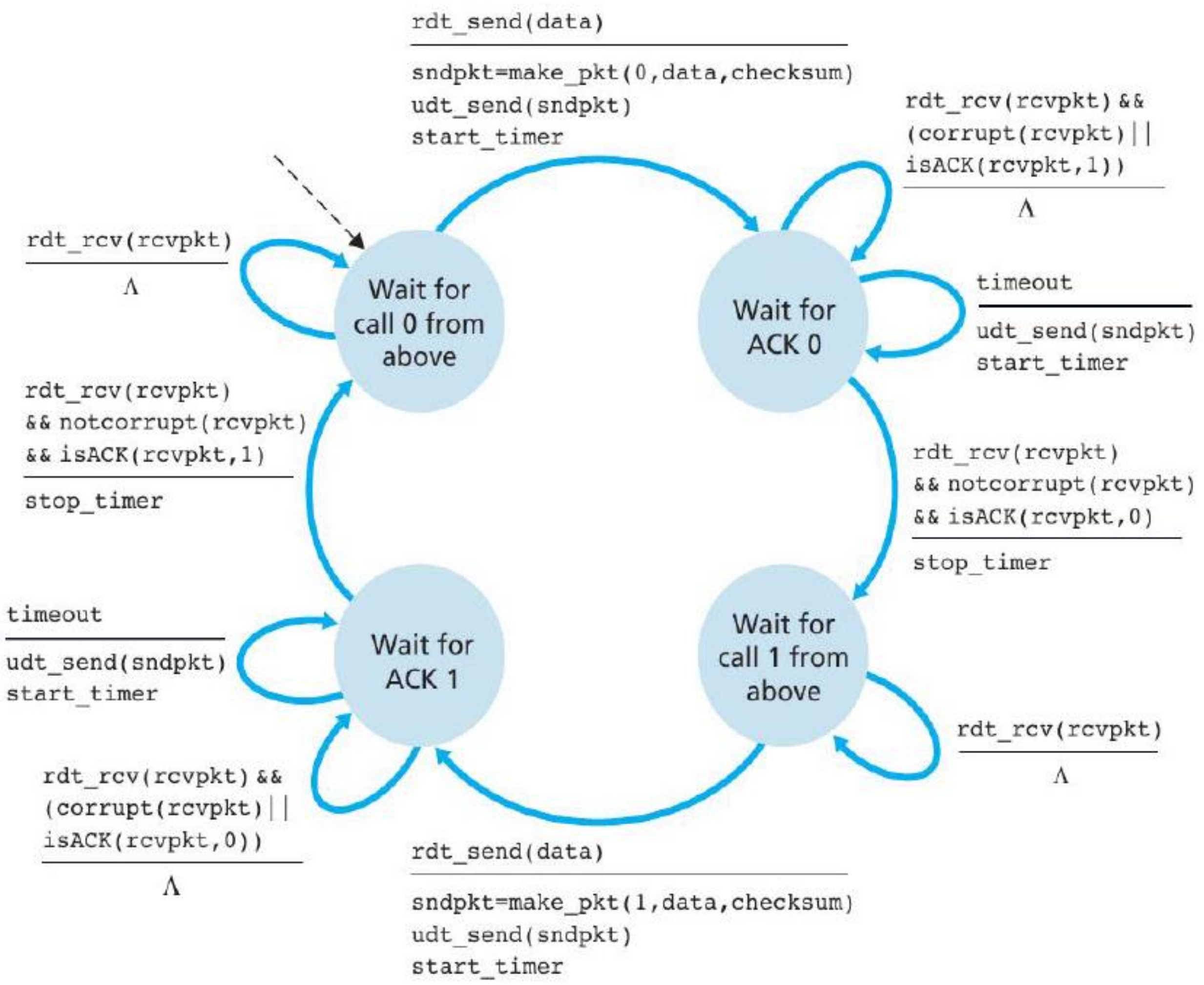
RDT Over a Lossy Channel with Bit Errors: rdt 2.2 sender



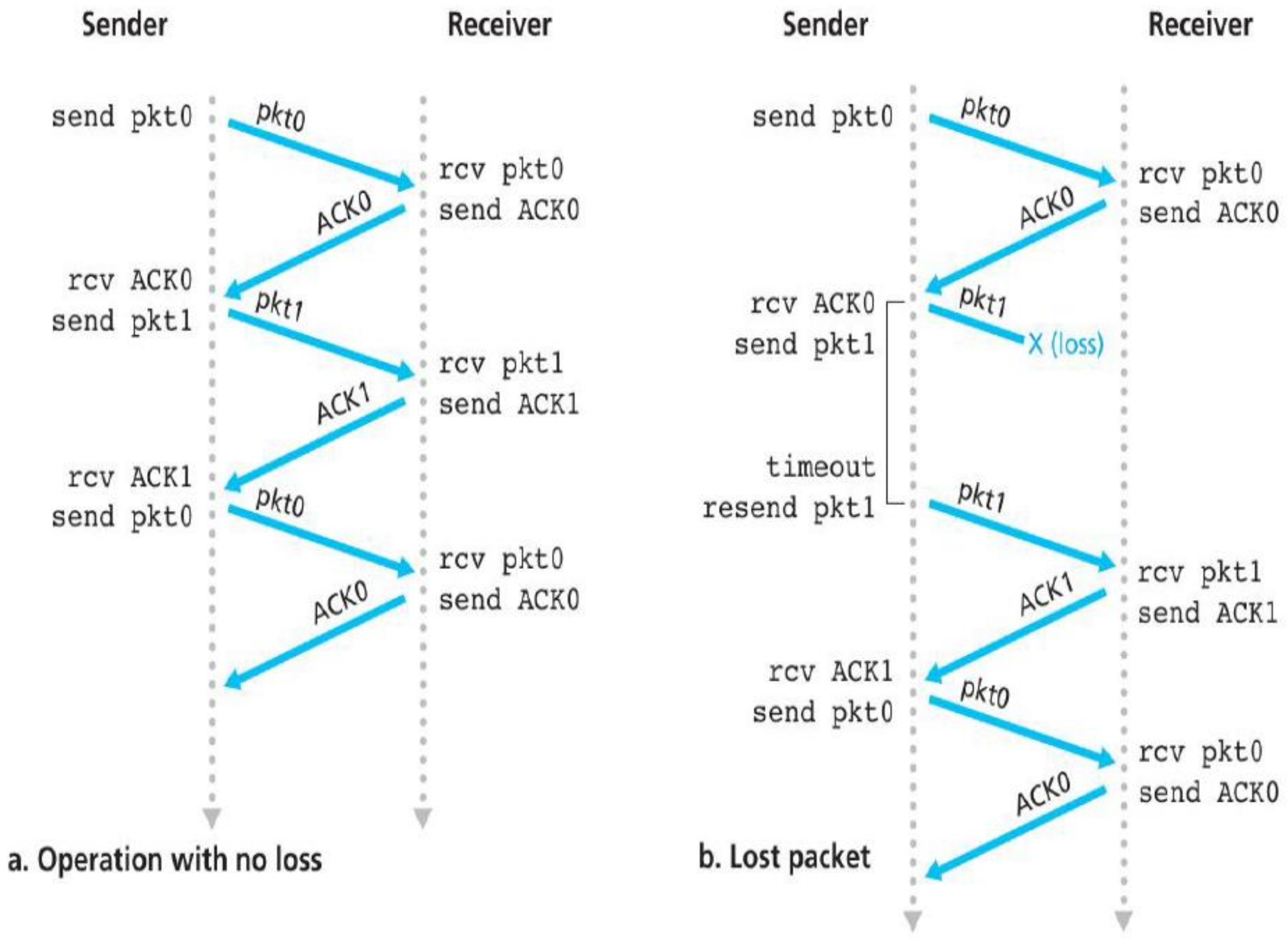
RDT Over a Lossy Channel with Bit Errors: rdt 2.2 receiver



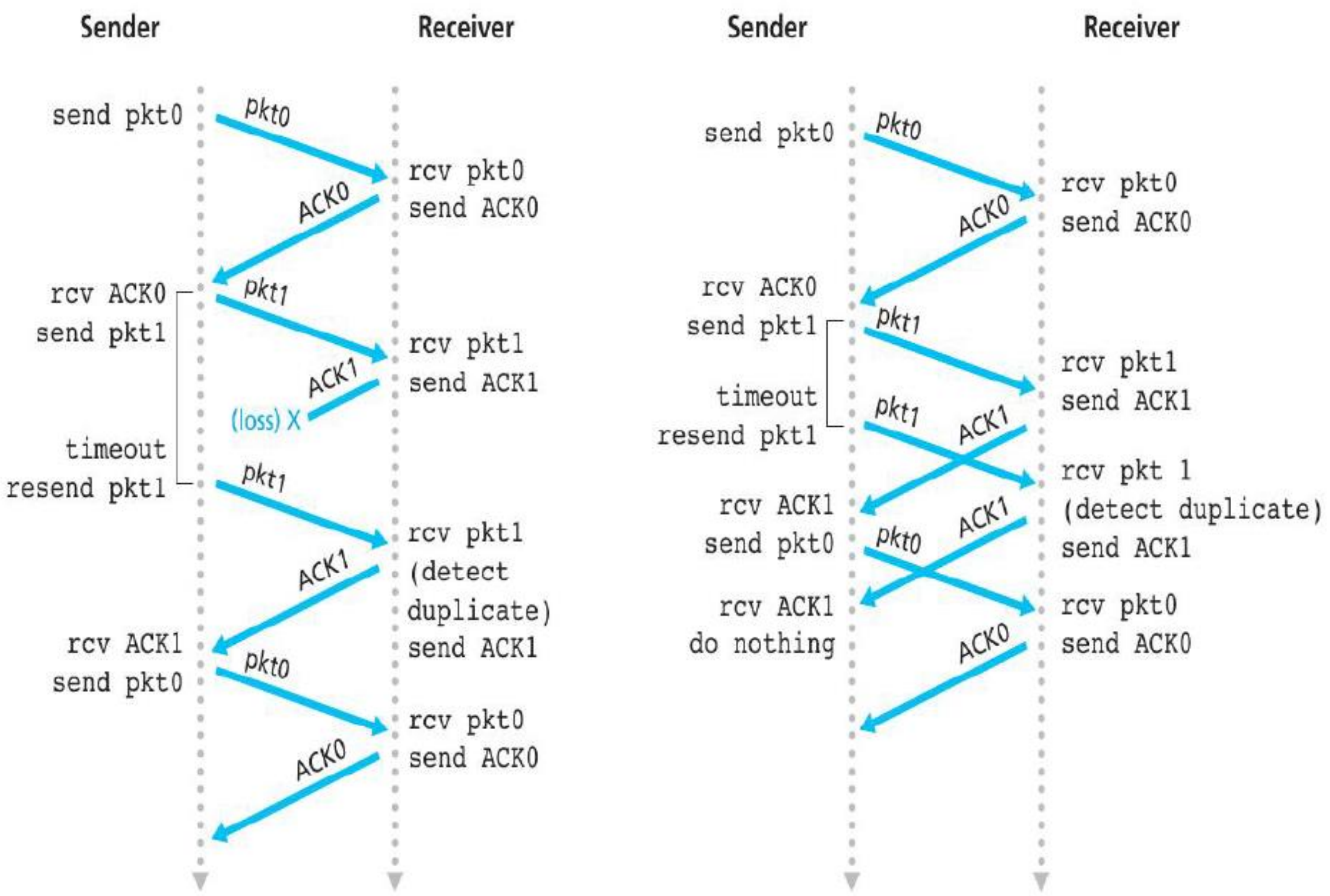
RDT 3.0: NAK-Free



RDT 3.0-Alternating-bit Protocol: Operation



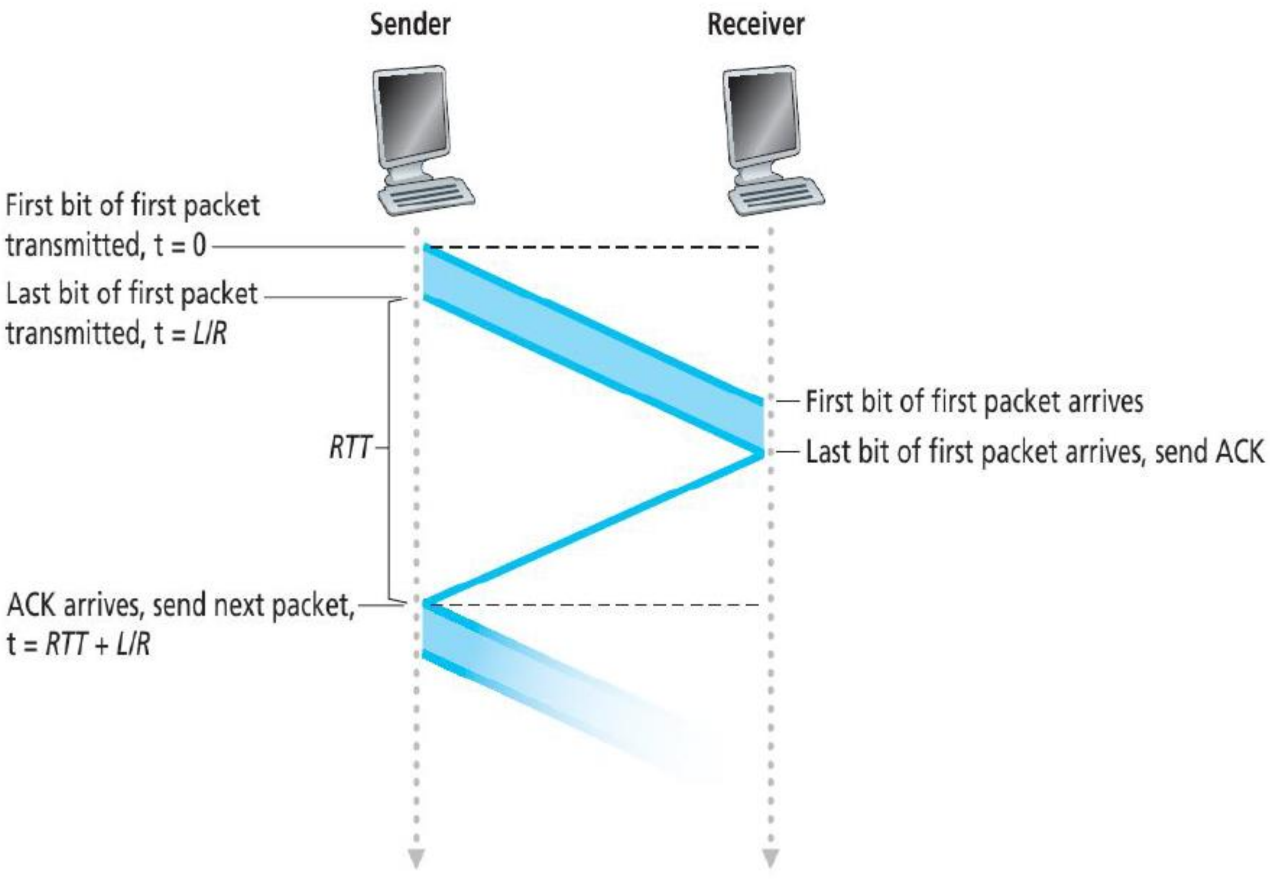
RDT 3.0-Alternating-bit Protocol: Operation



c. Lost ACK

d. Premature timeout

Stop-and-Wait Operation



a. Stop-and-wait operation

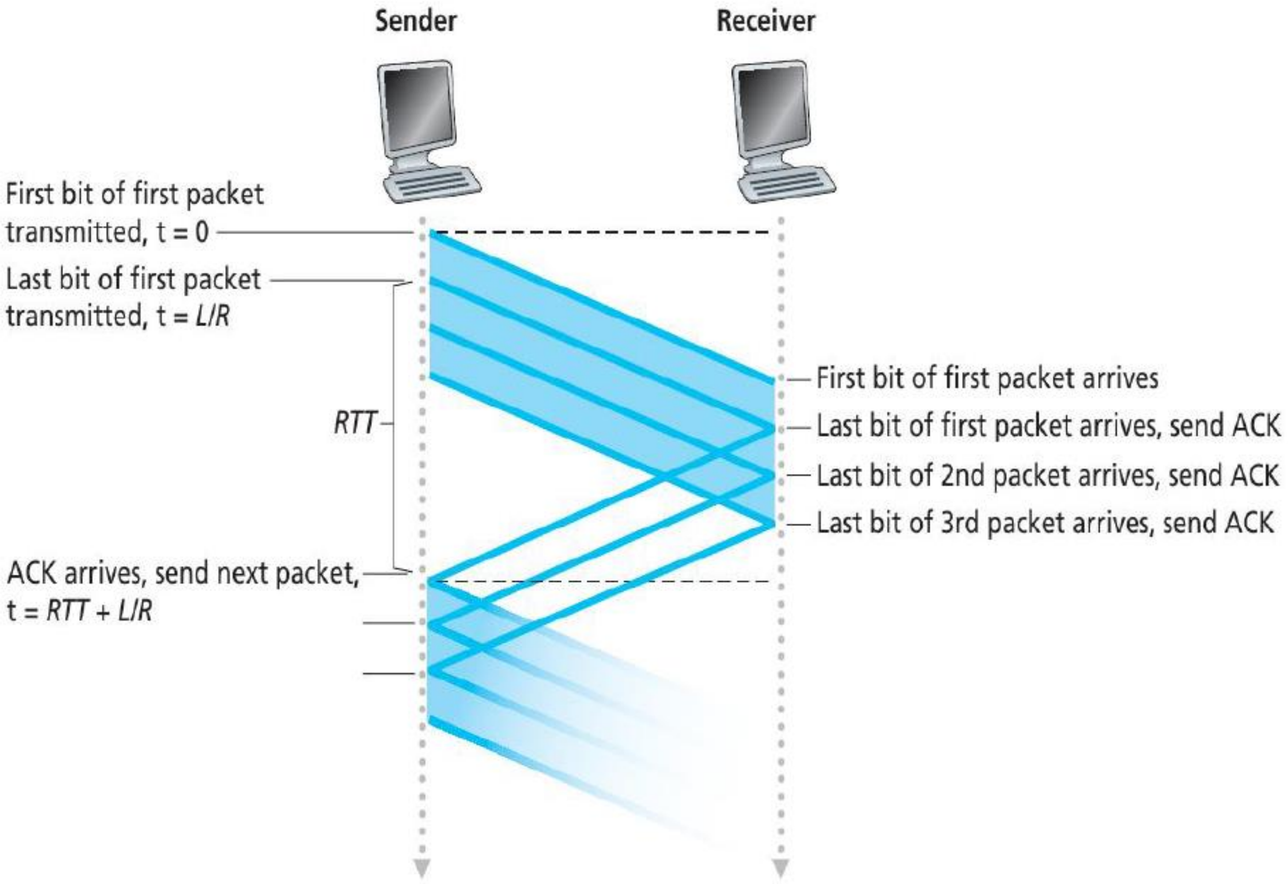
round-trip propagation delay between these two end systems, RTT, is approximately 30ms. connected by a channel with a transmission rate, *R*, of 1 Gbps

packet size *L* of 1,000 bytes

D_td = ?

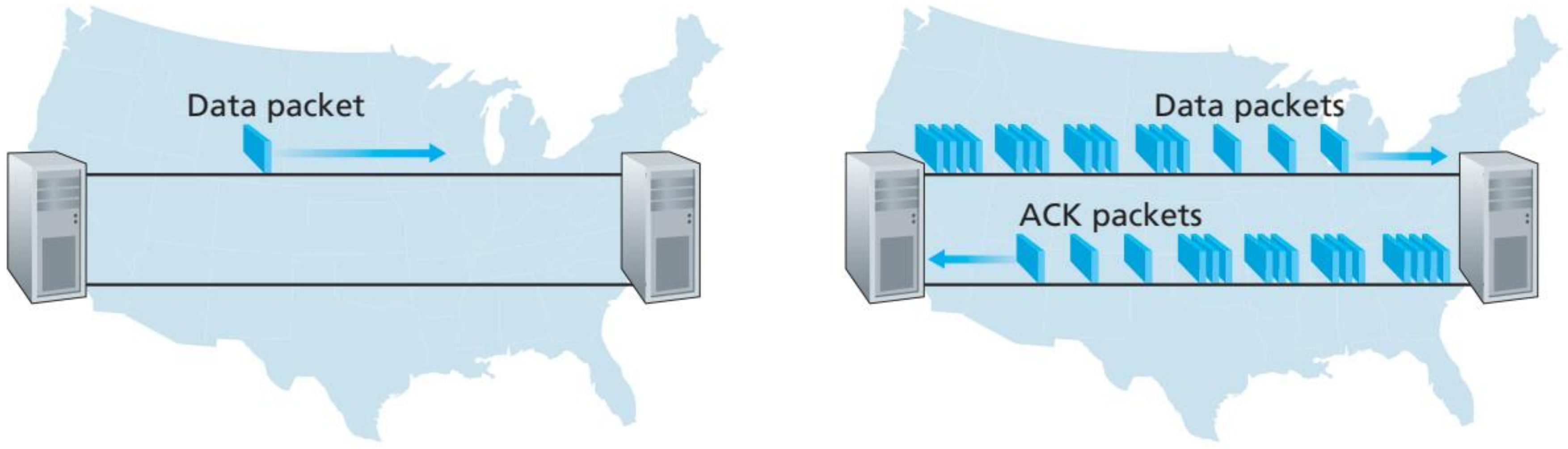
sender utilization U_{sender} ?

Pipelining



b. Pipelined operation

Pipelining



a. A stop-and-wait protocol in operation

b. A pipelined protocol in operation

The range of sequence numbers must be increased: each in-transit packet must have a unique sequence number

- sender and receiver sides of the protocols may have to buffer more than one packet.
- The range of sequence numbers needed and the buffering requirements will depend on the manner in which a data transfer protocol responds to lost, corrupted, and overly delayed packets.

→ Go-Back-N

→ Selective Repeat(SR)

GBN

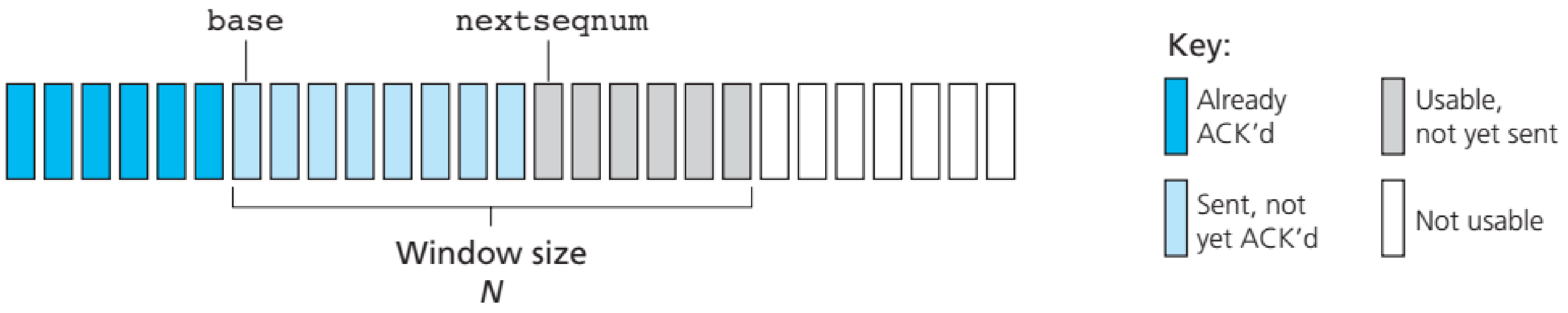
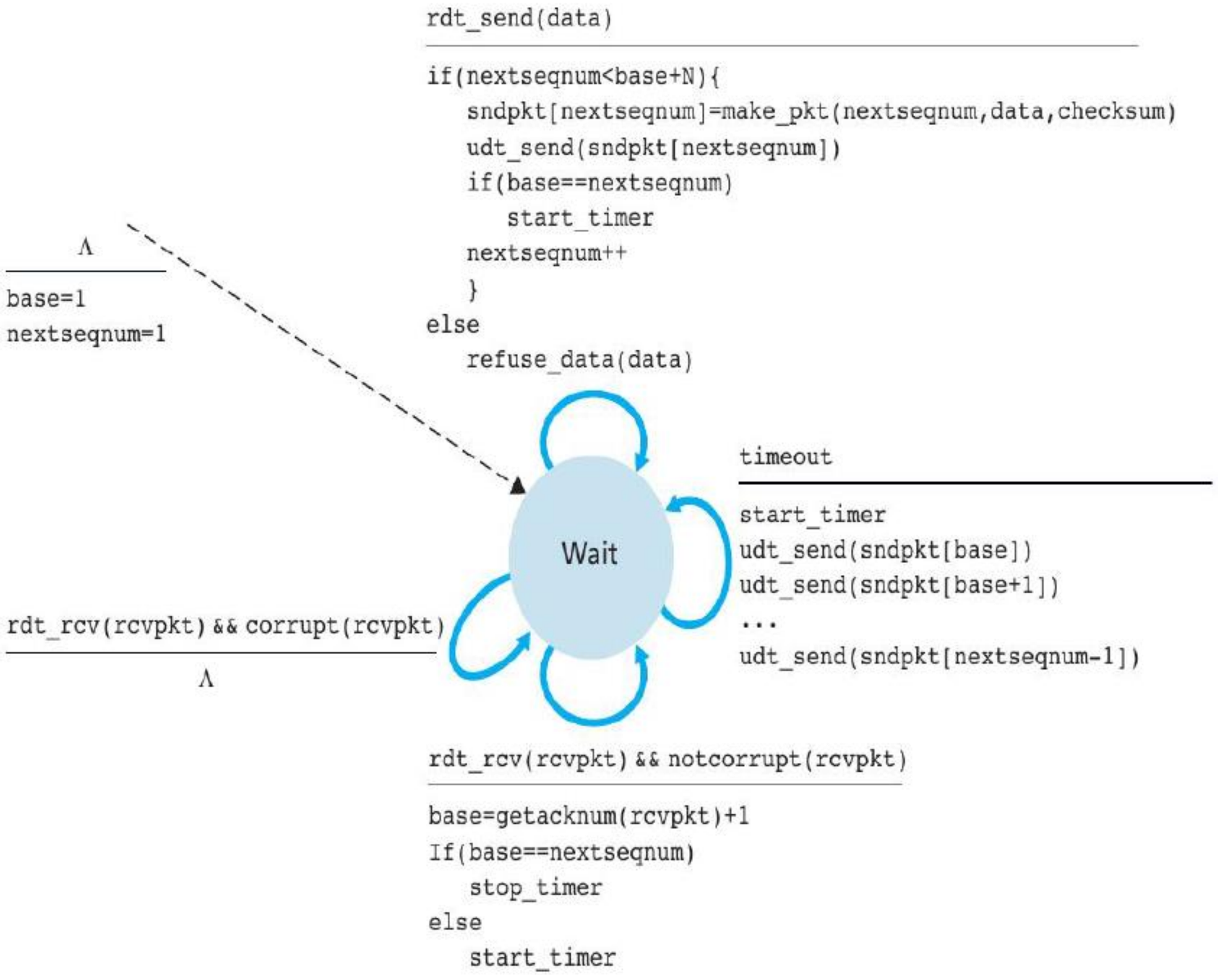


Figure 3.19 ♦ Sender's view of sequence numbers in Go-Back-N

k is the number of bits in the packet sequence number field, range of sequence numbers is $[0, 2^K - 1]$

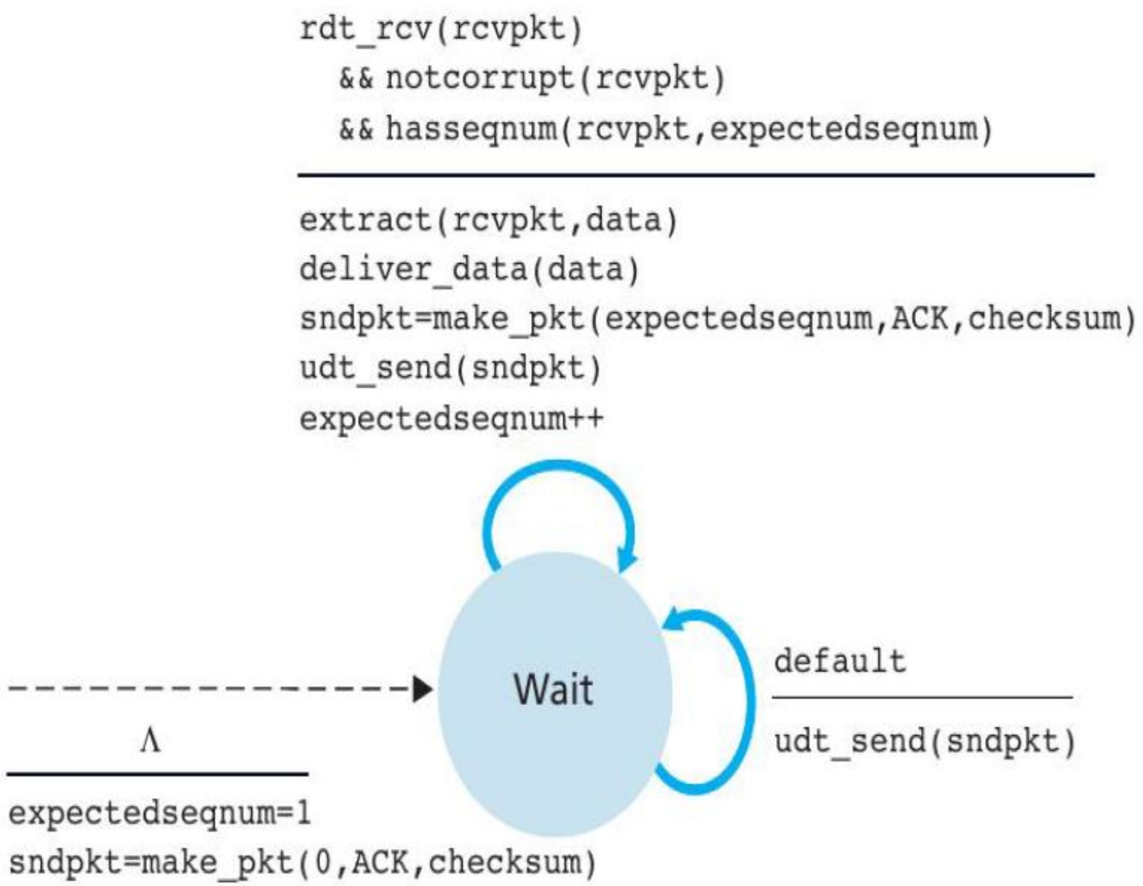
- Sequence numbers in the interval [0,base-1] → transmitted and acknowledged.
- The interval [base,nextseqnum-1] → sent but not yet acknowledged.
- Interval [nextseqnum,base+N-1] → packets that can be sent immediately; data from the upper layer.
- Greater than or equal to base+N cannot be used until an unacknowledged packet currently in the pipeline

GBN Sender

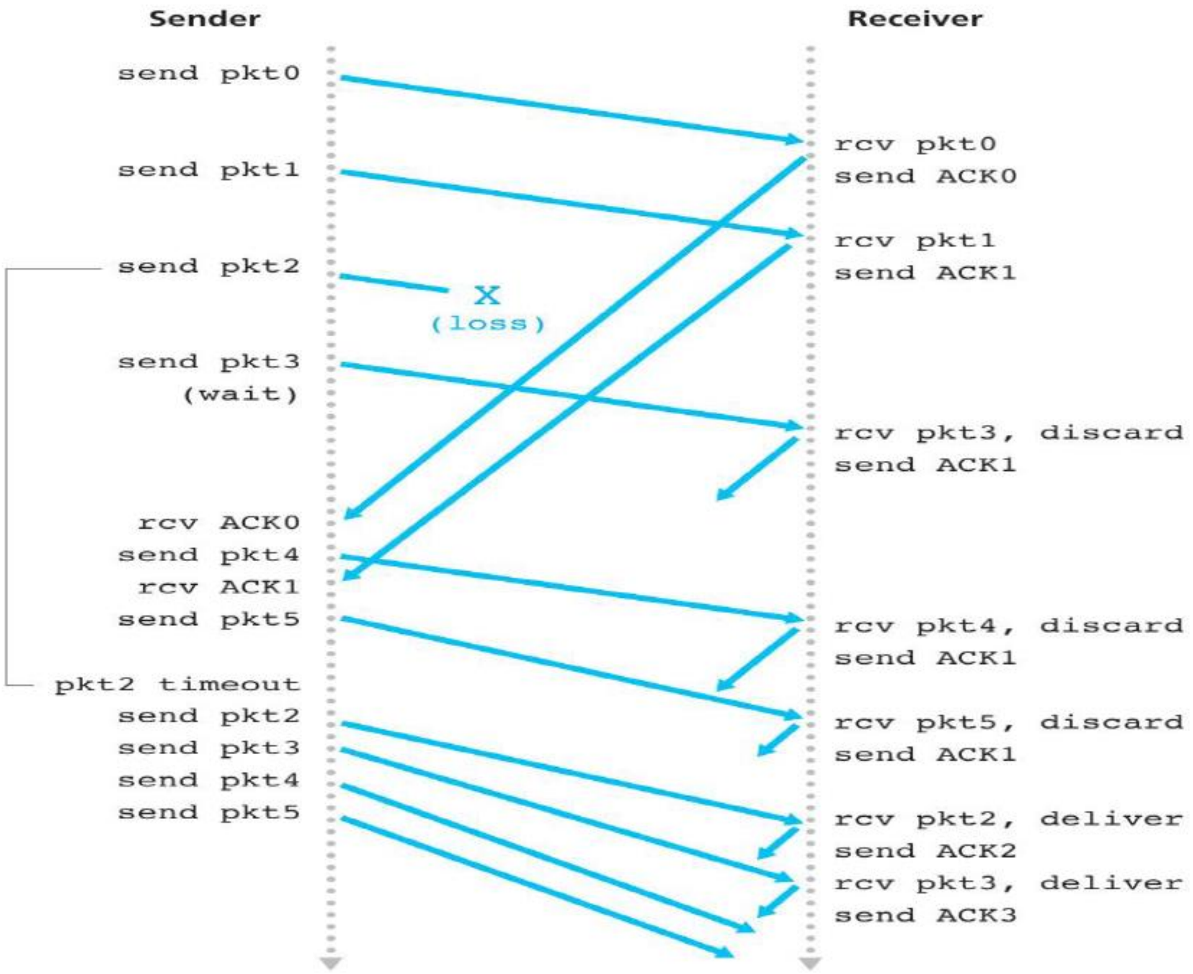


rdt_send(data)
if(nextseqnum<base+N){
sndpkt[nextseqnum]=make_pkt(nextseqnum,data,checksum)
udt_send(sndpkt[nextseqnum])
if(base==nextseqnum)
start_timer
nextseqnum++
}
else
refuse_data(data)
Λ
base=1
nextseqnum=1
timeout
start_timer
udt_send(sndpkt[base])
udt_send(sndpkt[base+1])
...
udt_send(sndpkt[nextseqnum-1])
Wait
rdt_rcv(rcvpkt) && corrupt(rcvpkt)
Λ
rdt_rcv(rcvpkt) && notcorrupt(rcvpkt)
base=getacknum(rcvpkt)+1
If(base==nextseqnum)
stop_timer
else
start_timer

GBN Receiver



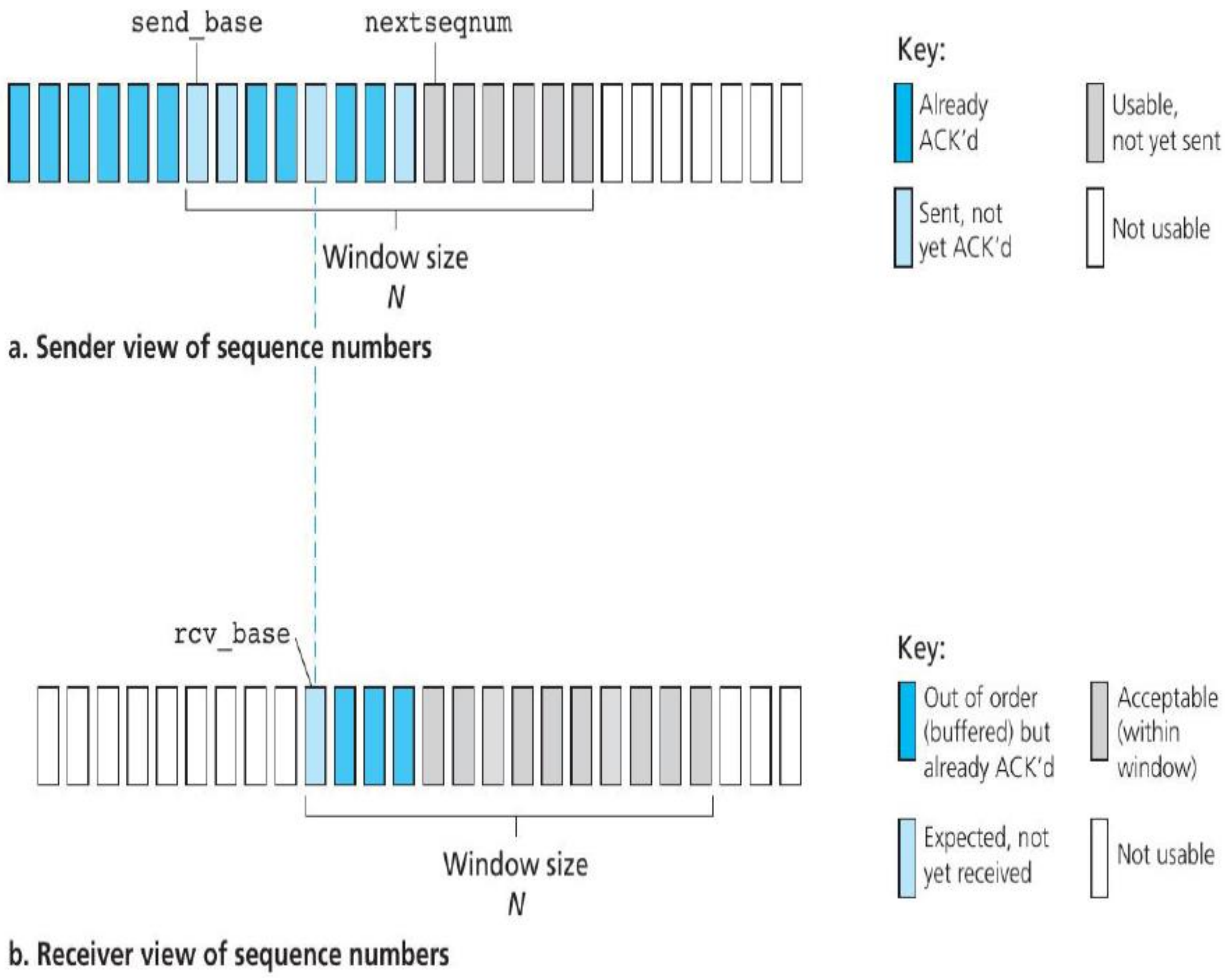
GBN Operation



Drawbacks of GBN:

- A single packet error can thus cause GBN to retransmit a large number of packets, many unnecessarily
- probability of channel errors increases, the pipeline can become filled with unnecessary retransmissions

Selective-Repeat



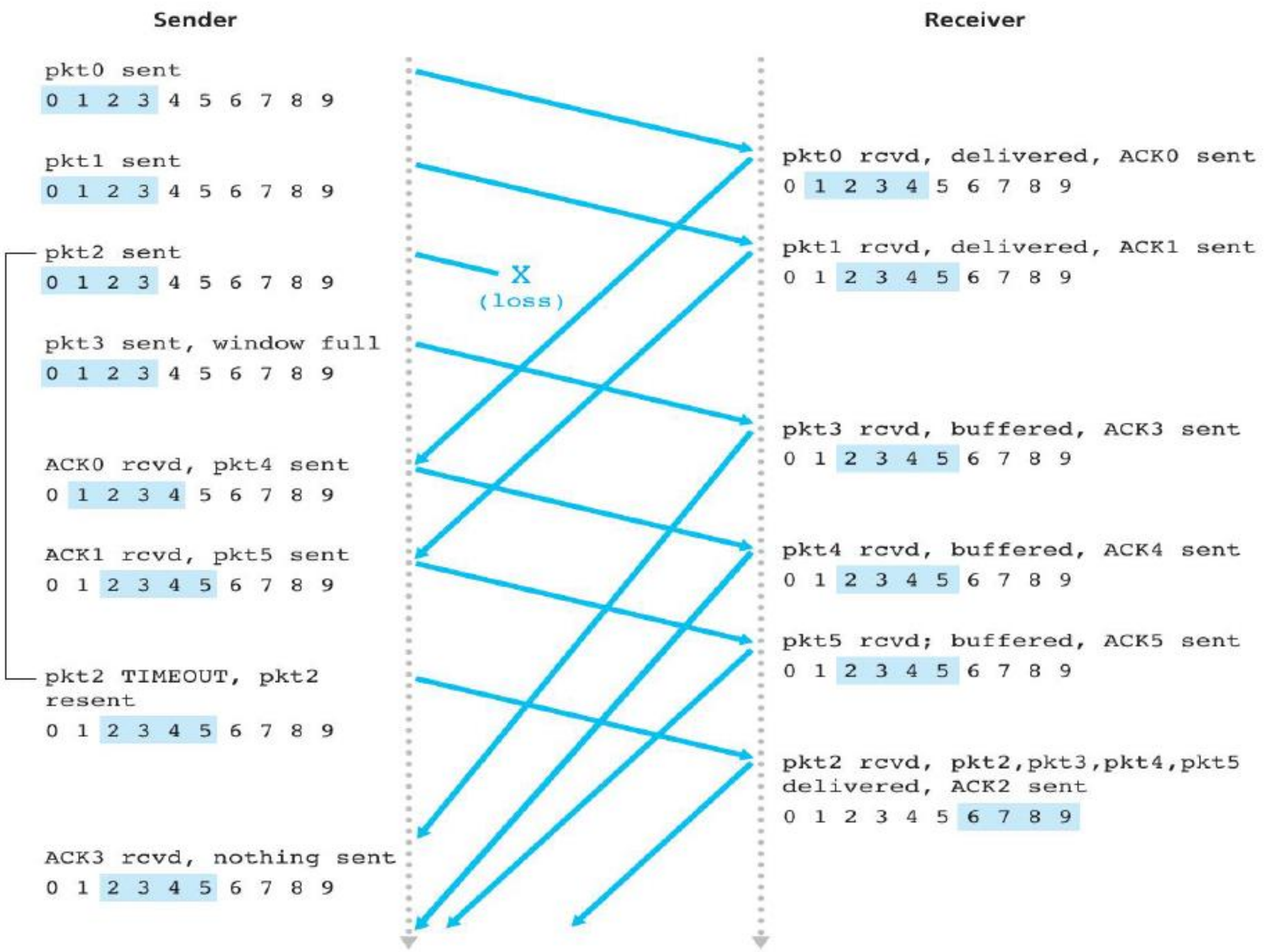
SR sender events and actions

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.

SR receiver events and actions

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.

SR Operation



Window Size in SR