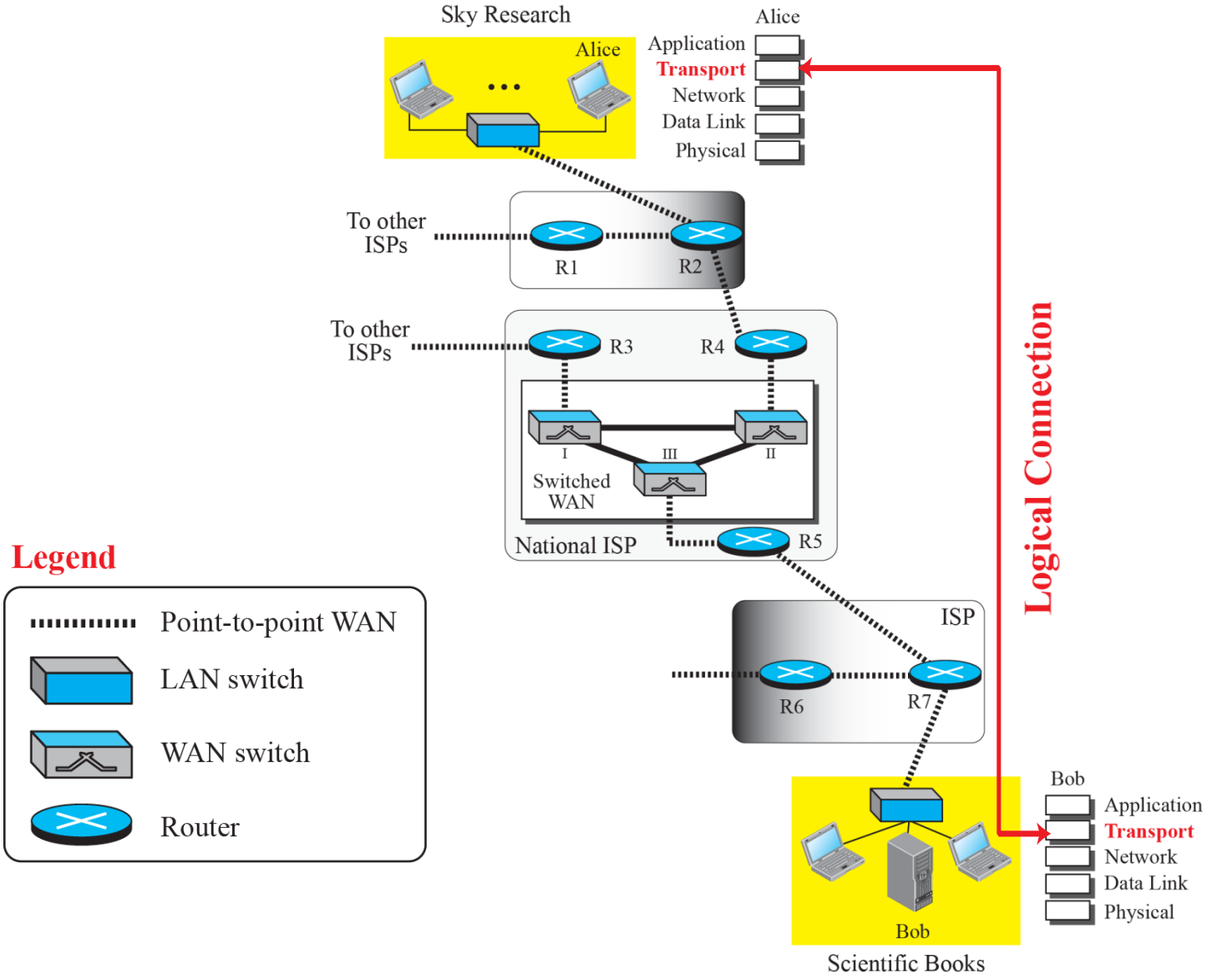
Chapter 23

Introduction to Transport Layer

INTRODUCTION

The transport layer is located between the application layer and the network layer. It provides a process-to-process communication between two application layers, one at the local host and the other at the remote host. Communication is provided using a logical connection. Figure: 23.1 shows the idea behind this logical connection.

Figure 23.1: Logical connection at the transport layer



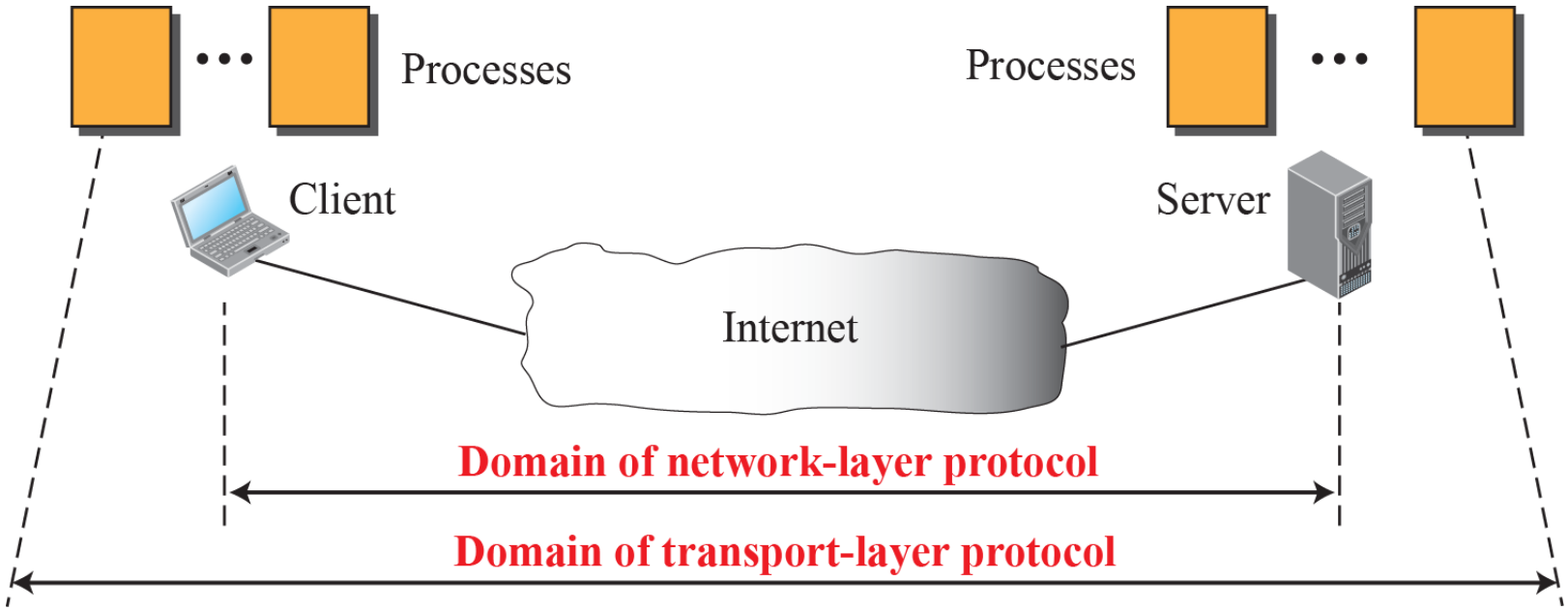
Transport-Layer Services

- As we discussed in Chapter 2, the transport layer is located between the network layer and the application layer. The transport layer is responsible for providing services to the application layer; it receives services from the network layer.
- Transport Layer Services:
  - Process-to-Process Communication
  - Addressing: Port Numbers
  - Encapsulation and Decapsulation
  - Multiplexing and Demultiplexing
  - Flow Control
  - Error Control
  - Combination of Flow and Error Control
  - Congestion Control

Process-to-Process Communication

- The first duty of a transport-layer protocol is to provide process-to-process communication.
- A process is an application-layer entity (running program) that uses the services of the transport layer.
- The network layer is responsible for communication at the computer level (host-to-host communication).
- A network-layer protocol can deliver the message only to the destination computer.
- However, this is an incomplete delivery.
- The message still needs to be handed to the correct process.
- This is where a transport-layer protocol takes over.
- A transport-layer protocol is responsible for delivery of the message to the appropriate process.

Process-to-Process Communication



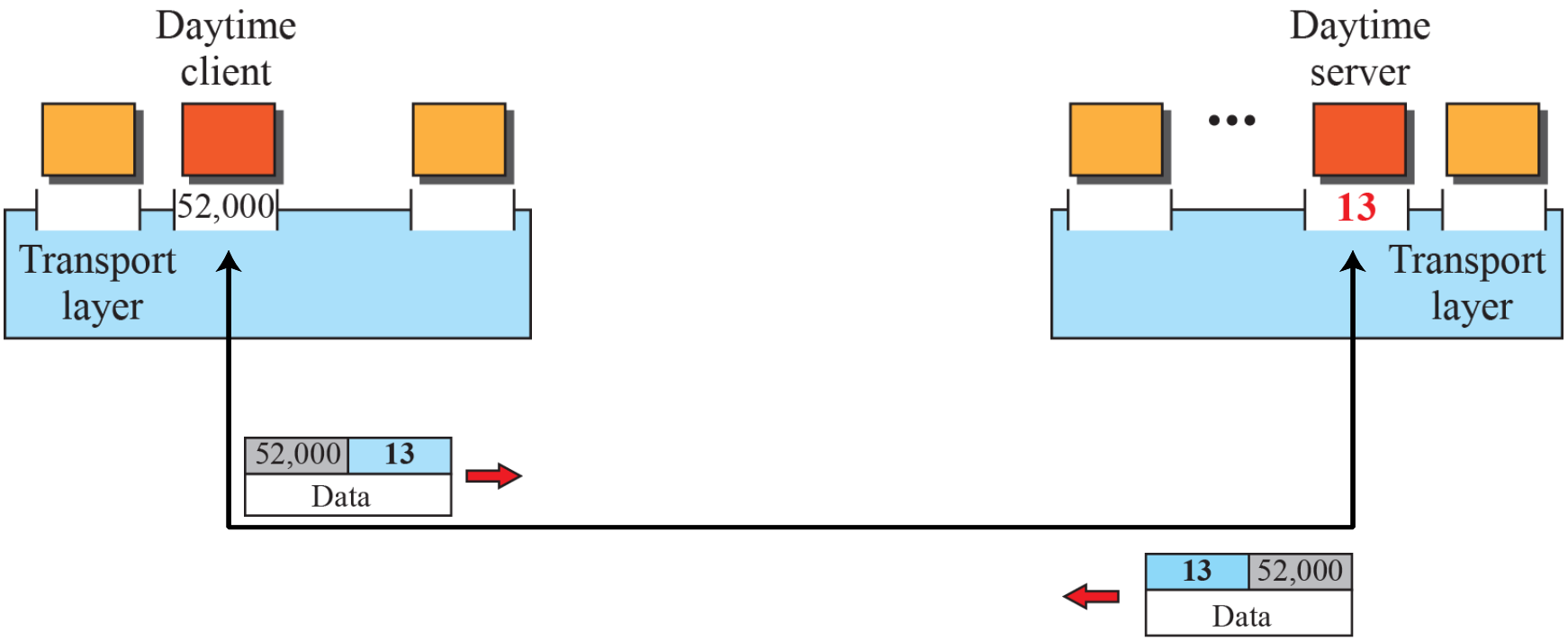
Addressing: Port Numbers

- Although there are a few ways to achieve process-to-process communication, the most common is through the client-server paradigm
- A process on the local host, called a client, needs services from a process usually on the remote host, called a server.
- A remote computer can run several server programs at the same time, just as several local computers can run one or more client programs at the same time.
- For communication, we must define the local host, local process, remote host, and remote process.
- The local host and the remote host are defined using IP addresses
- To define the processes, we need second identifiers, called port numbers.
- In the TCP/IP protocol suite, the port numbers are integers between 0 and 65,535 (16 bits).

Addressing: Port Numbers

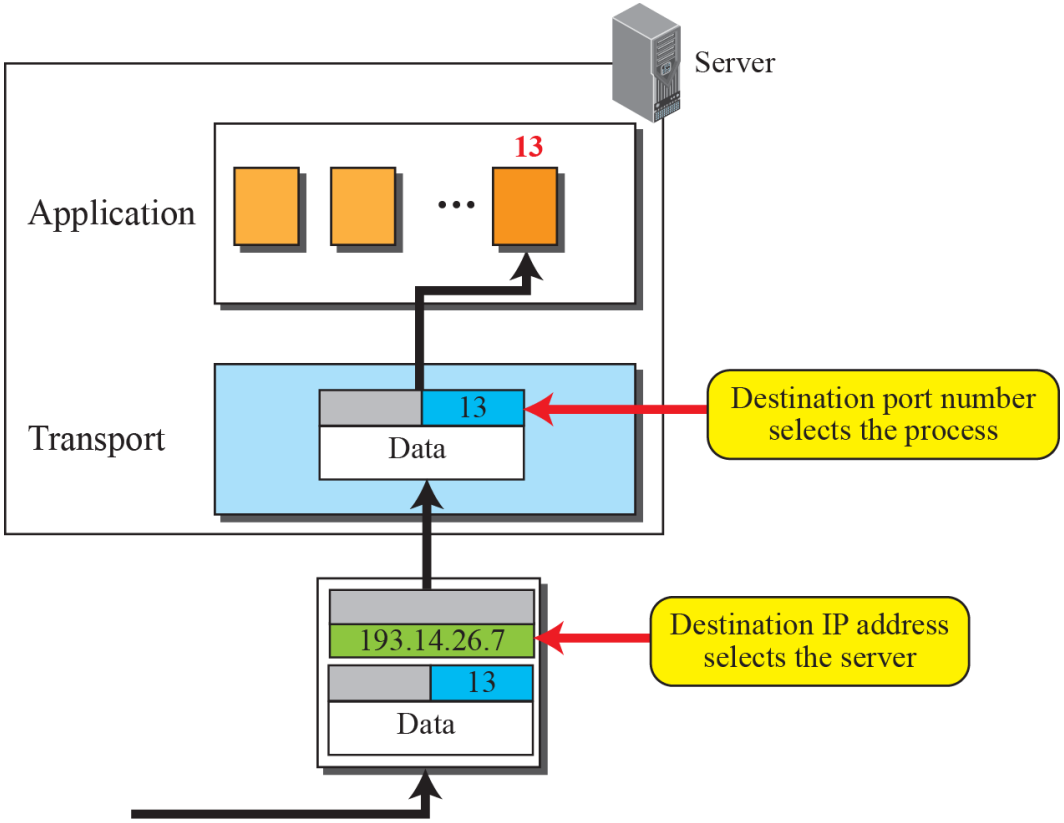
- The client program defines itself with a port number, called the ephemeral port number.
- The word ephemeral means “short-lived” and is used because the life of a client is normally short.
- An ephemeral port number is recommended to be greater than 1023 for some client/server programs to work properly.
- The server process must also define itself with a port number.
- This port number, however, cannot be chosen randomly.
- If the computer at the server site runs a server process and assigns a random number as the port number, the process at the client site that wants to access that server and use its services will not know the port number.
- Of course, one solution would be to send a special packet and request the port number of a specific server, but this creates more overhead.
- TCP/IP has decided to use universal port numbers for servers; these are called well-known port numbers.
- There are some exceptions to this rule; for example, there are clients that are assigned well-known port numbers.

Addressing: Port Numbers



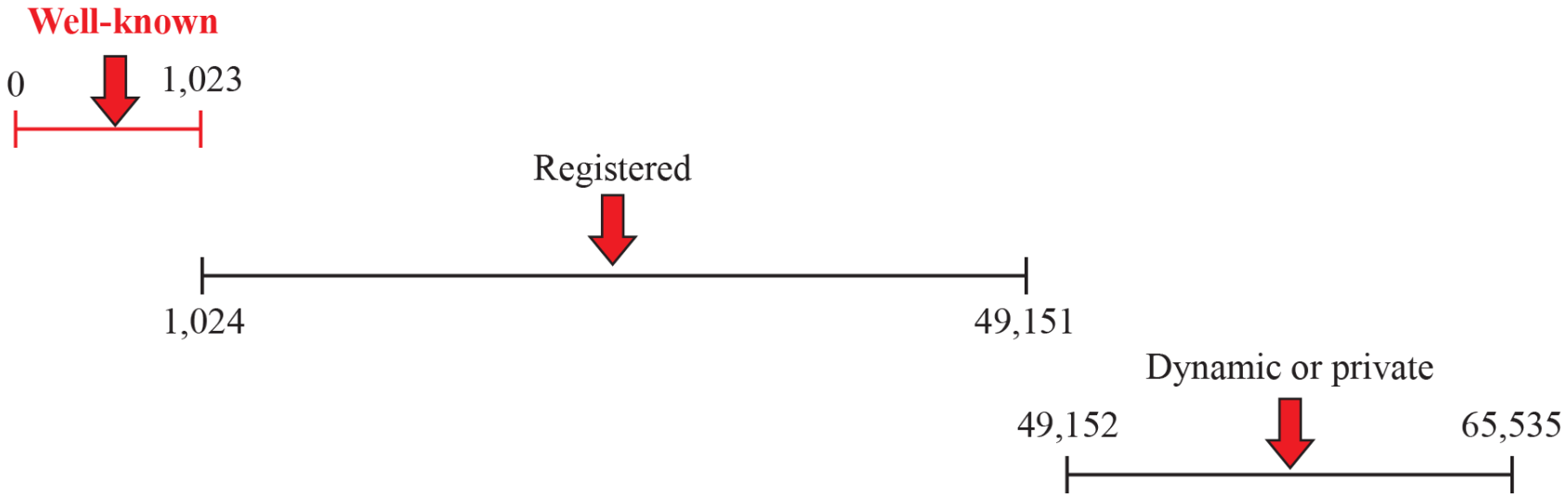
- Every client process knows the well-known port number of the corresponding server process.
- For example, while the daytime client process, a well-known client program, can use an ephemeral (temporary) port number, 52,000, to identify itself, the daytime server process must use the well-known (permanent) port number 13

Figure 23.4: IP addresses versus port numbers



- IP addresses and port numbers play different roles in selecting the final destination of data
- The destination IP address defines the host among the different hosts in the world.
- After the host has been selected, the port number defines one of the processes on this particular host

ICANN Ranges



- ICANN has divided the port numbers into three ranges:
  - ***Well-known ports.*** The ports ranging from 0 to 1023 are assigned and controlled by ICANN. These are the well-known ports.
  - ***Registered ports.*** The ports ranging from 1024 to 49,151 are not assigned or controlled by ICANN. They can only be registered with ICANN to prevent duplication.
  - ***Dynamic ports.*** The ports ranging from 49,152 to 65,535 are neither controlled nor registered. They can be used as temporary or private port numbers.

Example 23.1

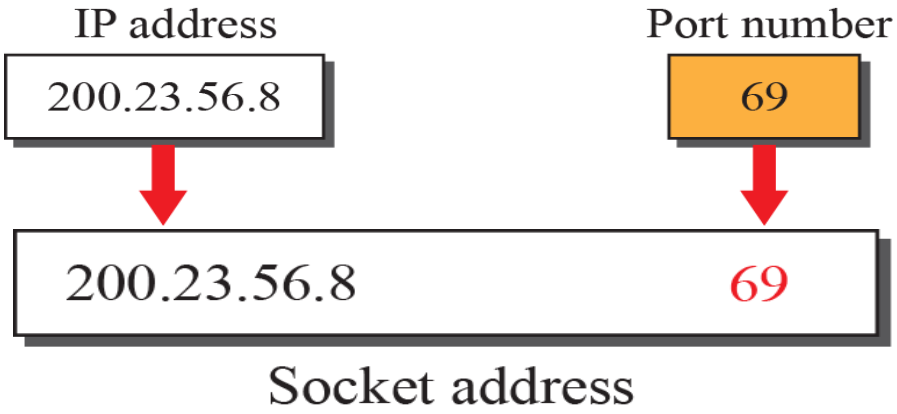
In UNIX, the well-known ports are stored in a file called /etc/services. We can use the *grep* utility to extract the line corresponding to the desired application.

```
$grep     tftp/etc/services
tftp 69/tcp
tftp 69/udp
```

SNMP (see Chapter 27) uses two port numbers (161 and 162), each for a different purpose.

```
$grep     snmp/etc/services
snmp161/tcp#Simple Net Mgmt Proto
snmp161/udp#Simple Net Mgmt Proto
snmptrap162/udp#Traps for SNMP
```

Socket Addresses

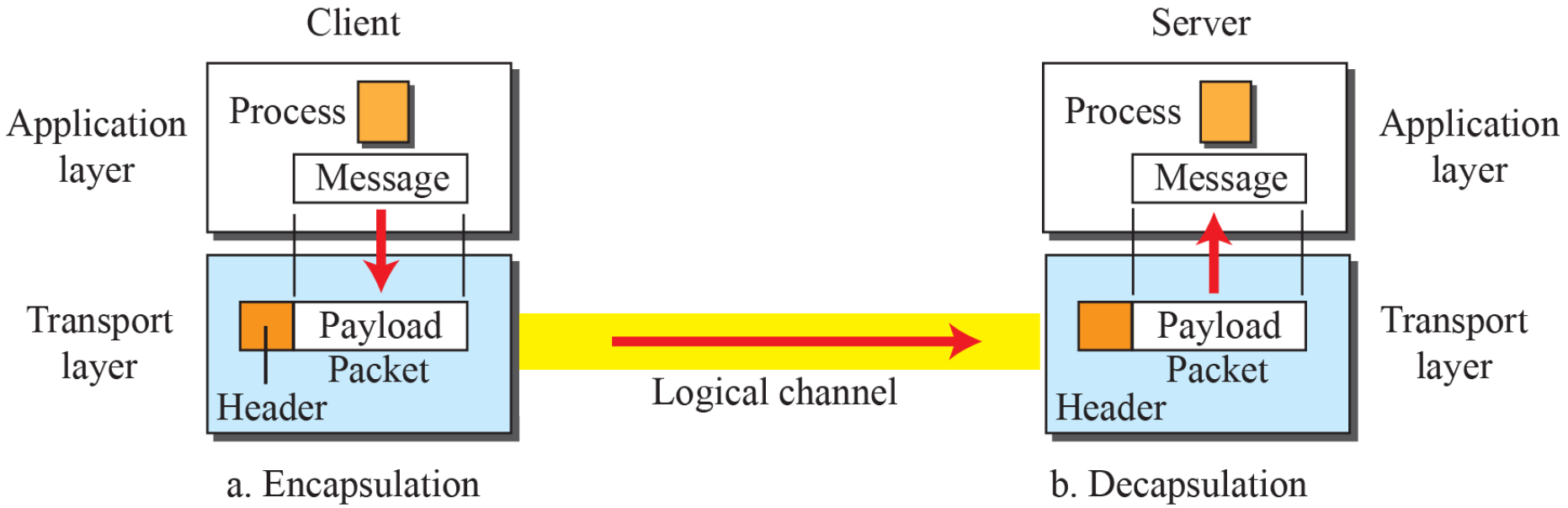


- A transport-layer protocol in the TCP suite needs both the IP address and the port number, at each end, to make a connection.
- The combination of an IP address and a port number is called a ***socket address.***
- The client socket address defines the client process uniquely just as the server socket address defines the server process uniquely
- To use the services of the transport layer in the Internet, we need a pair of socket addresses: the client socket address and the server socket address. These four pieces of information are part of the network-layer packet header and the transport-layer packet header.
- The first header contains the IP addresses; the second header contains the port numbers.

Encapsulation and Decapsulation

- To send a message from one process to another, the transport-layer protocol encapsulates and decapsulates messages
- Encapsulation happens at the sender site.
- When a process has a message to send, it passes the message to the transport layer along with a pair of socket addresses and some other pieces of information, which depend on the transport-layer protocol
- The transport layer receives the data and adds the transport-layer header
- The packets at the transport layer in the Internet are called user datagrams, segments, or packets, depending on what transport-layer protocol we use.
- Decapsulation happens at the receiver site.
- When the message arrives at the destination transport layer, the header is dropped and the transport layer delivers the message to the process running at the application layer.
- The sender socket address is passed to the process in case it needs to respond to the message received

Encapsulation and Decapsulation

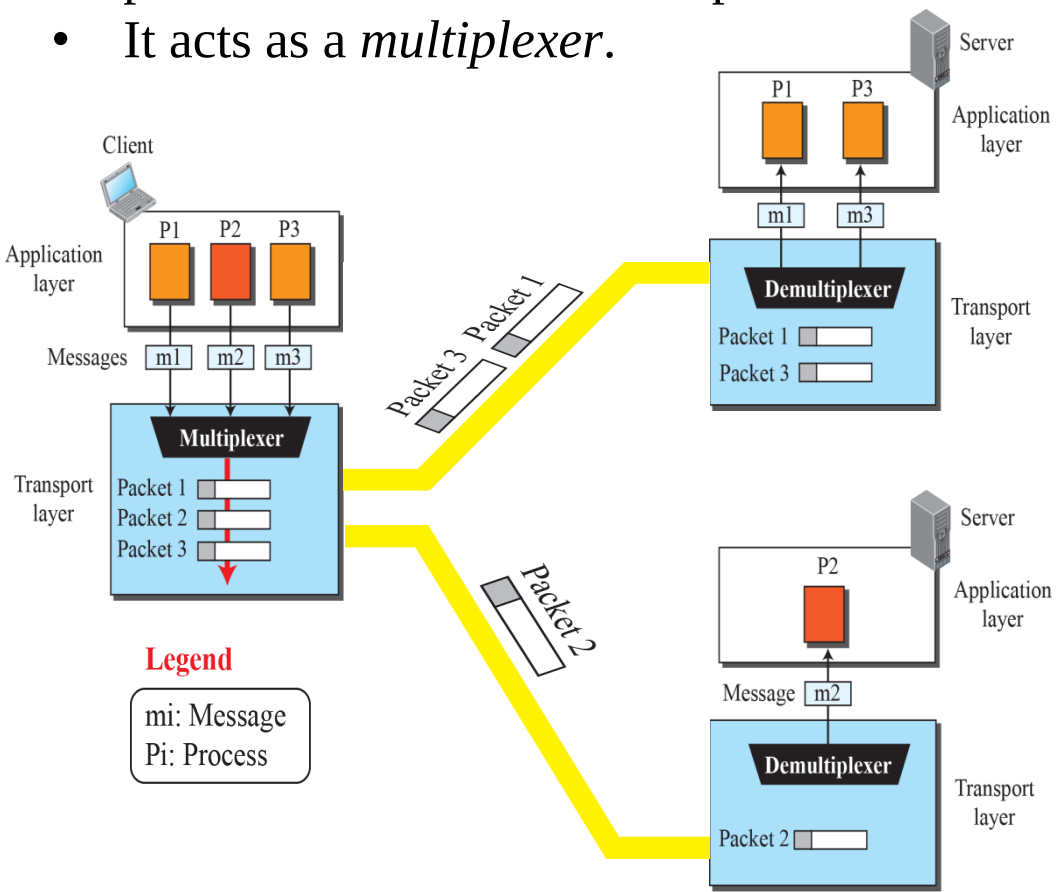


Multiplexing and Demultiplexing

- Whenever an entity accepts items from more than one source, this is referred to as multiplexing (many to one)
- Whenever an entity delivers items to more than one source, this is referred to as demultiplexing (one to many).
- The transport layer at the source performs multiplexing; the transport layer at the destination performs demultiplexing

Multiplexing and Demultiplexing

- Three client processes are running at the client site, P1, P2, and P3.
- The processes P1 and P3 need to send requests to the corresponding server process running in a server.
- The client process P2 needs to send a request to the corresponding server process running at another server.
- The transport layer at the client site accepts three messages from the three processes and creates three packets.
- It acts as a *multiplexer*.



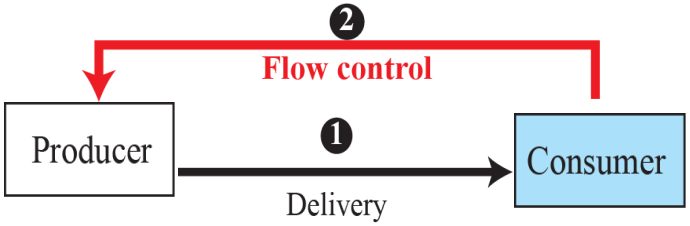
- The packets 1 and 3 use the same logical channel to reach the transport layer of the first server.
- When they arrive at the server, the transport layer does the job of a *demultiplexer* and distributes the messages to two different processes.
- The transport layer at the second server receives packet 2 and delivers it to the corresponding process.

Flow Control

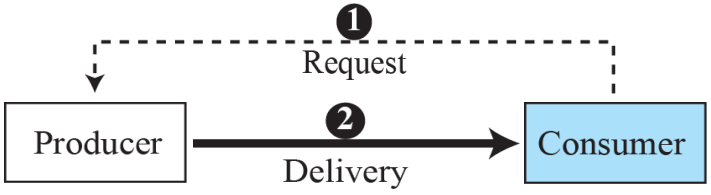
- Whenever an entity produces items and another entity consumes them, there should be a balance between production and consumption rates.
- If the items are produced faster than they can be consumed, the consumer can be overwhelmed and may need to discard some items.
- If the items are produced more slowly than they can be consumed, the consumer must wait, and the system becomes less efficient.
- Flow control is related to the first issue.
- We need to prevent losing the data items at the consumer site.

Pushing or pulling

- Delivery of items from a producer to a consumer can occur in one of two ways: pushing or pulling.
- If the sender delivers items whenever they are produced–without a prior request from the consumer–the delivery is referred to as pushing.
- If the producer delivers the items after the consumer has requested them, the delivery is referred to as pulling.
- When the producer pushes the items, the consumer may be overwhelmed and there is a need for flow control, in the opposite direction, to prevent discarding of the items.
- In other words, the consumer needs to warn the producer to stop the delivery and to inform the producer when it is again ready to receive the items.
- When the consumer pulls the items, it requests them when it is ready. In this case, there is no need for flow control.



a. Pushing

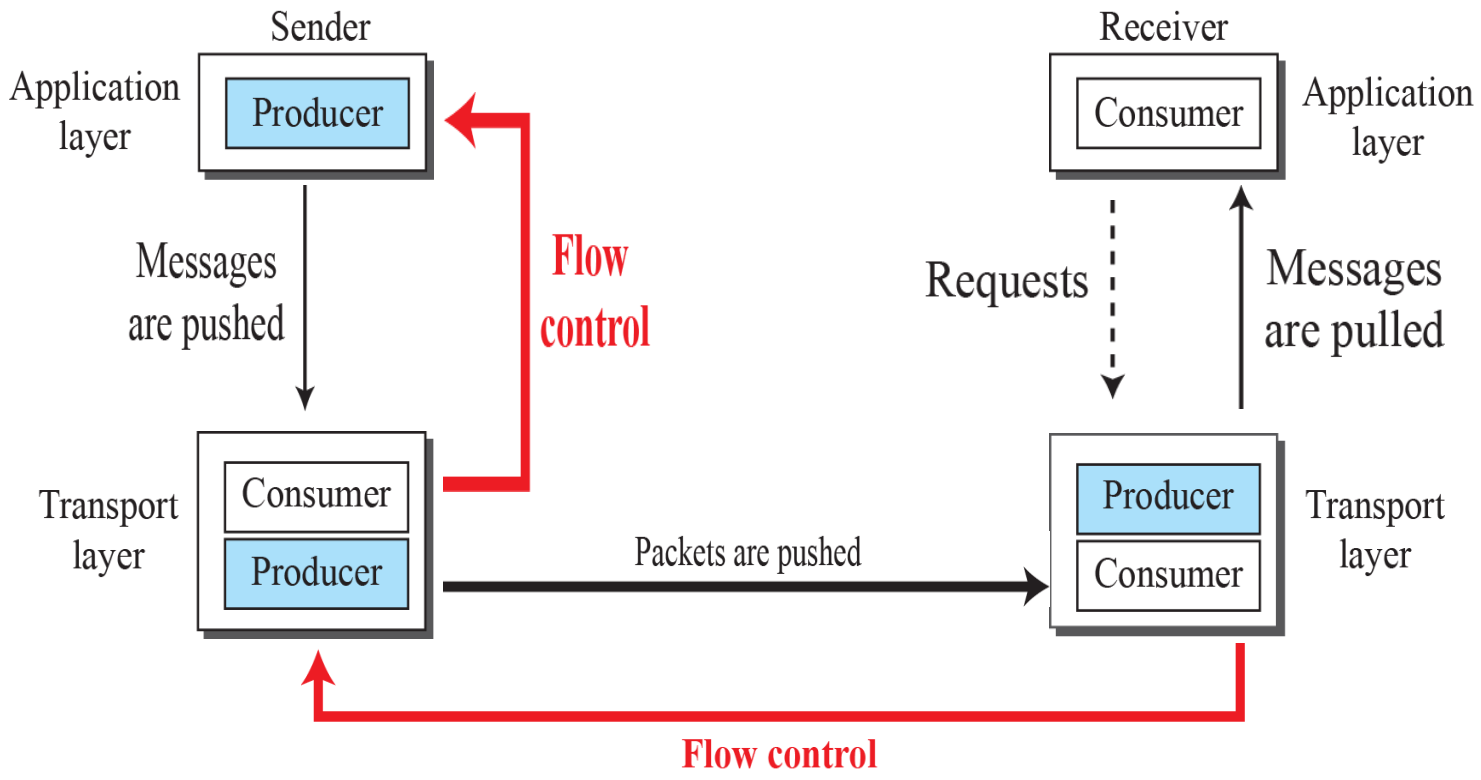


b. Pulling

Flow Control at Transport Layer

- In communication at the transport layer, we are dealing with four entities: sender process, sender transport layer, receiver transport layer, and receiver process.
- The sending process at the application layer is only a producer.
- It produces message chunks and pushes them to the transport layer.
- The sending transport layer has a double role: it is both a consumer and a producer.
- It consumes the messages pushed by the producer.
- It encapsulates the messages in packets and pushes them to the receiving transport layer.
- The receiving transport layer also has a double role: it is the consumer for the packets received from the sender and the producer that decapsulates the messages and delivers them to the application layer.
- The last delivery, however, is normally a pulling delivery; the transport layer waits until the application-layer process asks for messages.

Flow Control at Transport Layer



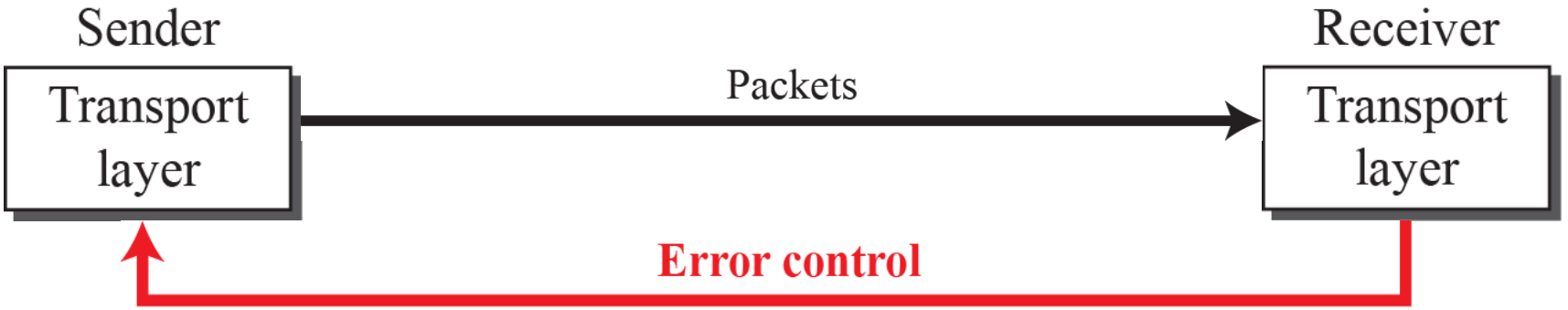
Buffers

- Although flow control can be implemented in several ways, one of the solutions is normally to use two buffers: one at the sending transport layer and the other at the receiving transport layer.
- A buffer is a set of memory locations that can hold packets at the sender and receiver.
- The flow control communication can occur by sending signals from the consumer to the producer.
- When the buffer of the sending transport layer is full, it informs the application layer to stop passing chunks of messages; when there are some vacancies, it informs the application layer that it can pass message chunks again.
- When the buffer of the receiving transport layer is full, it informs the sending transport layer to stop sending packets.
- When there are some vacancies, it informs the sending transport layer that it can send packets again.

Error control at the transport layer

- Error control at the transport layer is responsible for:

  1. Detecting and discarding corrupted packets.
  2. Keeping track of lost and discarded packets and resending them.
  3. Recognizing duplicate packets and discarding them.
  4. Buffering out-of-order packets until the missing packets arrive.



- Error control, unlike flow control, involves only the sending and receiving transport layers.
- We are assuming that the message chunks exchanged between the application and transport layers are error free.

Sequence Numbers

Error control requires that the sending transport layer knows which packet is to be resent and the receiving transport layer knows which packet is a duplicate, or which packet has arrived out of order.

This can be done if the packets are numbered.

We can add a field to the transport-layer packet to hold the sequence number of the packet.

When a packet is corrupted or lost, the receiving transport layer can somehow inform the sending transport layer to resend that packet using the sequence number.

The receiving transport layer can also detect duplicate packets if two received packets have the same sequence number.

The out-of-order packets can be recognized by observing gaps in the sequence numbers.

We can add a field to the transport-layer packet to hold the **sequence number** of the packet

Packets are numbered sequentially.

However, because we need to include the sequence number of each packet in the header, we need to set a limit. If the header of the packet allows m bits for the sequence number, the sequence numbers range from 0 to $2^m - 1$.

Sequence Numbers

If *m* is 4, the only sequence numbers are 0 through 15, inclusive.
However, we can wrap around the sequence.
So the sequence numbers in this case are:

0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, ...

In other words, the sequence numbers are modulo 2^m.

For error control, the sequence numbers are modulo 2^m, where *m* is the size of the sequence number field in bits.

Acknowledgement

- We can use both positive and negative signals as error control, but we discuss only positive signals, which are more common at the transport layer.
- The receiver side can send an acknowledgment (ACK) for each of a collection of packets that have arrived safe and sound.
- The receiver can simply discard the corrupted packets.
- The sender can detect lost packets if it uses a timer.
- When a packet is sent, the sender starts a timer.
- If an ACK does not arrive before the timer expires, the sender resends the packet.
- Duplicate packets can be silently discarded by the receiver.
- Out-of-order packets can be either discarded (to be treated as lost packets by the sender), or stored until the missing one arrives.

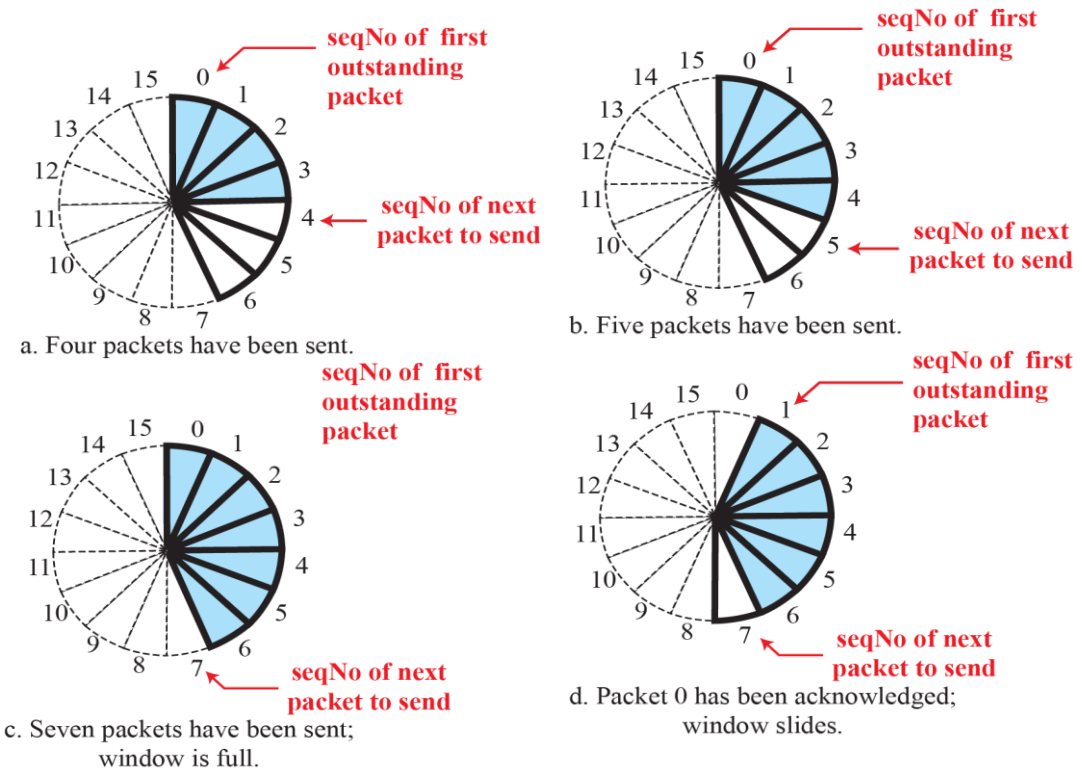
Combination of Flow and Error Control

- We have discussed that flow control requires the use of two buffers, one at the sender site and the other at the receiver site.
- We have also discussed that error control requires the use of sequence and acknowledgment numbers by both sides.
- These two requirements can be combined if we use two numbered buffers, one at the sender, one at the receiver.
- At the sender, when a packet is prepared to be sent, we use the number of the next free location, *x*, in the buffer as the sequence number of the packet.
- When the packet is sent, a copy is stored at memory location *x*, awaiting the acknowledgment from the other end.
- When an acknowledgment related to a sent packet arrives, the packet is purged and the memory location becomes free.
- At the receiver, when a packet with sequence number *y* arrives, it is stored at the memory location *y* until the application layer is ready to receive it.
- An acknowledgment can be sent to announce the arrival of packet *y*.

Sliding Window

- Since the sequence numbers use modulo 2^m, a circle can represent the sequence numbers from 0 to $2^m - 1$
- The buffer is represented as a set of slices, called the ***sliding window,*** that occupies part of the circle at any time.
- At the sender site, when a packet is sent, the corresponding slice is marked.
- When all the slices are marked, it means that the buffer is full and no further messages can be accepted from the application layer.
- When an acknowledgment arrives, the corresponding slice is unmarked.
- If some consecutive slices from the beginning of the window are unmarked, the window slides over the range of the corresponding sequence numbers to allow more free slices at the end of the window.

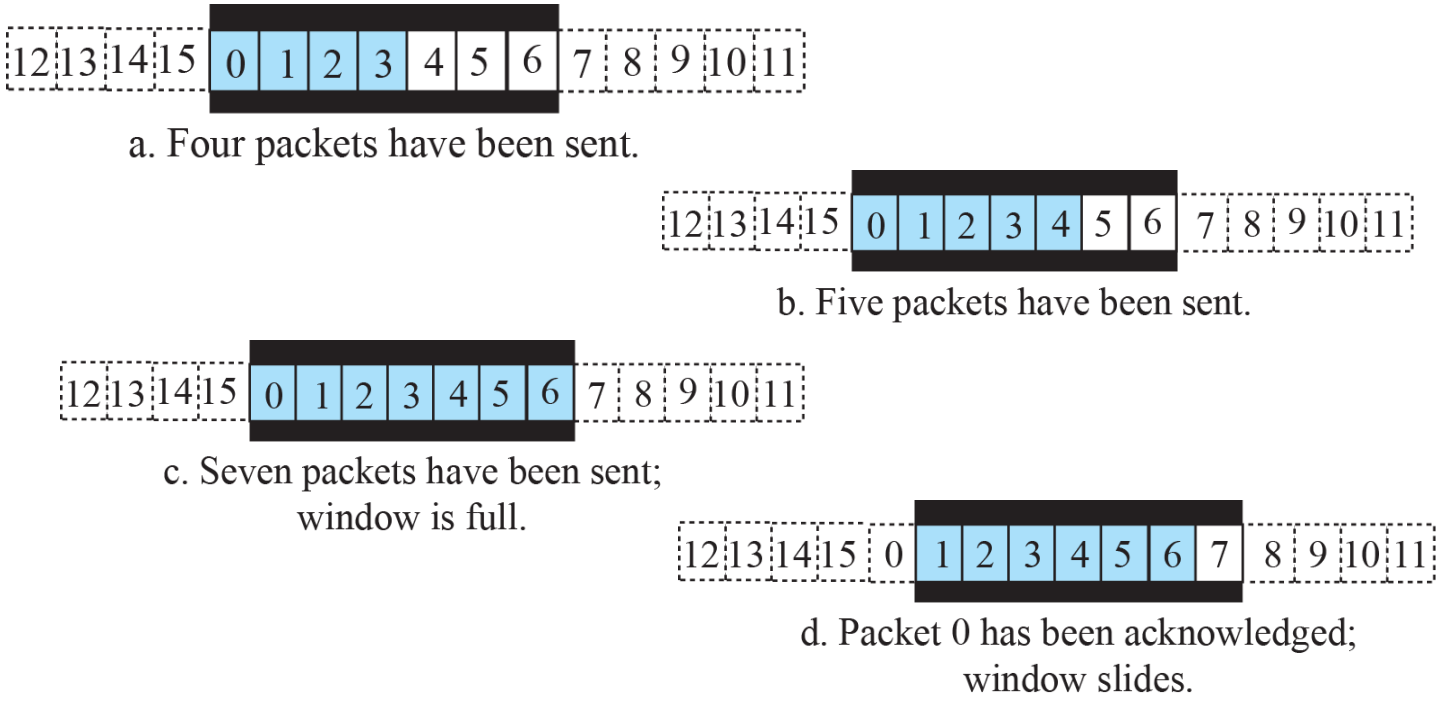
Sliding window in circular format



The Sliding Window at the Sender.

- The sequence numbers are in modulo 16 (m = 4) and the size of the window is 7.
- Note that the sliding window is just an abstraction: the actual situation uses computer variables to hold the sequence numbers of the next packet to be sent and the last packet sent.

Sliding window in linear format



- Most protocols show the sliding window using linear representation.
- The idea is the same, but it normally takes less space on paper.
- Both representations tell us the same thing.

Congestion Control

Congestion in a network may occur if the *load* on the network—the number of packets sent to the network—is greater than the *capacity* of the network—the number of packets a network can handle. **Congestion control** refers to the mechanisms and techniques that control the congestion and keep the load below the capacity.

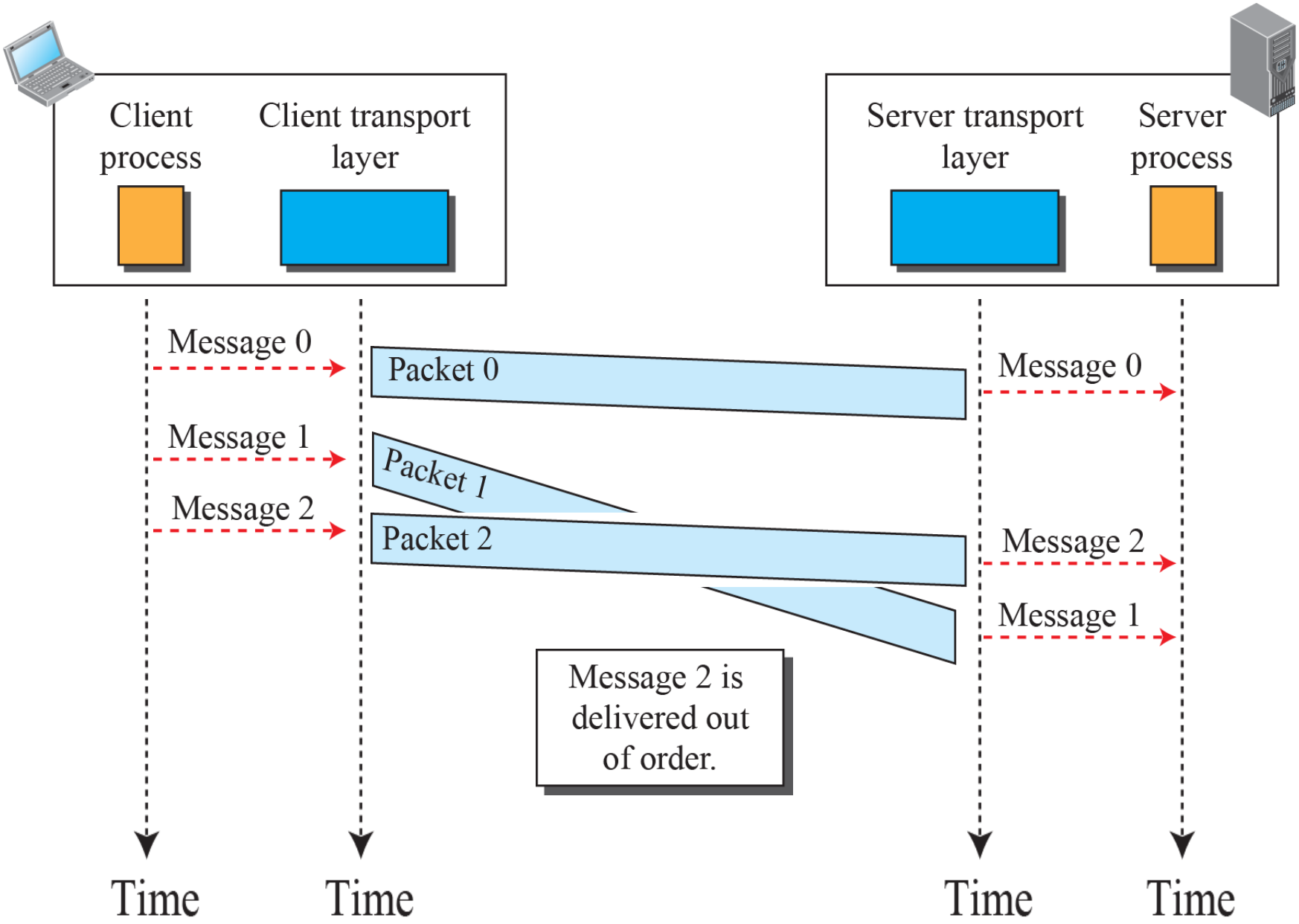
Connectionless and Connection-Oriented Protocols

- A transport-layer protocol, like a network-layer protocol, can provide two types of services: connectionless and connection-oriented.
- The nature of these services at the transport layer, however, is different from the ones at the network layer.
- At the network layer, a connectionless service may mean different paths for different datagrams belonging to the same message.
- At the transport layer, we are not concerned about the physical paths of packets (we assume a logical connection between two transport layers).
- Connectionless service at the transport layer means independency between packets; connection-oriented means dependency.

Connectionless Service

- In a connectionless service, the source process (application program) needs to divide its message into chunks of data of the size acceptable by the transport layer and deliver them to the transport layer one by one.
- The transport layer treats each chunk as a single unit without any relation between the chunks.
- When a chunk arrives from the application layer, the transport layer encapsulates it in a packet and sends it.
- To show the independency of packets, assume that a client process has three chunks of messages to send to a server process.
- The chunks are handed over to the connectionless transport protocol
- in order.
- However, since there is no dependency between the packets at the transport
- layer, the packets may arrive out of order at the destination and will be delivered out of order to the server process

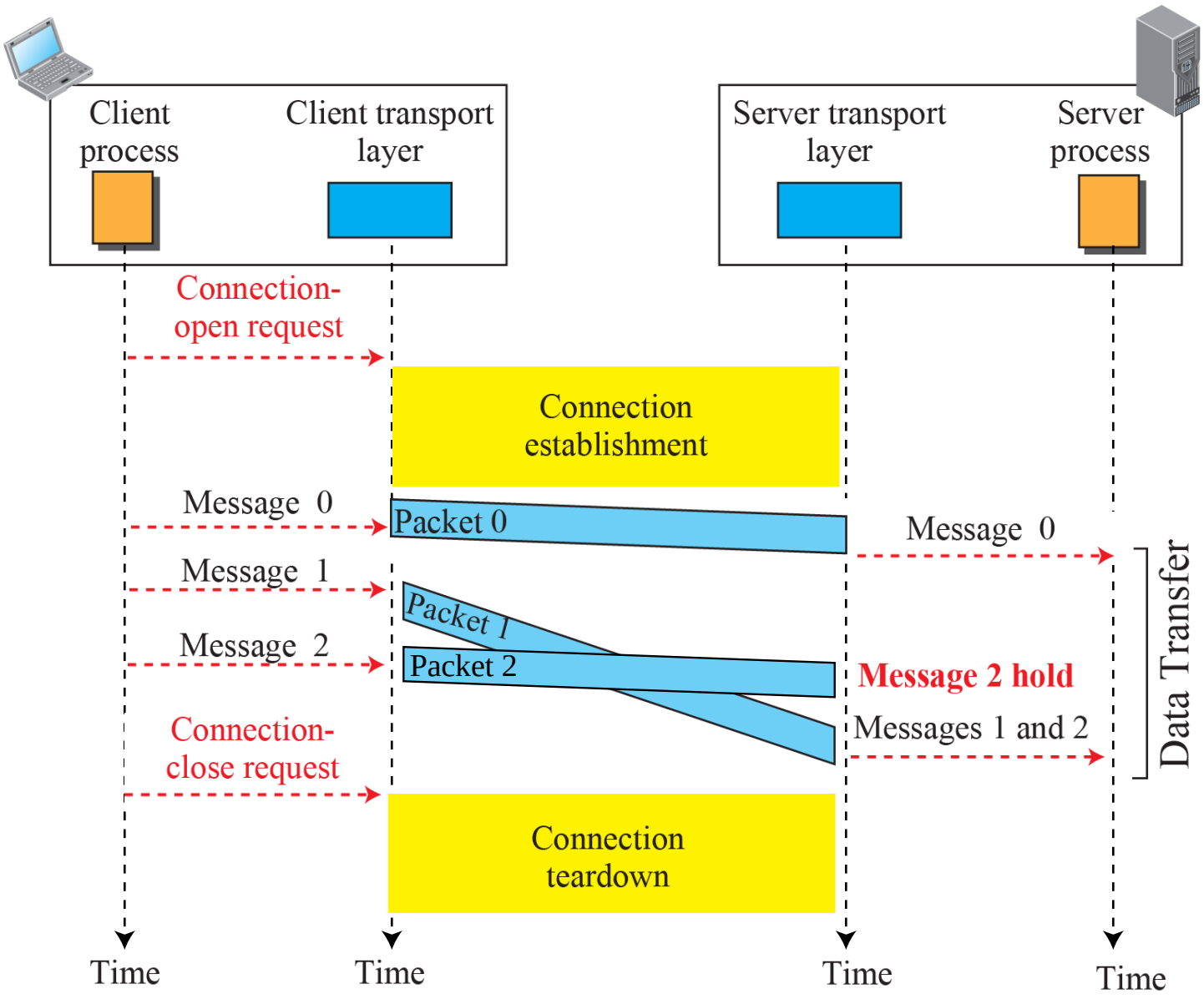
Connectionless Service



Connection-Oriented Service

- In a connection-oriented service, the client and the server first need to establish a logical connection between themselves.
- The data exchange can only happen after the connection establishment.
- After data exchange, the connection needs to be torn down
- As we mentioned before, the connection-oriented service at the transport layer is different from the same service at the network layer.
- In the network layer, connection-oriented service means a coordination between the two end hosts and all the routers in between.
- At the transport layer, connection-oriented service involves only the two hosts; the service is end to end.
- This means that we should be able to make a connection-oriented protocol at the transport layer over either a connectionless or connection-oriented protocol at the network layer.

Connection-Oriented Service



TRANSPORT-LAYER PROTOCOLS

We can create a transport-layer protocol by combining a set of services described in the previous sections. To better understand the behavior of these protocols, we start with the simplest one and gradually add more complexity. The TCP/IP protocol uses a transport-layer protocol that is either a modification or a combination of some of these protocols.

Simple Protocol

Our first protocol is a simple connectionless protocol with neither flow nor error control. We assume that the receiver can immediately handle any packet it receives. In other words, the receiver can never be overwhelmed with incoming packets. Figure 23.17 shows the layout for this protocol.

Figure 23.17: Simple protocol

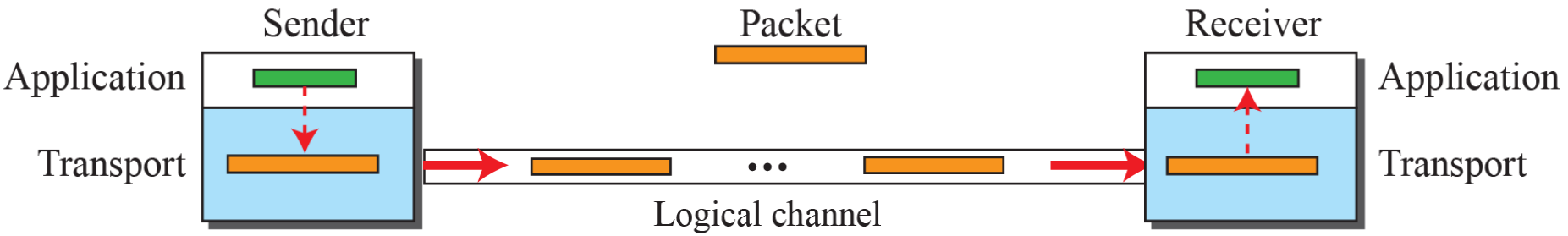
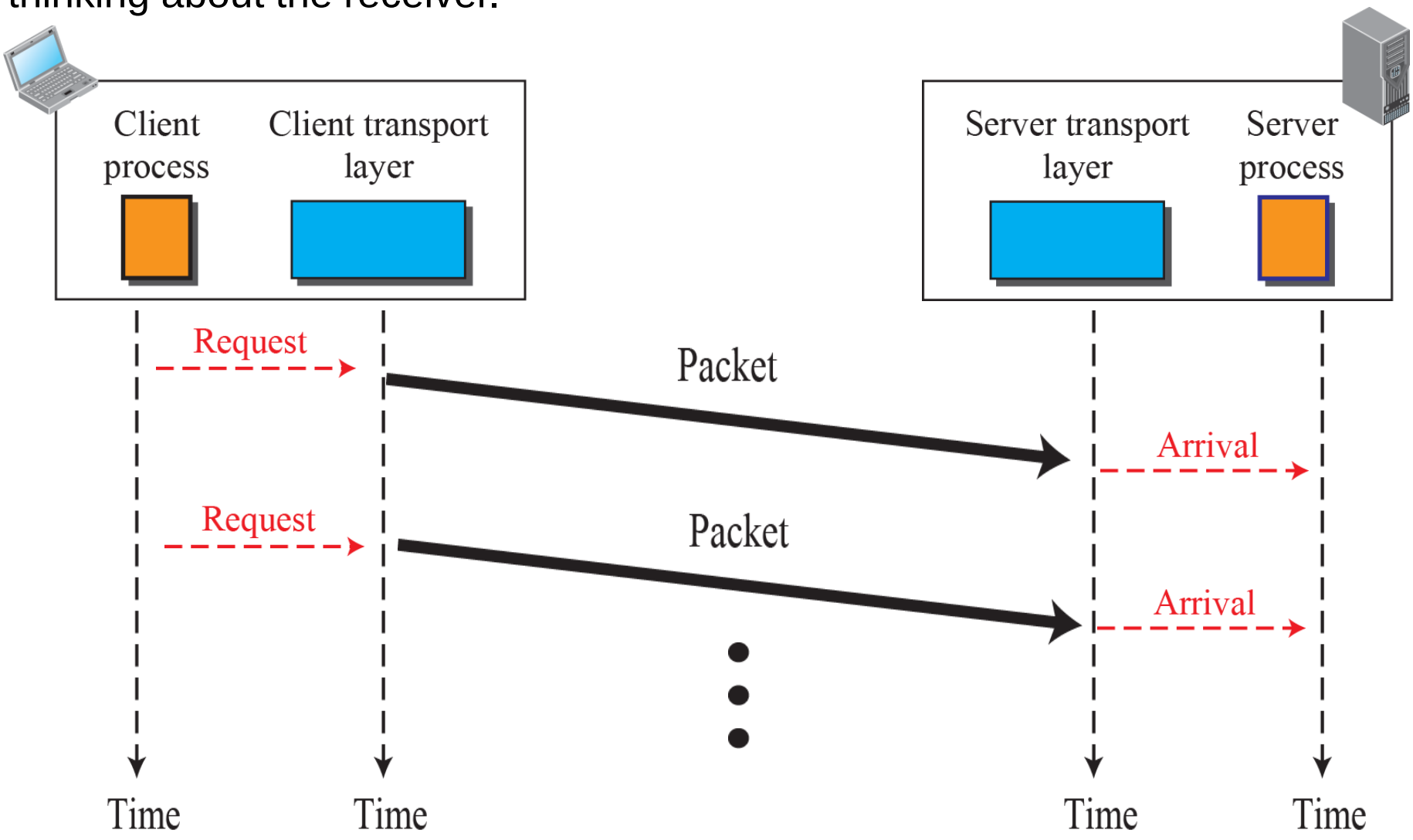


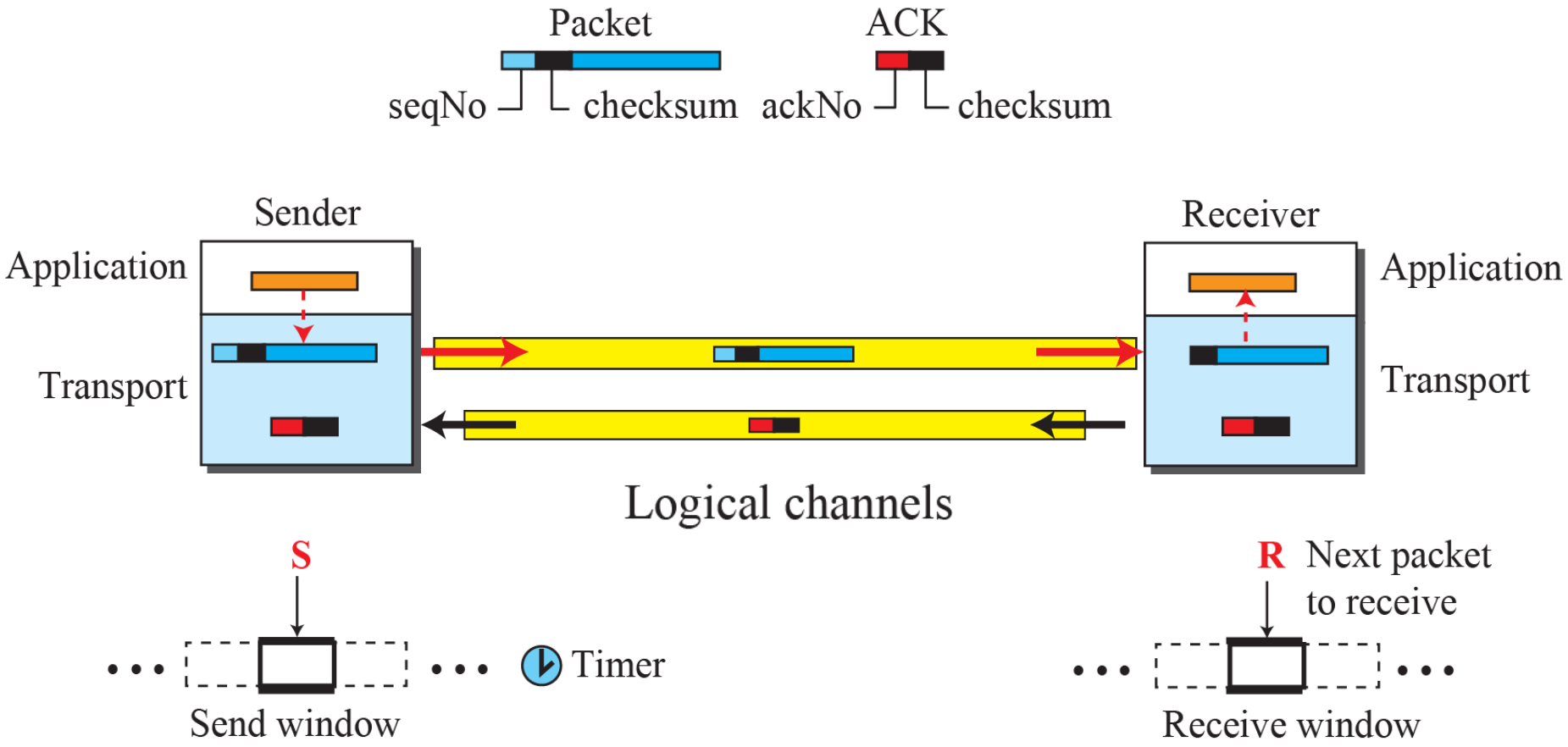
Figure shows an example of communication using this protocol. It is very simple. The sender sends packets one after another without even thinking about the receiver.



Stop-and-Wait Protocol

Our second protocol is a connection-oriented protocol called the Stop-and-Wait protocol, which uses both flow and error control. Both the sender and the receiver use a sliding window of size 1. The sender sends one packet at a time and waits for an acknowledgment before sending the next one. To detect corrupted packets, we need to add a checksum to each data packet. When a packet arrives at the receiver site, it is checked. If its checksum is incorrect, the packet is corrupted and silently discarded.

Figure 23.20: Stop-and-Wait protocol



Sequence Number

Assume we have used x as a sequence number; we only need to use x + 1 after that. There is no need for x + 2.

Has to be 0 and 1 no need for 2

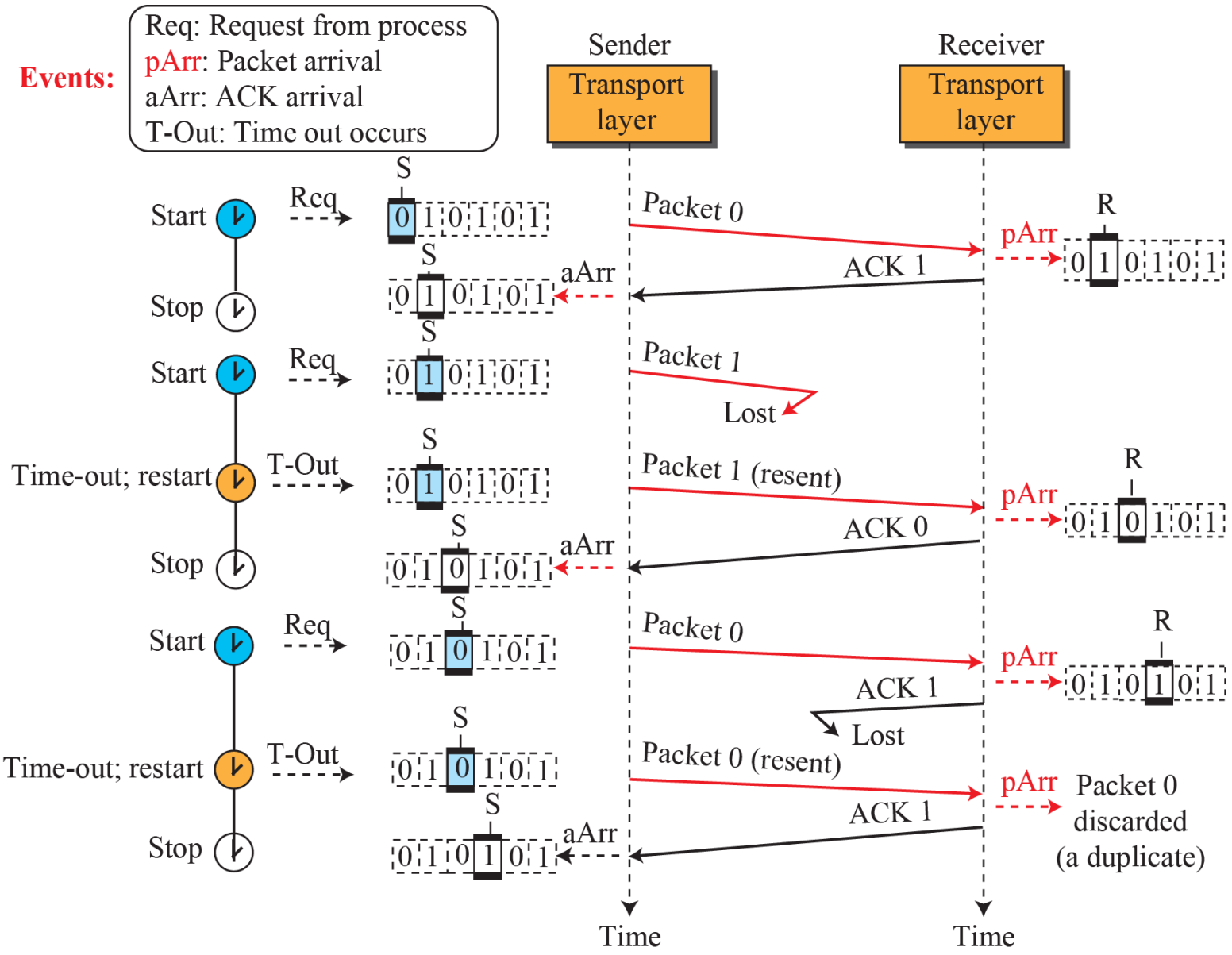
Acknowledgement Number

The acknowledgment numbers always announce the sequence number of the next packet expected by the receiver. For example, if packet 0 has arrived safe and sound, the receiver sends an ACK with acknowledgment 1 (meaning packet 1 is expected next).

Example 23.4

Figure 23.22 shows an example of the Stop-and-Wait protocol. Packet 0 is sent and acknowledged. Packet 1 is lost and resent after the time-out. The resent packet 1 is acknowledged and the timer stops. Packet 0 is sent and acknowledged, but the acknowledgment is lost. The sender has no idea if the packet or the acknowledgment is lost, so after the time-out, it resends packet 0, which is acknowledged.

Figure 23.22: Flow diagram for Example 3.4



Efficiency

- The Stop-and-Wait protocol is very inefficient if our channel has a large bandwidth (high data rate); and the round-trip delay is long.
- The product of these two is called the bandwidth-delay product.
- We can think of the channel as a pipe.
- The bandwidth-delay product then is the volume of the pipe in bits.
- The pipe is always there.
- It is not efficient if it is not used.
- The bandwidth-delay product is a measure of the number of bits a sender can transmit through the system while waiting for an acknowledgment from the receiver.

Example 23.5

Assume that, in a Stop-and-Wait system, the bandwidth of the line is 1 Mbps, and 1 bit takes 20 milliseconds to make a round trip. What is the bandwidth-delay product? If the system data packets are 1,000 bits in length, what is the utilization percentage of the link?

Solution

The bandwidth-delay product is $(1\times10^{6}) \times (20\times10^{-3}) = 20{,}000$ bits.

The system can send 20,000 bits during the time it takes for the data to go from the sender to the receiver and the acknowledgment to come back.

However, the system sends only 1,000 bits.

We can say that the link utilization is only 1,000/20,000, or 5 percent.

For this reason, in a link with a high bandwidth or long delay, the use of Stop-and-Wait wastes the capacity of the link.

Example 23.6

What is the utilization percentage of the link in Example 23.5 if we have a protocol that can send up to 15 packets before stopping and worrying about the acknowledgments?

Solution

The bandwidth-delay product is still 20,000 bits.
The system can send up to 15 packets or 15,000 bits during a round trip.
This means the utilization is 15,000/20,000, or 75 percent.

Of course, if there are damaged packets, the utilization percentage is much less because packets have to be resent.

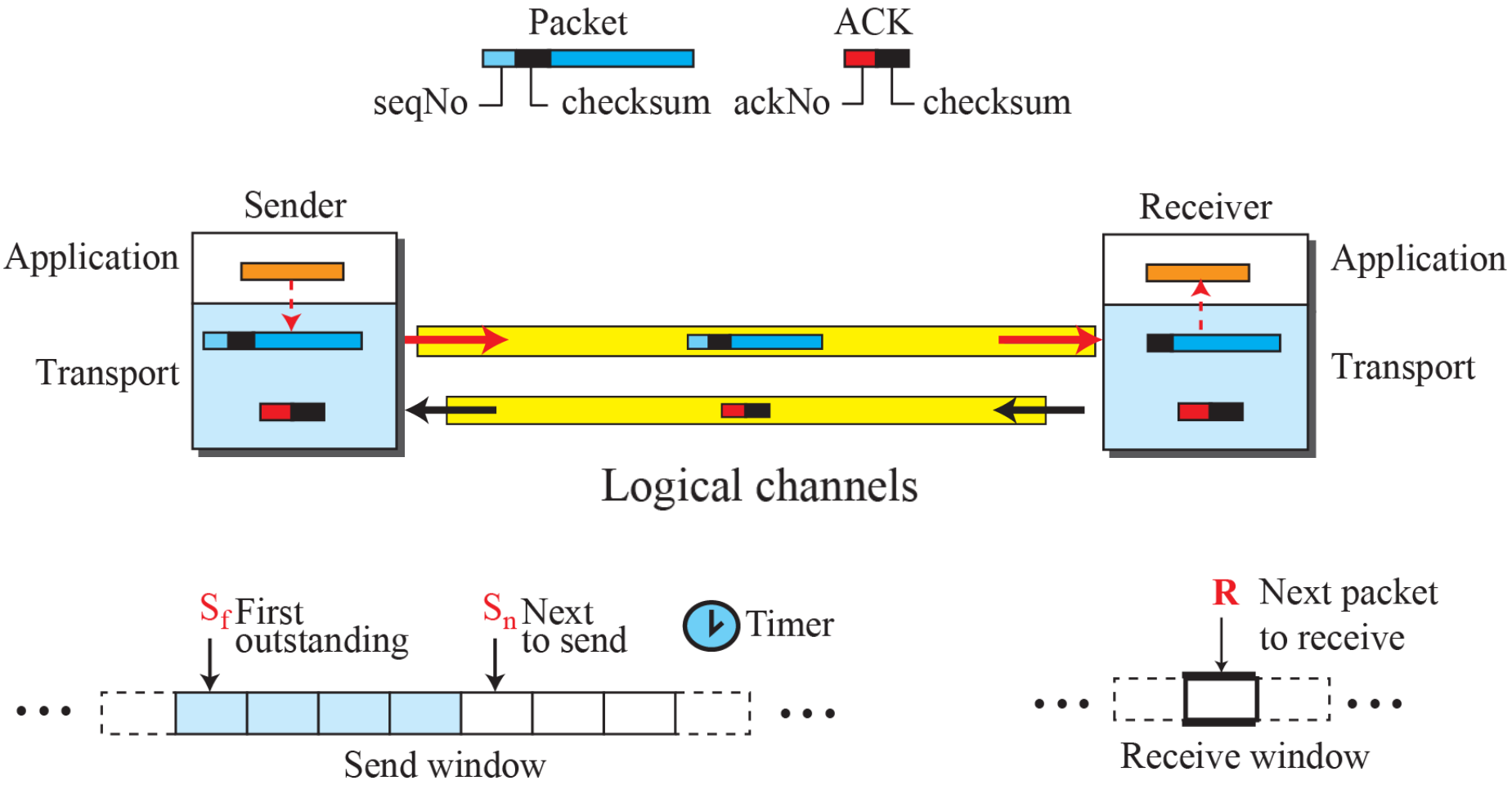
Pipelining

- In networking and in other areas, a task is often begun before the previous task has ended.
- This is known as pipelining.
- There is **no pipelining in the Stop-and-Wait protocol because a sender must wait for a packet to reach the destination and be acknowledged before the next packet can be sent.**
- However, pipelining does apply to our next two protocols because several packets can be sent before a sender receives feedback about the previous packets.
- Pipelining improves the efficiency of the transmission if the number of bits in transition is large with respect to the bandwidth-delay product.

Go-Back-N Protocol (GBN)

To improve the efficiency of transmission (to fill the pipe), multiple packets must be in transition while the sender is waiting for acknowledgment. In other words, we need to let more than one packet be outstanding to keep the channel busy while the sender is waiting for acknowledgment. In this section, we discuss one protocol that can achieve this goal; in the next section, we discuss a second. The first is called Go-Back-N (GBN).

Figure 23.23: Go-Back-N protocol



Sequence Number

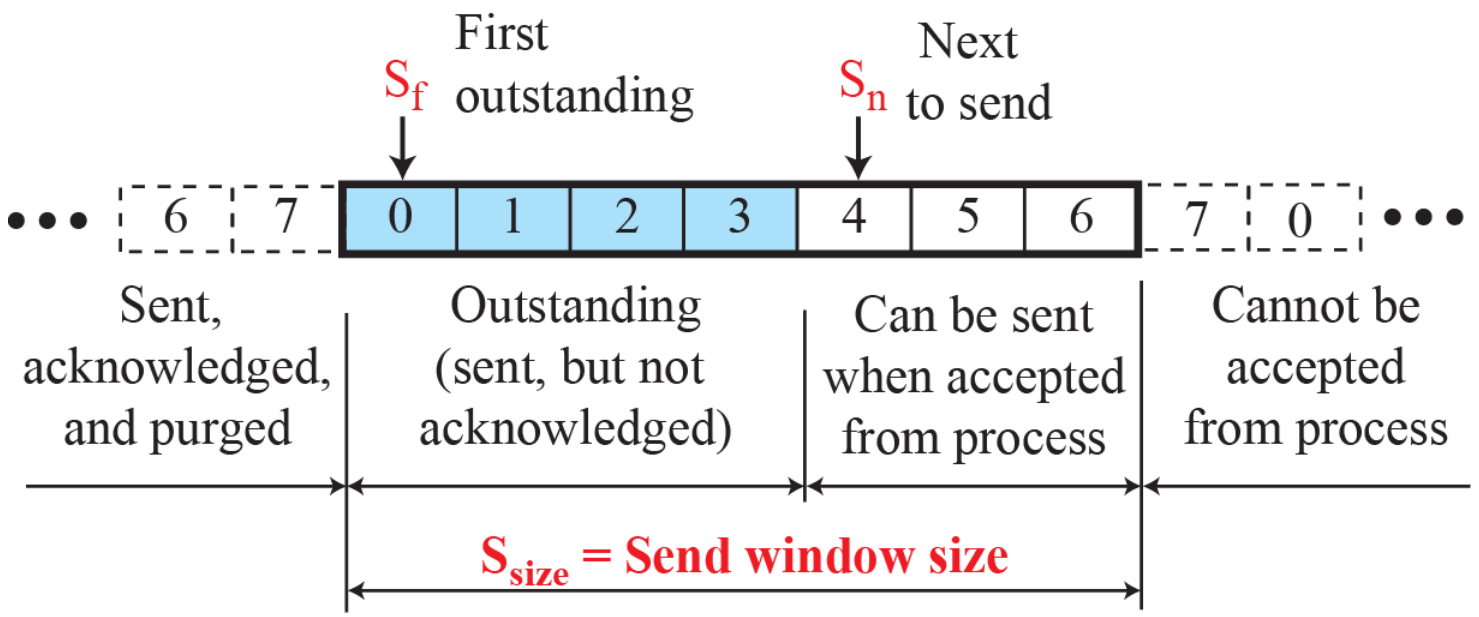
As we mentioned before, the sequence numbers are modulo 2^m, where m is the size of the sequence number field in bits.

Acknowledgement Number

An acknowledgment number in this protocol is cumulative and defines the sequence number of the next packet expected.

For example, if the acknowledgment number (ackNo) is 7, it means all packets with sequence number up to 6 have arrived, safe and sound, and the receiver is expecting the packet with sequence number 7.

Figure 23.24: Send window for Go-Back-N



Note

The send window is an abstract concept defining an imaginary box of size $2^m - 1$ with three variables: S_f, S_n, and S_{size}.

Figure 23.25: ***Sliding the send window***

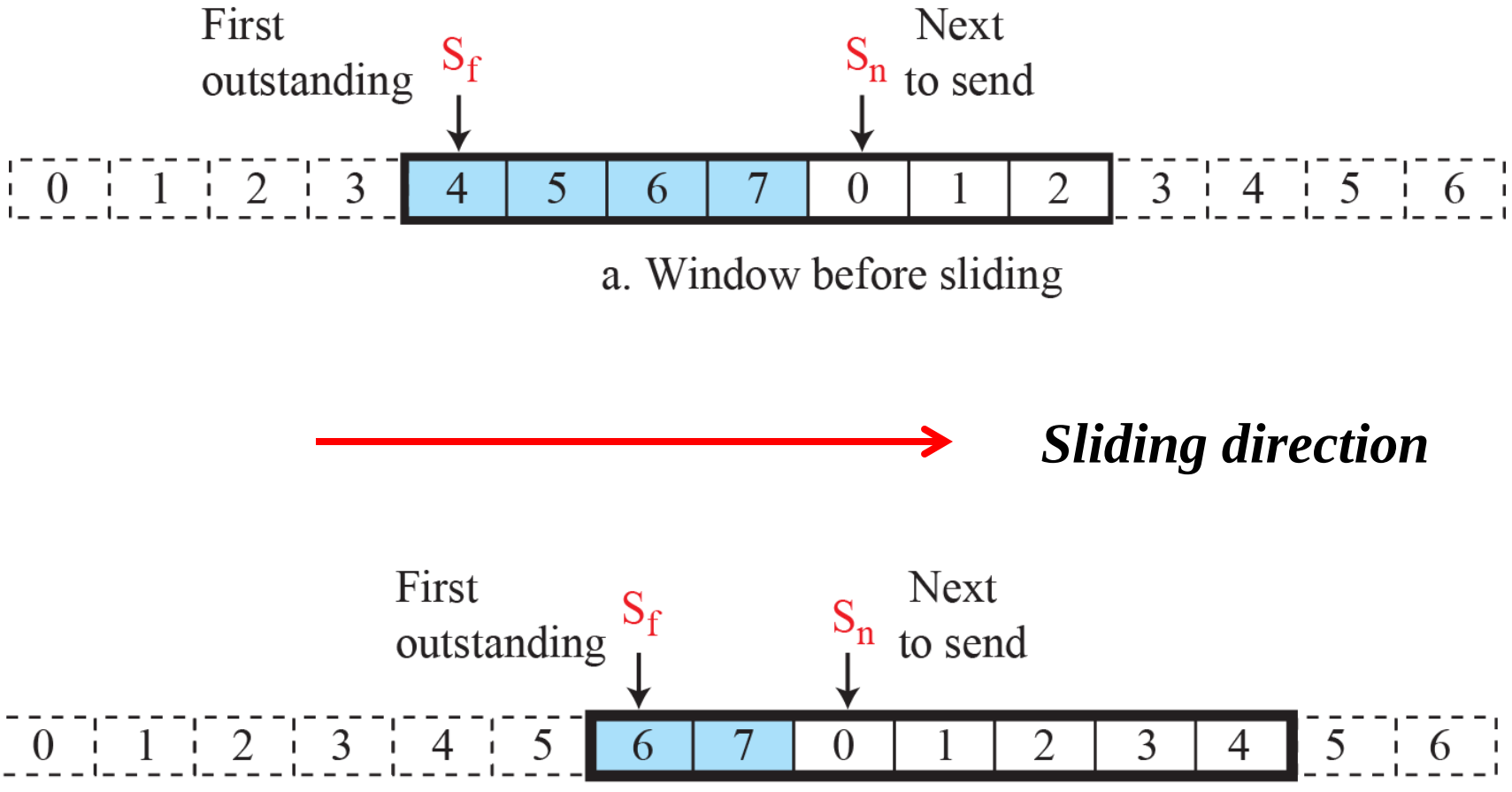
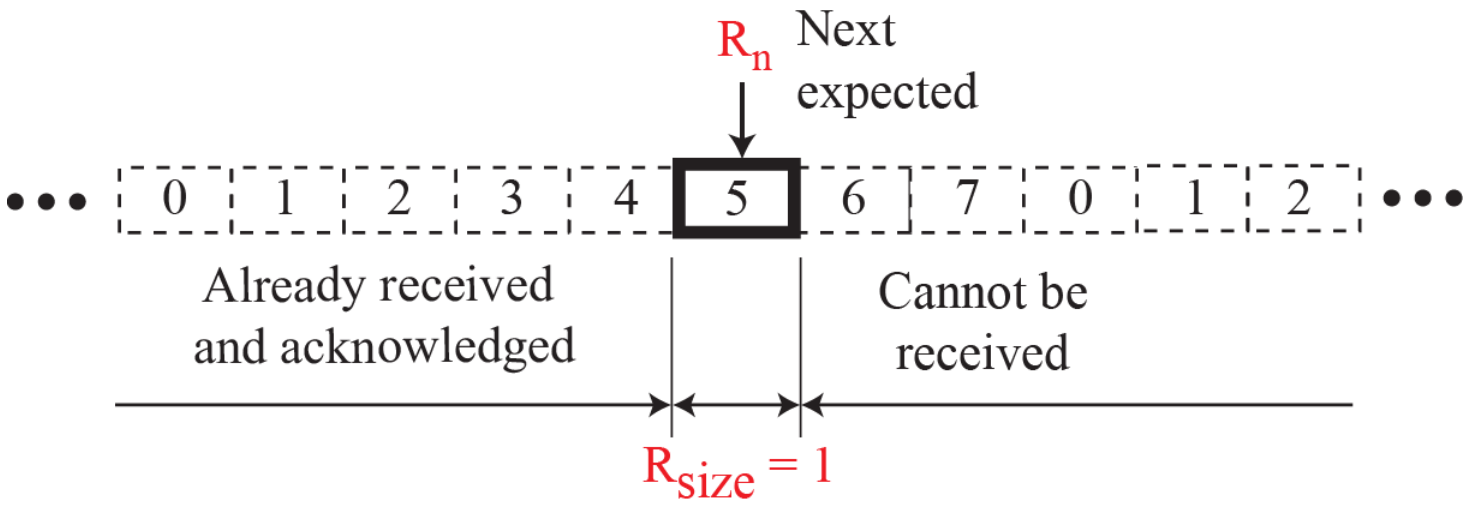


Figure 23.26: Receive window for Go-Back-N



Note

In Go-Back-N, the size of the send window must be less than 2^m; the size of the receiver window is always 1.

Note

Stop-and-Wait is a special case of Go-Back-N ARQ in which the size of the send window is 1.

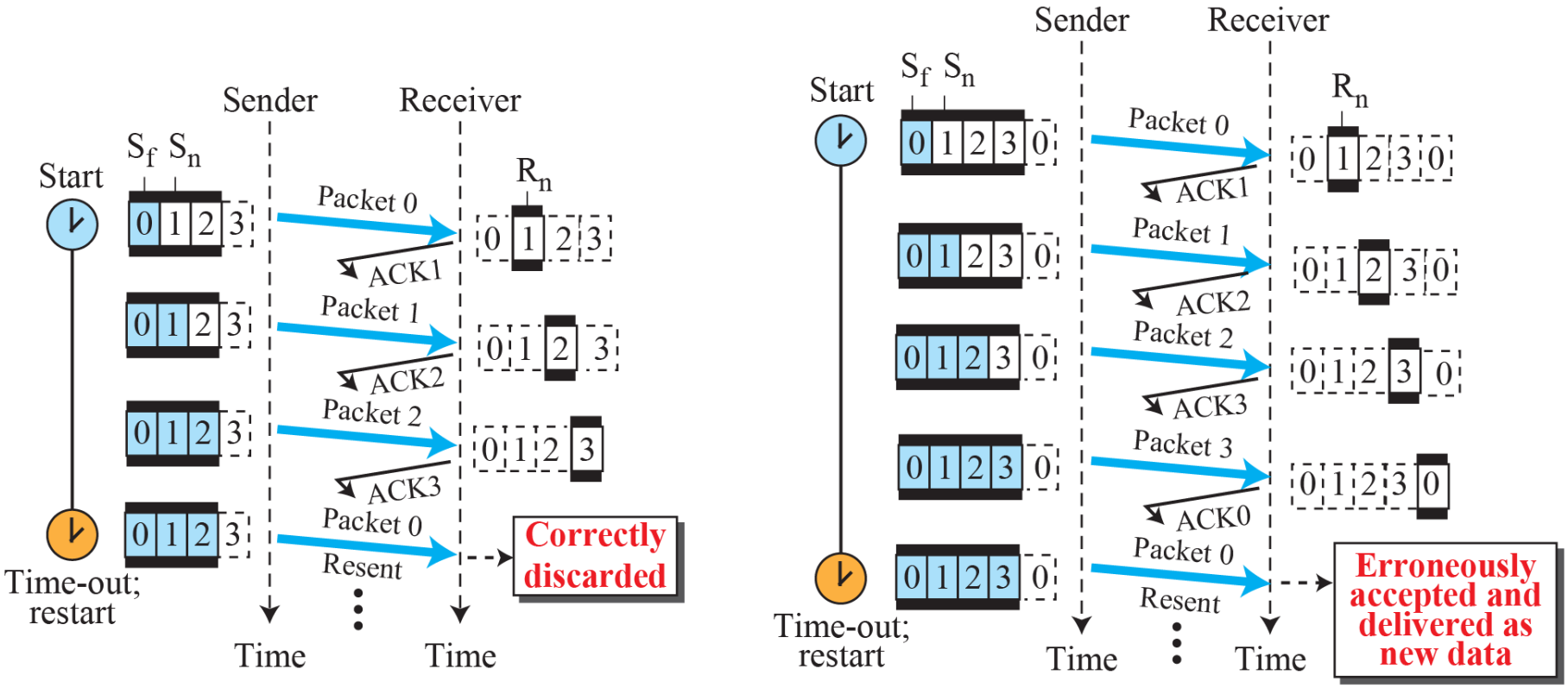
Timer

Although there can be a timer for each packet that is sent, in our protocol we use only one. The reason is that the timer for the first outstanding packet always expires first. We resend all outstanding packets when this timer expires.

Resending Packets

When the timer expires, the sender resends all outstanding packets. For example, suppose the sender has already sent packet 6 ($S_n = 7$), but the only timer expires. If $S_f = 3$, this means that packets 3, 4, 5, and 6 have not been acknowledged; the sender goes back and resends packets 3, 4, 5, and 6. That is why the protocol is called Go-Back-N. On a time-out, the machine goes back N locations and resends all packets.

Figure 23.28: Send window size for Go-Back-N



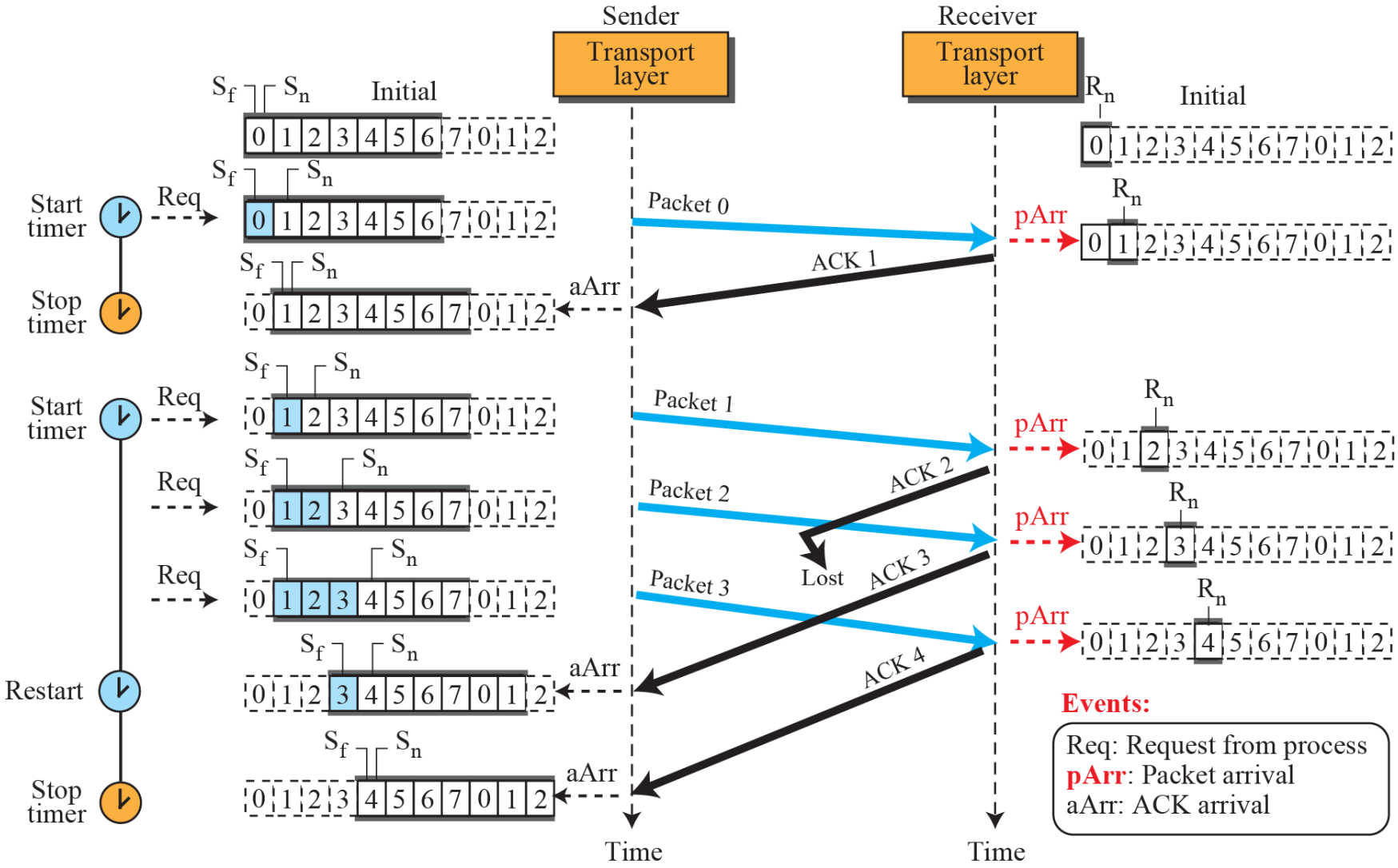
a. Send window of size $< 2^m$

b. Send window of size $= 2^m$

Example 23.7

Figure 23.29 shows an example of Go-Back-*N*. This is an example of a case where the forward channel is reliable, but the reverse is not. No data packets are lost, but some ACKs are delayed and one is lost. The example also shows how cumulative acknowledgments can help if acknowledgments are delayed or lost.

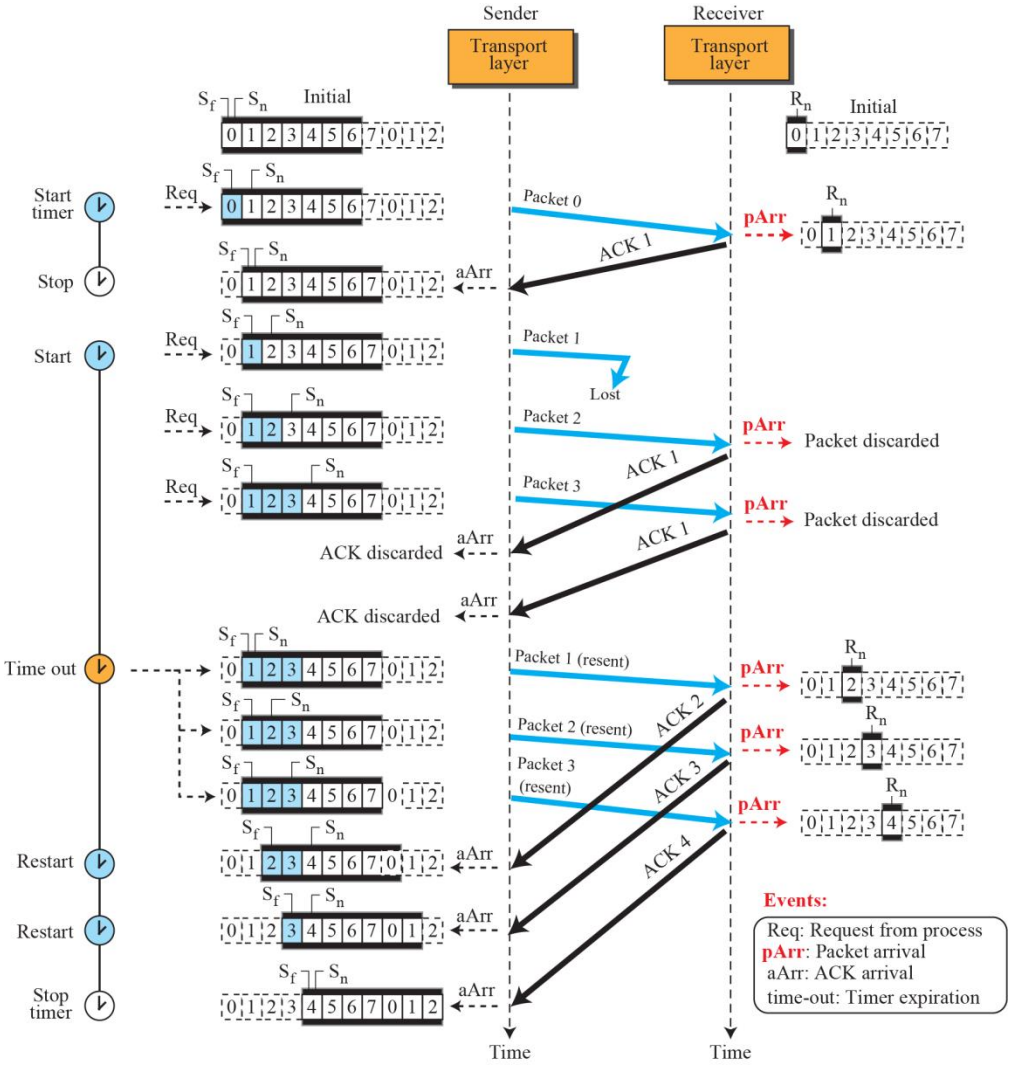
Figure 23.29: Flow diagram for Example 3.7



Example 23.8

Figure 23.30 shows what happens when a packet is lost. Packets 0, 1, 2, and 3 are sent. However, packet 1 is lost. The receiver receives packets 2 and 3, but they are discarded because they are received out of order (packet 1 is expected). When the receiver receives packets 2 and 3, it sends ACK1 to show that it expects to receive packet 23. However, these ACKs are not useful for the sender because the ackNo is equal to S_f, not greater that S_f . So the sender discards them. When the time-out occurs, the sender resends packets 1, 2, and 3, which are acknowledged.

Figure 23.30: Flow diagram for Example 3.8



Go-Back-N versus Stop-and-Wait

- The reader may find that there is a similarity between the Go-Back-N protocol and the Stop-and-Wait protocol.
- The Stop-and-Wait protocol is actually a Go-Back-N protocol in which there are only two sequence numbers and the send window size is 1.
- In other words, m = 1 and $2^m - 1 = 1$.
- In Go-Back-N, we said that the arithmetic is modulo 2^m
- In Stop-and-Wait it is modulo 2, which is the same as 2^m when m = 1.

Selective-Repeat Protocol

The Go-Back-N protocol simplifies the process at the receiver. The receiver keeps track of only one variable, and there is no need to buffer out-of-order packets; they are simply discarded. However, this protocol is inefficient if the underlying network protocol loses a lot of packets. Each time a single packet is lost or corrupted, the sender resends all outstanding packets, even though some of these packets may have been received safe and sound but out of order.

Figure 23.31: Outline of Selective-Repeat

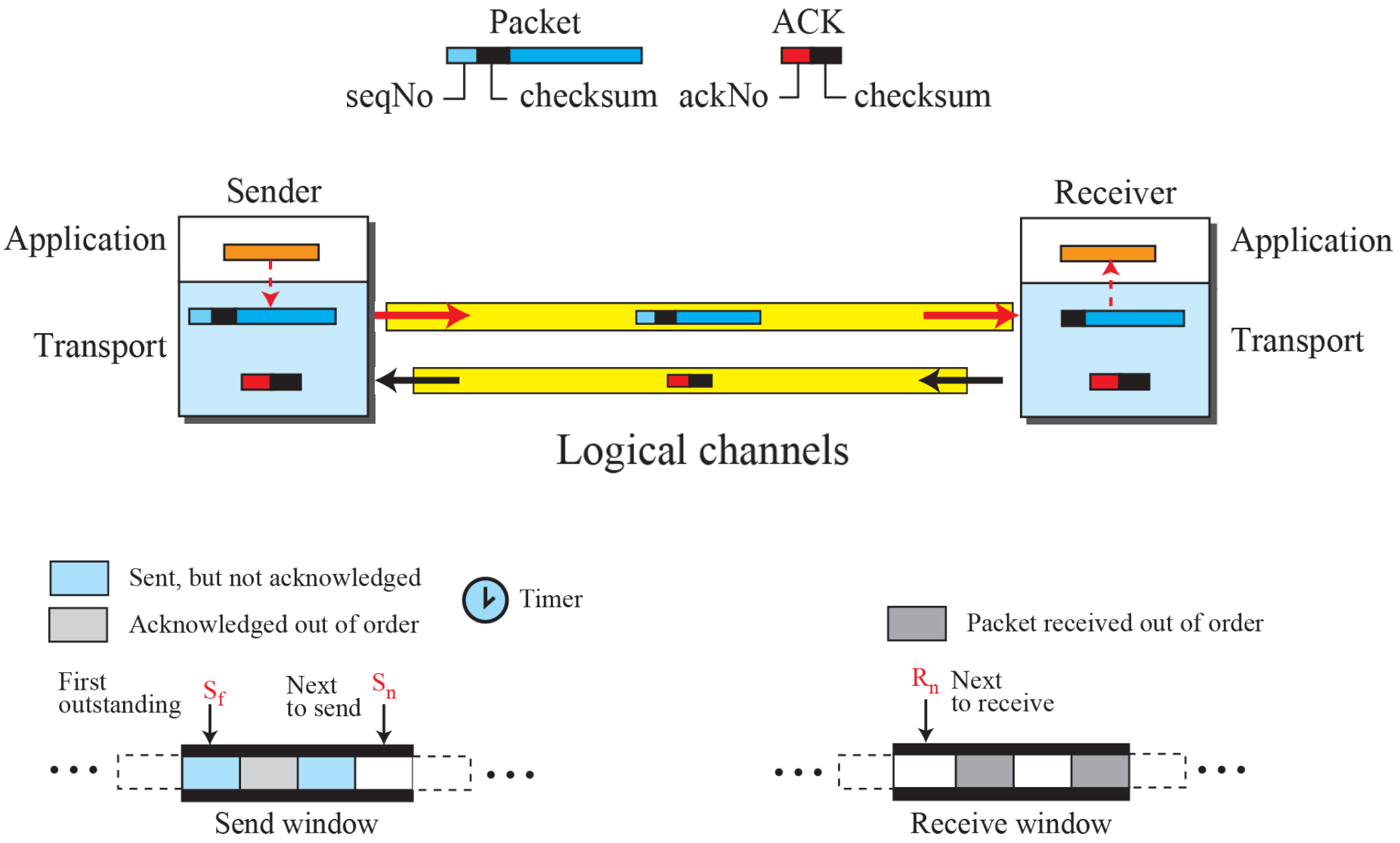


Figure 23.32: Send window for Selective-Repeat protocol

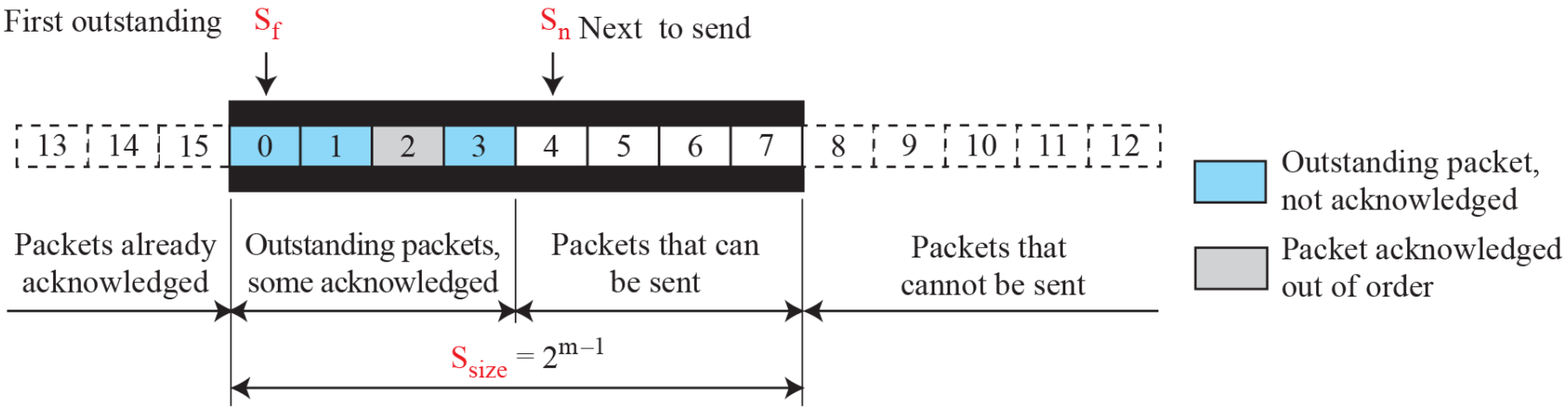
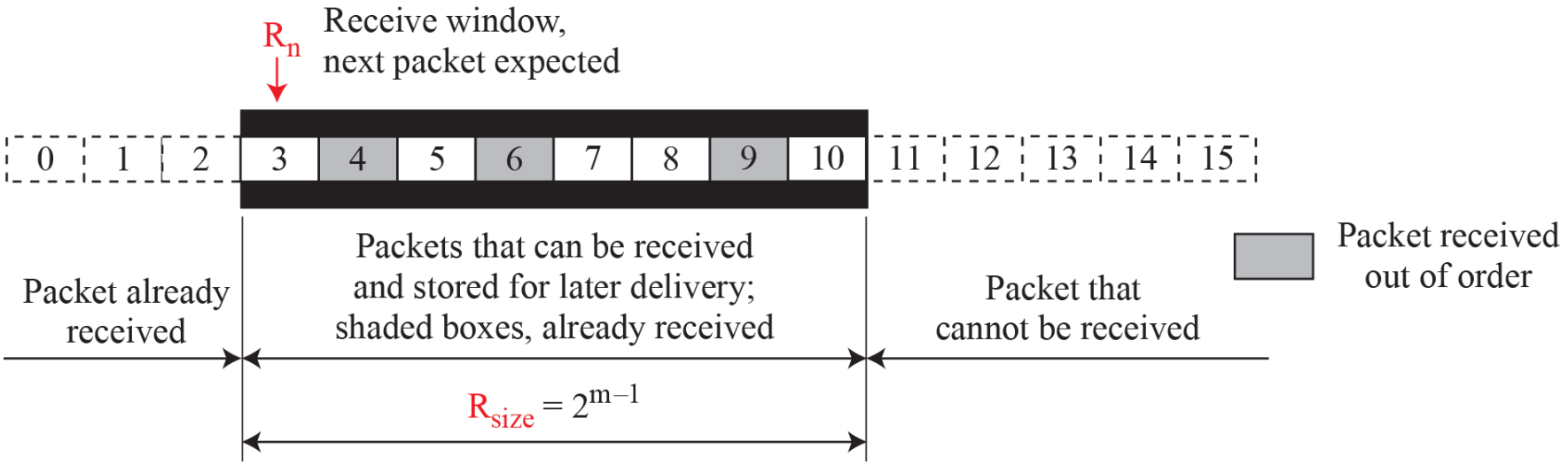


Figure 23.33: *Receive window for Selective-Repeat protocol*



Timer

Theoretically, Selective-Repeat uses one timer for each outstanding packet. When a timer expires, only the corresponding packet is resent. In other words, GBN treats outstanding packets as a group; SR treats them individually. However, most transport-layer protocols that implement SR use only a single timer. For this reason, we use only one timer.

Acknowledgments

In the Selective-Repeat protocol, an acknowledgment number defines the sequence number of the error-free packet received.

Example 23.9

Assume a sender sends 6 packets: packets 0, 1, 2, 3, 4, and 5. The sender receives an ACK with ackNo = 3. What is the interpretation if the system is using GBN or SR?

Solution

If the system is using GBN, it means that packets 0, 1, and 2 have been received uncorrupted and the receiver is expecting packet 3. If the system is using SR, it means that packet 3 has been received uncorrupted; the ACK does not say any thing about other packets.

Example 23.10

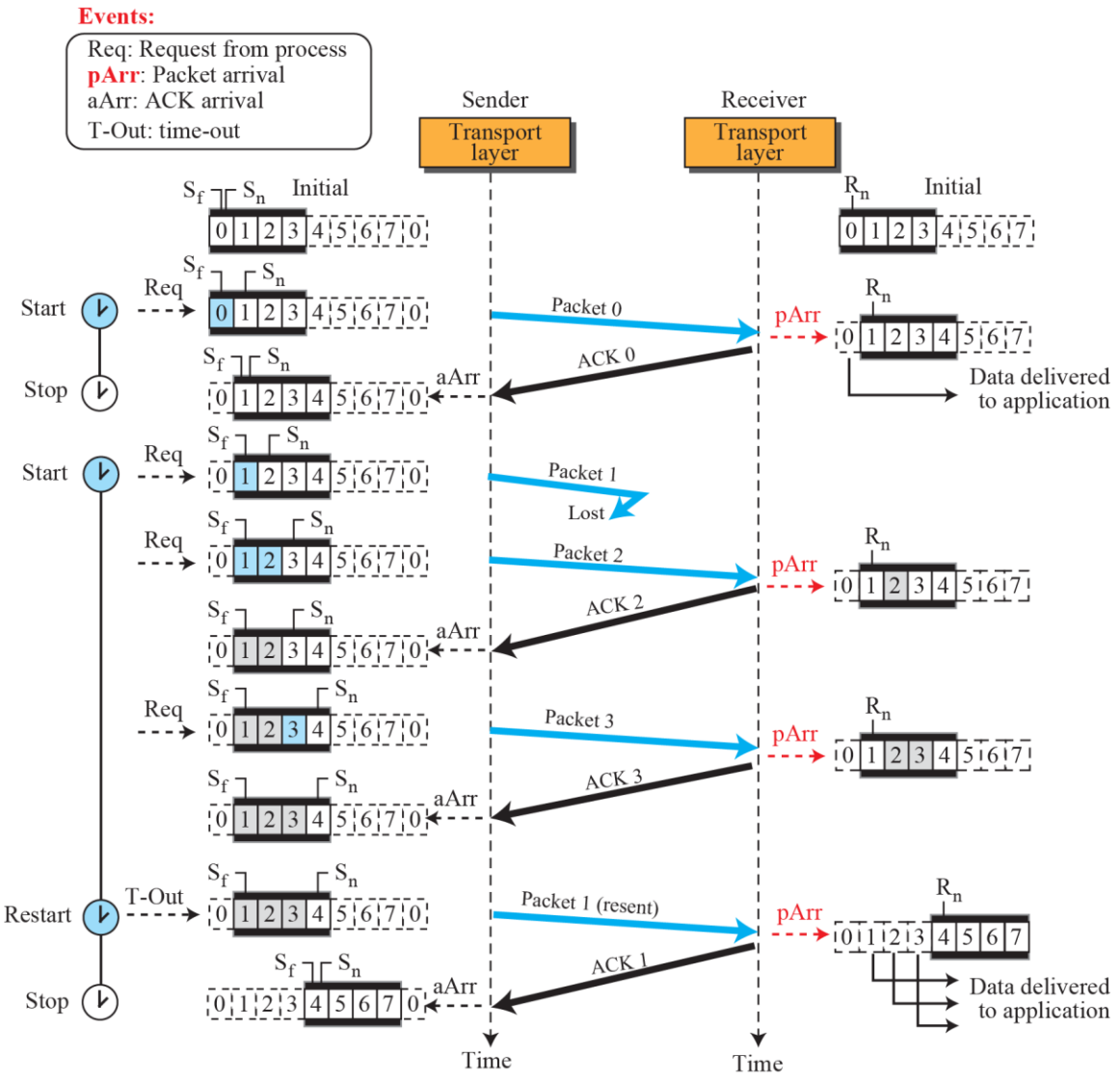
This example is similar to Example 23.8 (Figure 23.30) in which packet 1 is lost. We show how Selective-Repeat behaves in this case. Figure 3.35 shows the situation.

At the sender, packet 0 is transmitted and acknowledged. Packet 1 is lost. Packets 2 and 3 arrive out of order and are acknowledged. When the timer times out, packet 1 (the only unacknowledged packet) is resent and is acknowledged. The send window then slides.

Example 23.10 (continued)

At the receiver site we need to distinguish between the acceptance of a packet and its delivery to the application layer. At the second arrival, packet 2 arrives and is stored and marked (shaded slot), but it cannot be delivered because packet 1 is missing. At the next arrival, packet 3 arrives and is marked and stored, but still none of the packets can be delivered. Only at the last arrival, when finally a copy of packet 1 arrives, can packets 1, 2, and 3 be delivered to the application layer. There are two conditions for the delivery of packets to the application layer: First, a set of consecutive packets must have arrived. Second, the set starts from the beginning of the window.

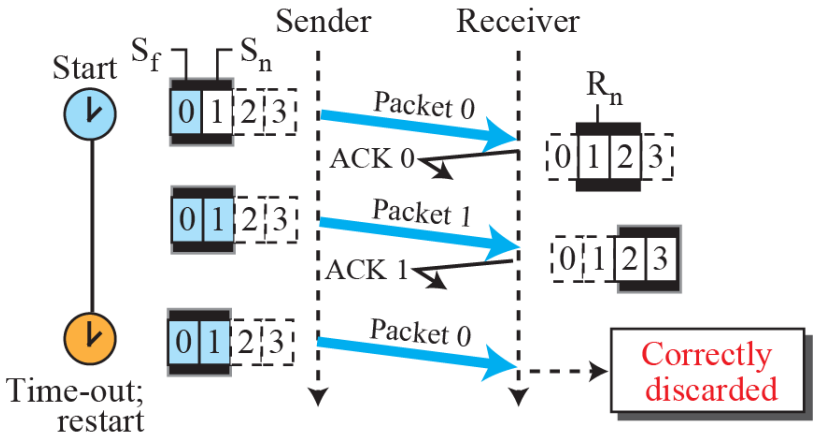
Figure 23.35: Flow diagram for Example 3.10



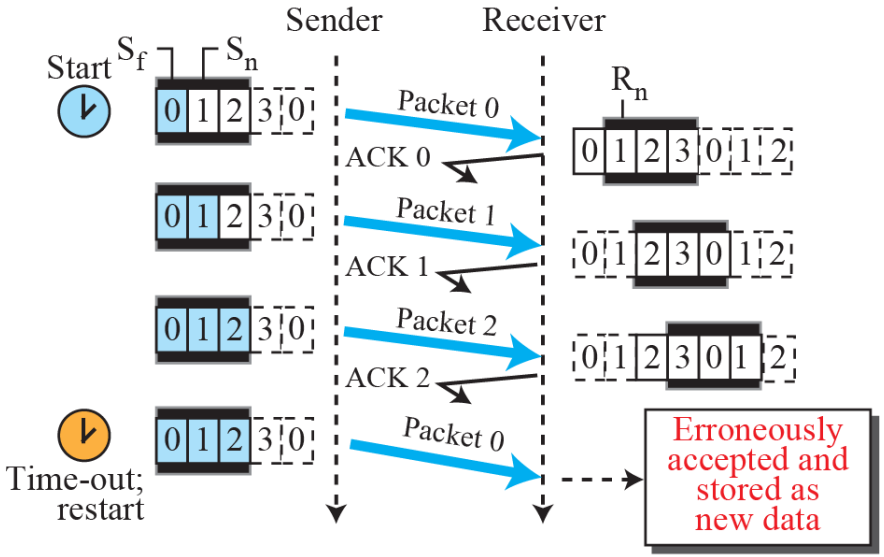
Note

In Selective Repeat, the size of the sender and receiver window must be at most one-half of 2^m.

Figure 23.36: Selective-Repeat, window size

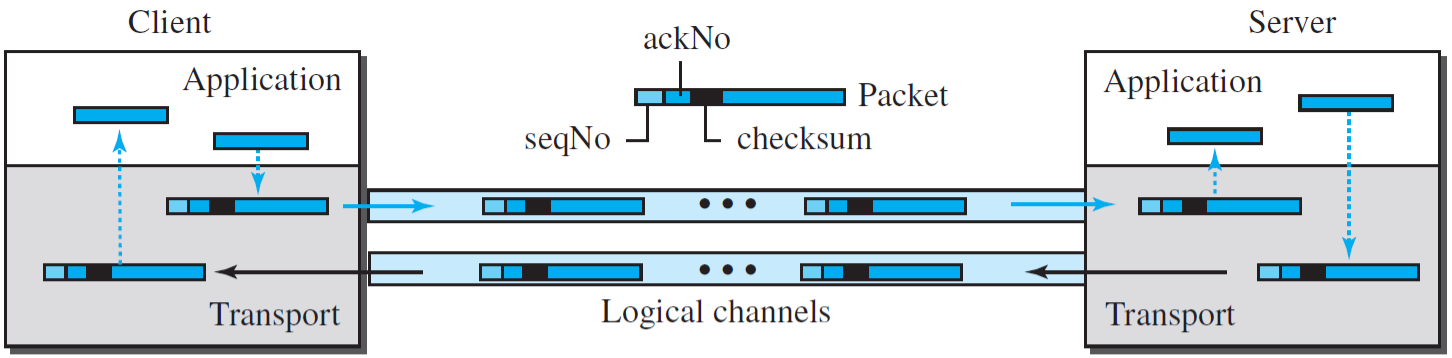


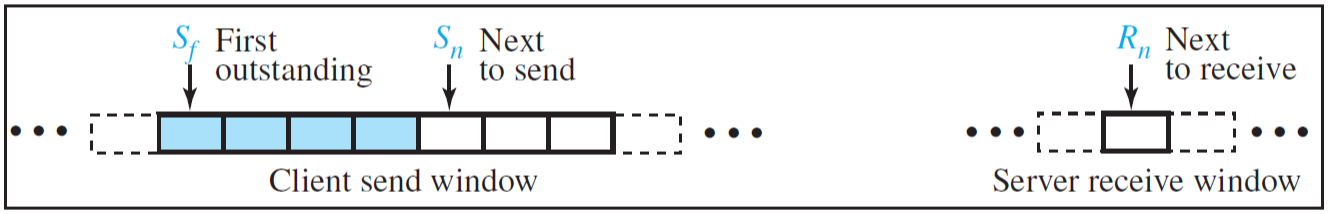
a. Send and receive windows of size = 2^{m-1}



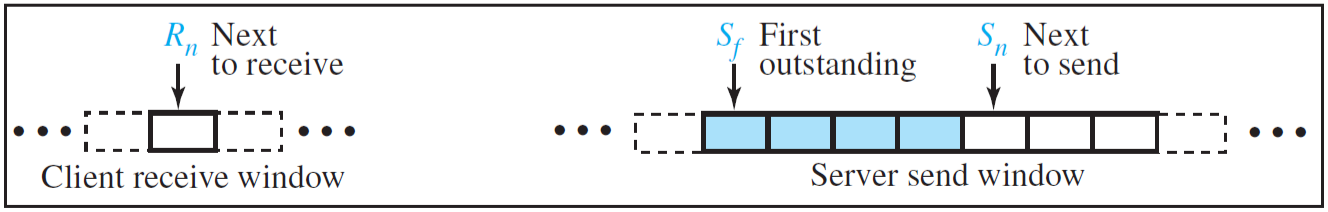
b. Send and receive windows of size > 2^{m-1}

Bidirectional Protocols: Piggybacking





Windows for communication from client to server



Windows for communication from server to client