

# Chapter 20: Unicast Routing

## *Outline*

***20.1 INTRODUCTION***

***20.2 ROUTING ALGORITHMS***

## 20-1 INTRODUCTION

*Unicast routing in the Internet, with a large number of routers and a huge number of hosts, can be done only by using hierarchical routing: routing in several steps using different routing algorithms.*

*In this chapter, we discuss the general concept and algorithms of unicast routing.*

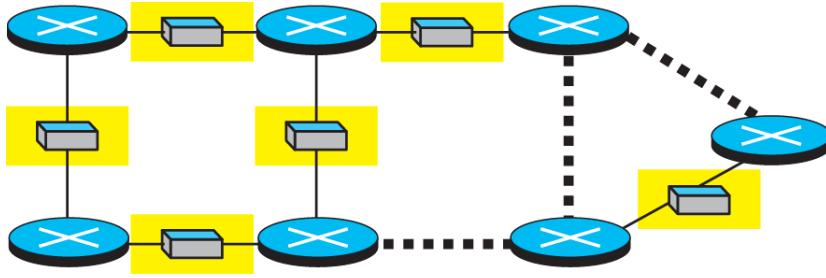
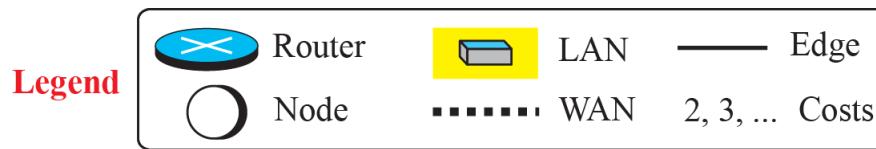
## **20.1.1 General Idea**

*In unicast routing, a packet is routed, hop by hop, from its source to its destination with the use of forwarding tables.*

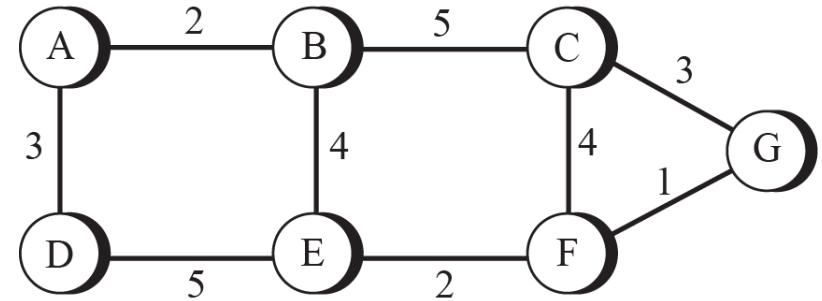
*To find the best route, an internet can be modeled as a graph. A graph is a set of nodes and edges that connect the nodes. To model an internet as a graph, we can represent each router as a node and each network between a pair of routers as an edge.*

*In fact, an internet can be modeled as a weighted graph, in which each edge is associated with a cost. If there is no edge between the nodes, the cost is infinity.*

## 20.1.2 Least-Cost Routing



a. An internet



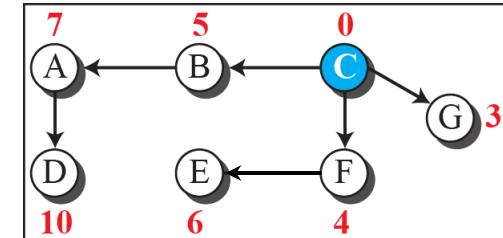
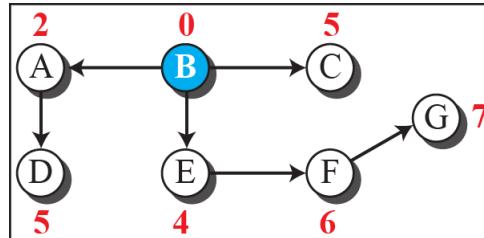
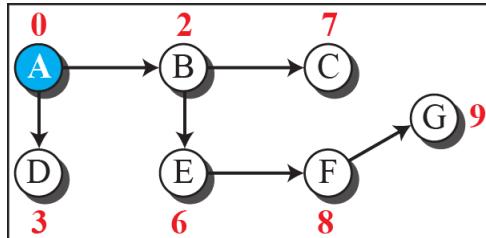
b. The weighted graph

*One of the ways to determine the best route from the source router to the destination router is to find the least cost between the two in the weighted graph representation.*

*In other words, the source router chooses a route to the destination router in such a way that the total cost for the route is the least cost among all possible routes.*

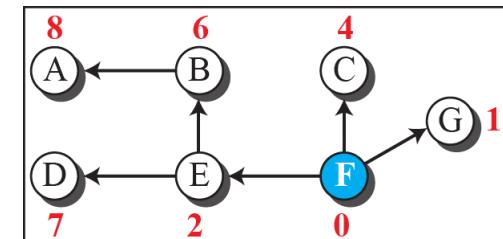
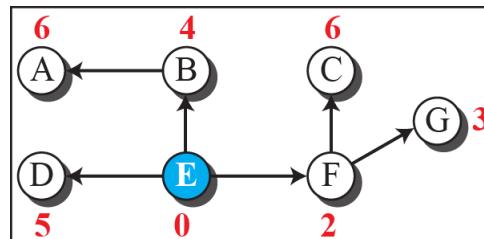
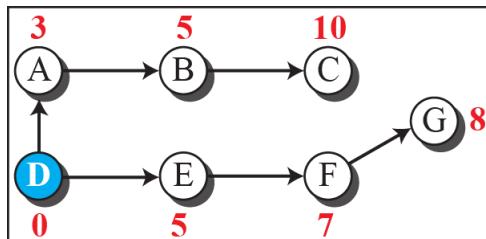
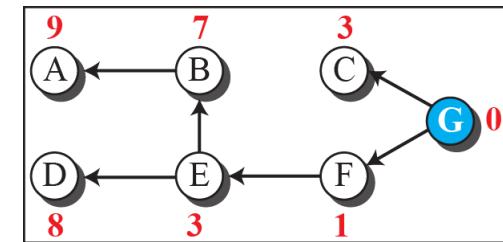
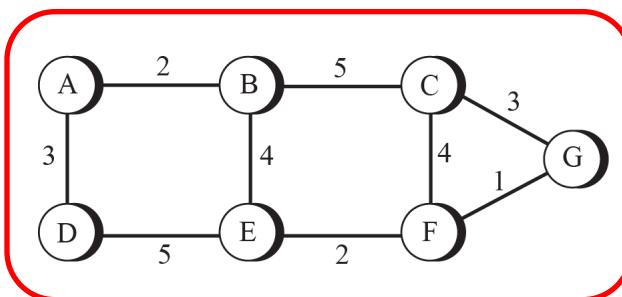
## Figure 20.2: Least-cost trees

A least-cost tree is a tree with the source router as the root that spans the whole graph (visits all other nodes) and in which the path between the root and any other node is the shortest. Note, if there are  $N$  routers in an internet, there are  $N$  least-cost trees (one shortest-path tree for each node).



### Legend

- Root of the tree
- Intermediate or end node
- Total cost from the root



### Properties:

- (1) The cost of the least-cost routes is the same in both directions (from inverse trees), i.e.,  $[A \rightarrow G \text{ (using least-cost tree A)}] = [G \rightarrow A \text{ (using least-cost tree G)}] = 9$ .
- (2) The cost of using a combination of routes with different least-cost trees is the same as using a single least-cost tree, i.e.,  $[A \rightarrow E \text{ (using least-cost tree A)} + E \rightarrow G \text{ (using least-cost tree E)}] = [A \rightarrow G \text{ (using least-cost tree A)}] = 9$ .

## 20-2 ROUTING ALGORITHMS

**Several routing algorithms have been designed in the past. The differences between these methods are in the way they interpret the least cost and the way they create the least-cost tree for each node.**

**In this section, we discuss distance-vector routing and link-state routing.**

## **20.2.1 Distance-Vector Routing**

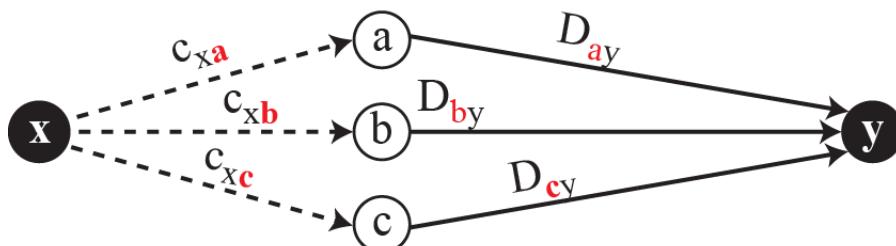
*In distance-vector routing, the first thing each node creates is its own least-cost tree with the rudimentary information it has about its immediate neighbors. The incomplete trees are exchanged between immediate neighbors to make the trees more and more complete and to represent the whole internet.*

*In distance-vector routing, a router continuously tells all of its neighbors what it knows about the whole internet (although the knowledge can be incomplete).*

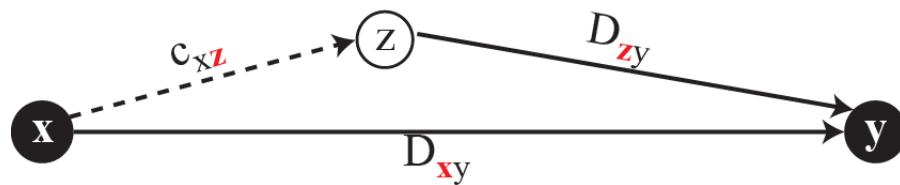
## Figure 20.3: Graphical idea behind Bellman-Ford equation

At the heart of distance vector routing is the Bellman-Ford equation. This equation is used to find the least cost between a source node,  $x$ , and a destination node,  $y$ , through some intermediary nodes ( $a, b, c, \dots$ ) where the costs between the source and the intermediary nodes and the least costs between the intermediary nodes and the destination are given.

The following shows the general case in which  $D_{ij}$  is the shortest distance and  $c_{ij}$  is the cost between nodes  $i$  and  $j$ .



a. General case with three intermediate nodes



b. Updating a path with a new route

$$D_{xy} = \min\{(c_{xa} + D_{ay}), (c_{xb} + D_{by}), (c_{xc} + D_{cy}), \dots\}$$

$$D_{xy} = \min\{D_{xy}, (c_{xz} + D_{zy})\}$$

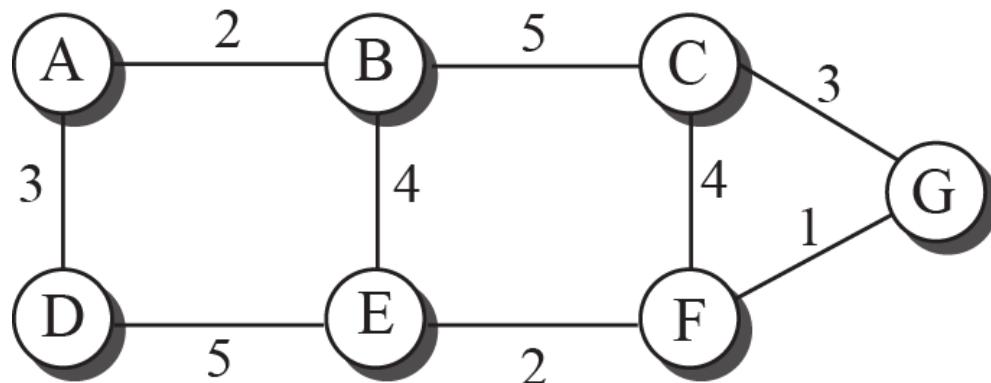
**Figure 20.5: Distance vectors (one-dimensional arrays)**

A	0
B	2
C	$\infty$
D	3
E	$\infty$
F	$\infty$
G	$\infty$

A	2
B	0
C	5
D	$\infty$
E	4
F	$\infty$
G	$\infty$

A	$\infty$
B	5
C	0
D	$\infty$
E	$\infty$
F	4
G	3

Note that the initial distance vector for each node is populated with immediate neighbors only.

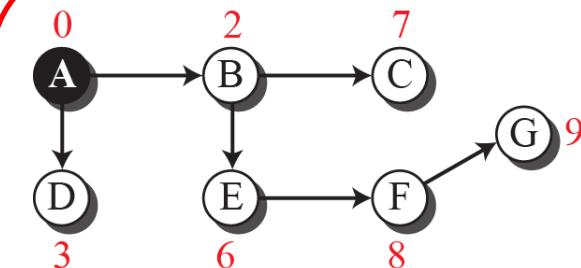


A	3
B	$\infty$
C	$\infty$
D	0
E	5
F	$\infty$
G	$\infty$

A	$\infty$
B	4
C	$\infty$
D	5
E	0
F	2
G	$\infty$

A	$\infty$
B	$\infty$
C	4
D	$\infty$
E	2
F	0
G	1

Distance Vector for Node A



a. Tree for node A (Least Cost)

A	$\infty$
B	$\infty$
C	3
D	$\infty$
E	$\infty$
F	1
G	0

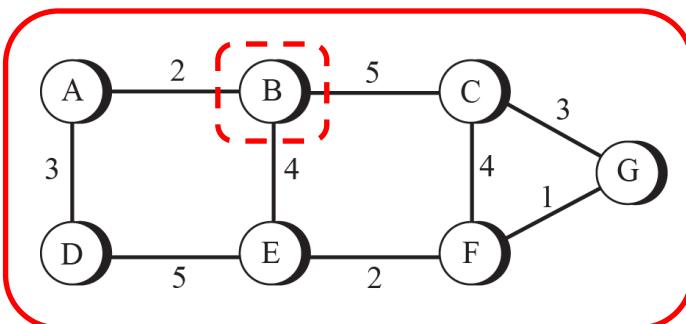
Name of the distance vector defines the root

Indexes of the distance vector defines the destinations

Value of each cell defines the least cost from the root to the destination

b. Distance vector for node A

## Figure 20.6: Updating distance vectors



New B      Old B      A

	A	B	C	D	E	F	G
A	2						
B	0	2					
C	5	5	5				
D	5	8	8	3			
E	4	4	4	∞	∞		
F	∞	∞	∞	∞	6		
G	∞	∞	∞	∞	∞	∞	

$B[ ] = \min(B[ ], 2 + A[ ])$

New B      Old B      E

	A	B	C	D	E	F	G
A	2						
B	0	2					
C	5	5	5				
D	5	5	5	5			
E	4	4	4	0	4	2	
F	∞	∞	∞	∞	∞	2	
G	∞	∞	∞	∞	∞	∞	

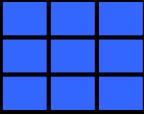
$B[ ] = \min(B[ ], 4 + E[ ])$

a. First event: B receives a copy of A's vector.

b. Second event: B receives a copy of E's vector.

**Note:**  
 $X[ ]$ : the whole vector

After each node is updated, it immediately sends its updated vector to all neighbors.



## Table 20.1: Distance-Vector Routing Algorithm

```
1 Distance_Vector_Routing ( )
2 {
3     // Initialize (create initial vectors for the node)
4     D[myself] = 0
5     for (y = 1 to N)
6     {
7         if (y is a neighbor)
8             D[y] = c[myself][y]
9         else
10            D[y] = ∞
11    }
12    send vector {D[1], D[2], ..., D[N]} to all neighbors
13    // Update (improve the vector with the vector received from a neighbor)
14    repeat (forever)
15    {
16        wait (for a vector Dw from a neighbor w or any change in the link)
17        for (y = 1 to N)
18        {
19            D[y] = min [D[y], (c[myself][w] + Dw[y])]      // Bellman-Ford equation
20        }
21        if (any change in the vector)
22            send vector {D[1], D[2], ..., D[N]} to all neighbors
23    }
24 } // End of Distance Vector
```

## **20.2.2 Link-State Routing**

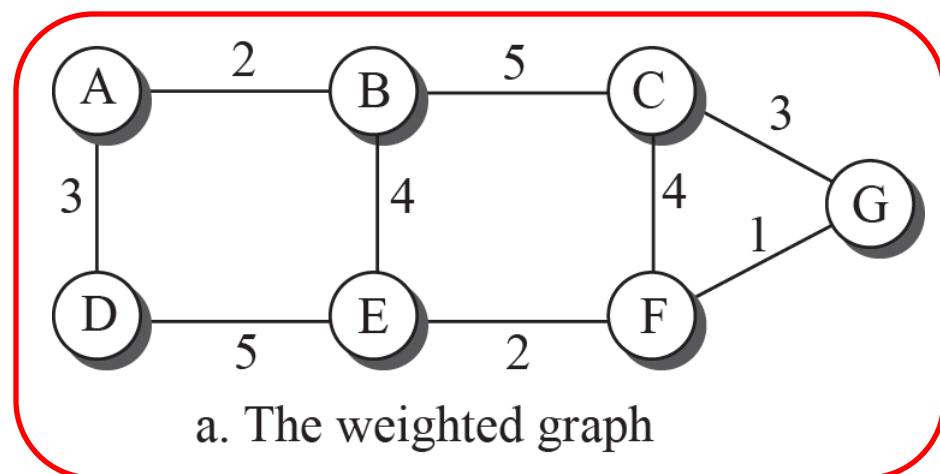
*In link-state routing, the term link-state is used to define the characteristic of a link (an edge) that represents a network in the internet.*

*In this algorithm, the cost associated with an edge defines the state of the link. Links with lower costs are preferred to links with higher costs; if the cost of a link is infinity, it means that the link does not exist or has been broken.*

*In link-state routing, each router tells the whole internet what it knows about its neighbors.*

**Figure 20.8: Example of a link-state database**

To create a least-cost tree, each node needs to have a complete map of the network. The collection of states for all links is called the link-state database (LSDB), represented as a two-dimensional array (matrix). There is only one LSDB for the whole internet; each node needs to have a duplicate of it to be able to create the least-cost tree.

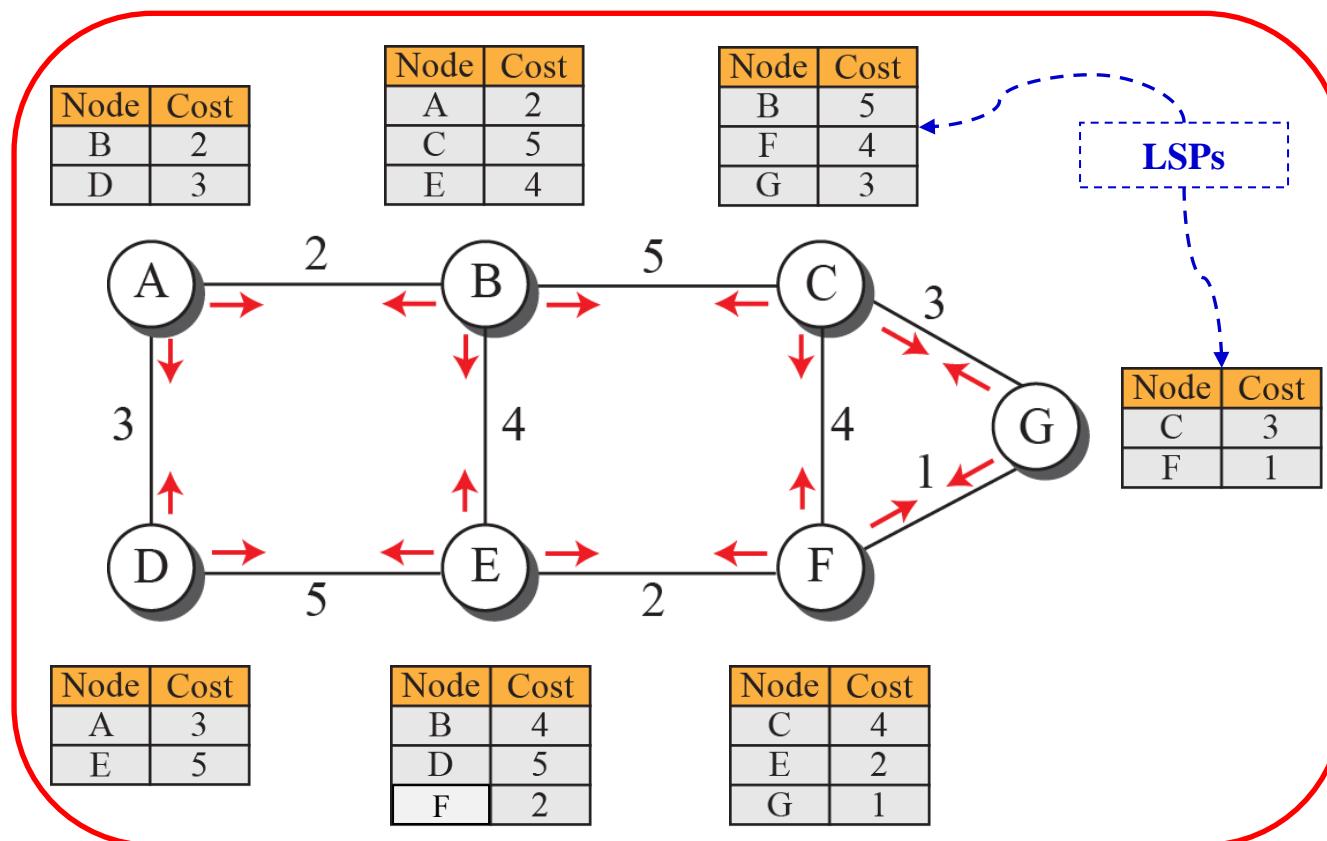


	A	B	C	D	E	F	G
A	0	2	$\infty$	3	$\infty$	$\infty$	$\infty$
B	2	0	5	$\infty$	4	$\infty$	$\infty$
C	$\infty$	5	0	$\infty$	$\infty$	4	3
D	3	$\infty$	$\infty$	0	5	$\infty$	$\infty$
E	$\infty$	4	$\infty$	5	0	2	$\infty$
F	$\infty$	$\infty$	4	$\infty$	2	0	1
G	$\infty$	$\infty$	3	$\infty$	$\infty$	1	0

b. Link state database

## Figure 20.9: LSPs created and sent out by each node to build LSDB

The LSDB is created by a process called flooding. Each node requests and collects two pieces of information in a link-state packet (LSP) from its immediate neighboring nodes: the identity of the node and the cost of the link. When a node receives a LSP on one of its interfaces, it compares it with the copy it may already have. If it's newer, it keeps the newer copy and then sends it out to each of its interfaces except the one from which the LSP arrived. After receiving all new LSPs, each node creates the comprehensive LSDB.

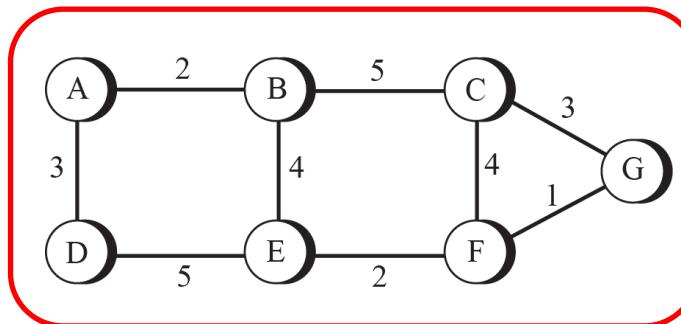
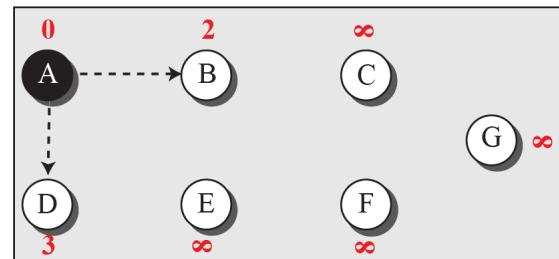


## Figure 20.10: Least-cost tree

To create the least-cost tree for itself, each node needs to run the iterative Dijkstra's algorithm:

- (1) The node (e.g., Node A below) chooses itself as the root of the tree, creating a tree with a single node, and sets the total cost of each node based on the information in the LSDB.
- (2) The node selects one node, among all nodes not in the tree, which is closest to the root, and adds this to the tree. After this node is added to the tree, the cost of all other nodes not in the tree needs to be updated because the paths may have been changed.
- (3) The node repeats step 2 until all nodes are added to the tree.

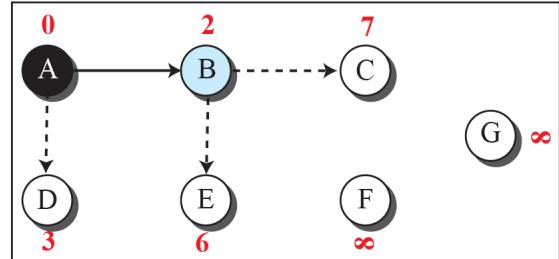
Initialization



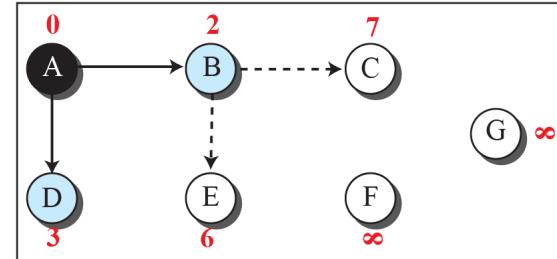
### Legend

<span style="background-color: black; border-radius: 50%; width: 15px; height: 15px;"></span>	Root node
<span style="background-color: lightblue; border-radius: 50%; width: 15px; height: 15px;"></span>	Node in the path
<span style="background-color: white; border-radius: 50%; width: 15px; height: 15px;"></span>	Node not yet in the path
<span style="border-top: 1px dashed black; border-bottom: 1px solid black; width: 15px; height: 15px;"></span>	Potential path
<span style="border: 1px solid black; width: 15px; height: 15px;"></span>	Path

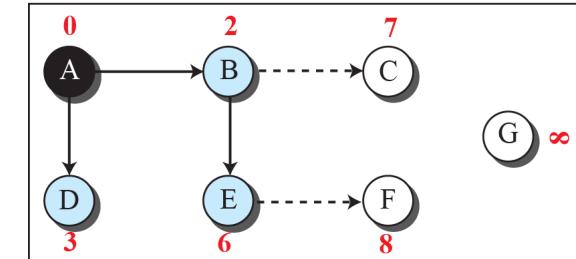
Iteration 1



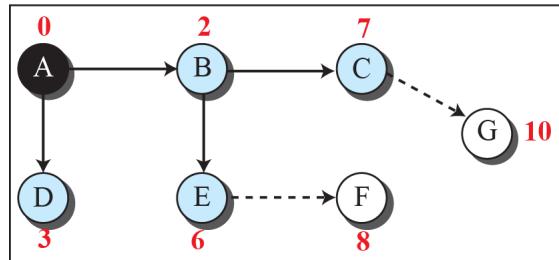
Iteration 2



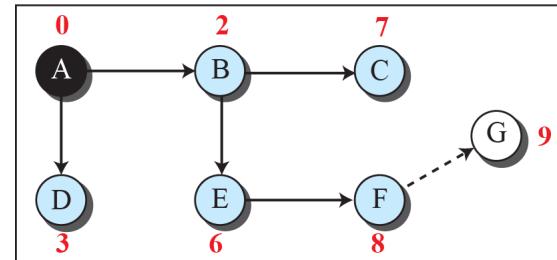
Iteration 3



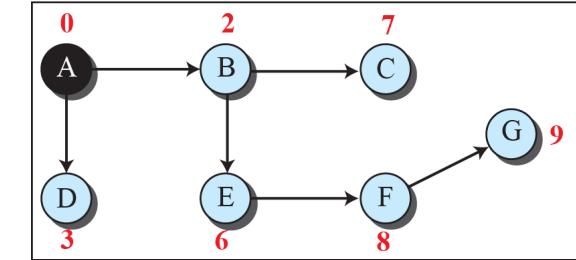
Iteration 4



Iteration 5



Iteration 6



## Table 20.2: Dijkstra's Algorithm

```
1 Dijkstra's Algorithm ( )
2 {
3     // Initialization
4     Tree = {root}                                // Tree is made only of the root
5     for (y = 1 to N)                            // N is the number of nodes
6     {
7         if (y is the root)
8             D[y] = 0                                // D[y] is shortest distance from root to node y
9         else if (y is a neighbor)
10            D[y] = c[root][y]                      // c[x][y] is cost between nodes x and y in LSDB
11        else
12            D[y] = ∞
13    }
14    // Calculation
15    repeat
16    {
17        find a node w, with D[w] minimum among all nodes not in the Tree
18        Tree = Tree ∪ {w}                          // Add w to tree
19        // Update distances for all neighbor of w
20        for (every node x, which is neighbor of w and not in the Tree)
21        {
22            D[x] = min{D[x], (D[w] + c[w][x])}
23        }
24    } until (all nodes included in the Tree)
25 } // End of Dijkstra
```

# Chapter 24: Transport-Layer

## Protocols

### *Outline*

**24.1 INTRODUCTION**

**24.2 UDP**

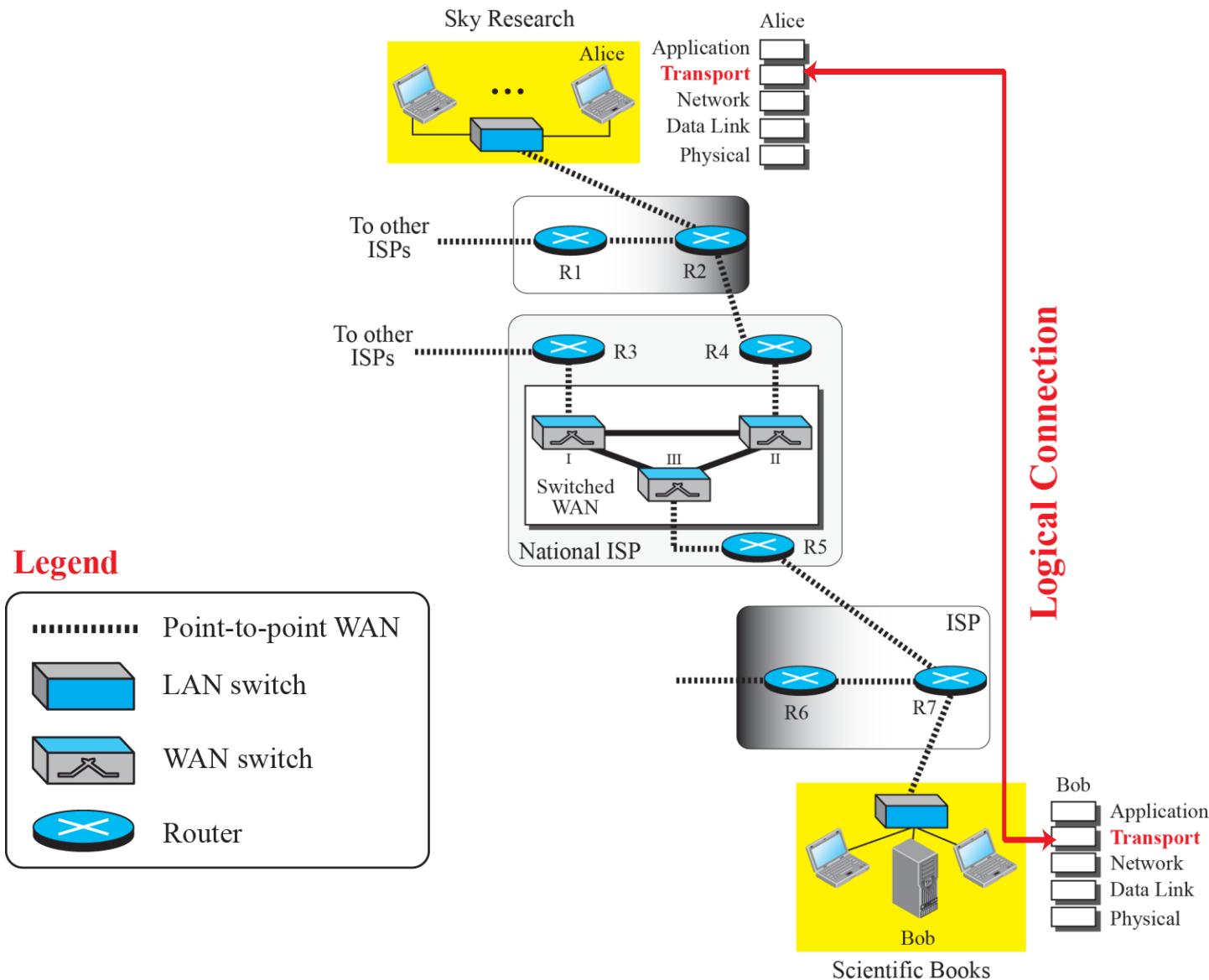
**24.3 TCP**

## 24-1 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: Logical connection at the transport layer**



## 23.1.1 Transport-Layer Services

The transport layer is located between the network layer and the application layer. It is responsible for providing services to the application layer and receives services from the network layer.

**Process-to-Process Communication:** as opposed to host-to-host communication in the network-layer, a process is an application-layer entity (running program).

**Encapsulation and Decapsulation:** process sends message to the transport layer and the sender adds the transport-layer header (encapsulation). Packets at the transport layer are called user datagrams / segments. Decapsulation happens at the receiver.

**Flow Control:** balance between production and consumption rates.

**Error Control:** the transport layer is responsible for (a) detecting and discarding corrupted packets, (b) keeping track of lost and discarded packets and resending them, (c) recognizing duplicate packets and discarding them and (d) buffering out-of-order packets until missing packets arrive.

**Congestion Control:** implements mechanisms and techniques to keep the load below the capacity.

## 24.1.2 Port Numbers

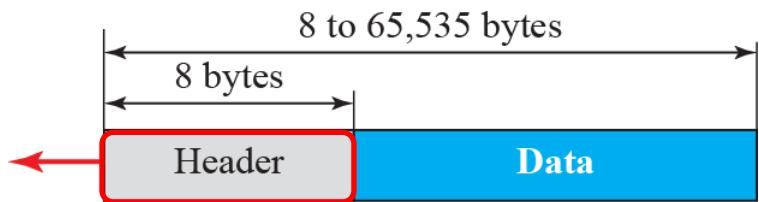
The transport-layer protocol has several responsibilities. One is to create a process-to-process communication: the protocols use port numbers to accomplish this. Port numbers provide end-to-end addresses at the transport layer and allow multiplexing and demultiplexing at this layer, just as IP addresses do at the network layer.

Port	Protocol	UDP	TCP	Description
7	Echo	√		Echoes back a received datagram
9	Discard	√		Discards any datagram that is received
11	Users	√	√	Active users
13	Daytime	√	√	Returns the date and the time
17	Quote	√	√	Returns a quote of the day
19	Chargen	√	√	Returns a string of characters
20, 21	FTP		√	File Transfer Protocol
23	TELNET		√	Terminal Network
25	SMTP		√	Simple Mail Transfer Protocol
53	DNS	√	√	Domain Name Service
67	DHCP	√	√	Dynamic Host Configuration Protocol
69	TFTP	√		Trivial File Transfer Protocol
80	HTTP		√	Hypertext Transfer Protocol
111	RPC	√	√	Remote Procedure Call
123	NTP	√	√	Network Time Protocol
161, 162	SNMP		√	Simple Network Management Protocol

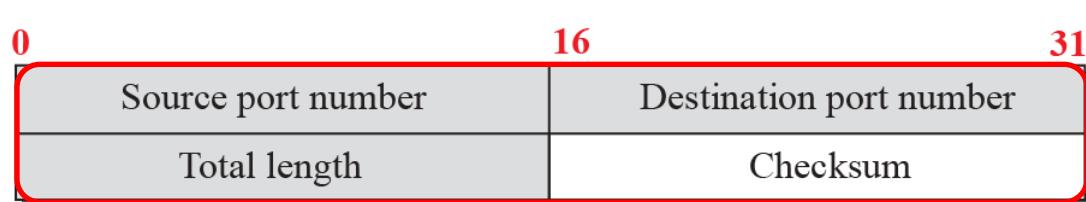
## 24-2 UDP

*The User Datagram Protocol (UDP) is a connectionless, unreliable transport protocol. However, UDP is a very simple and efficient protocol using a minimum of overhead.*

*UDP packets, called user datagrams, have a fixed-size header of 8 bytes made of four fields, each of 2 bytes. The first two fields define the source and destination port numbers, the third field defines the total length of the user datagram, header plus data. The last field can carry the optional checksum.*



a. UDP user datagram

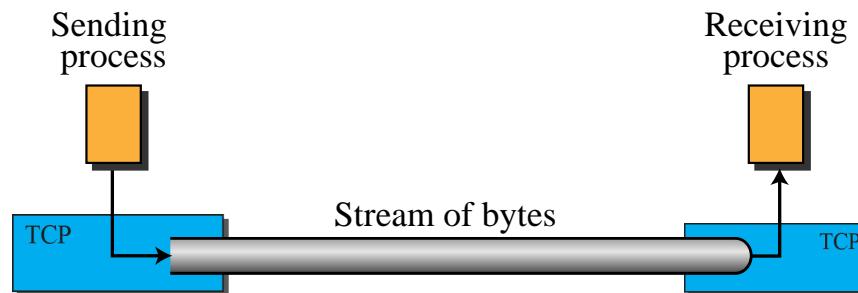


b. Header format

## 24-3 TCP

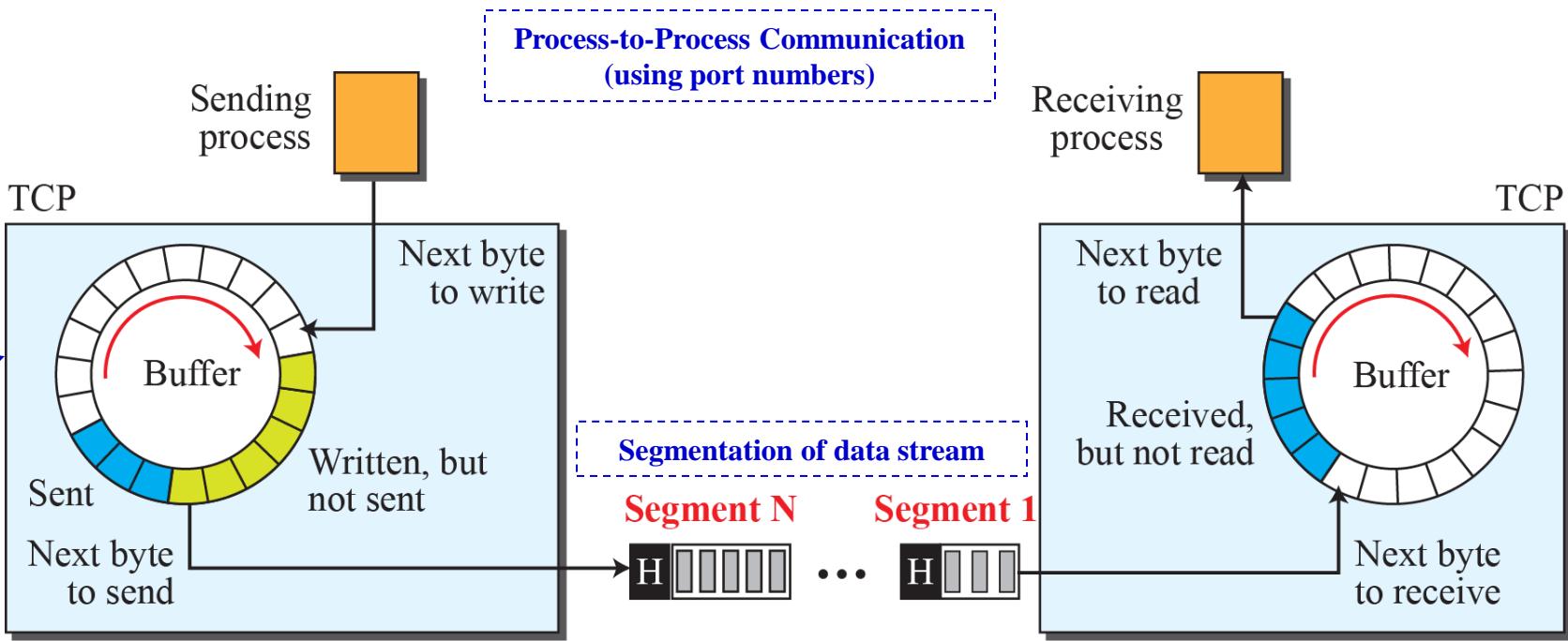
***Transmission Control Protocol (TCP) is a connection-oriented, reliable protocol.***

***TCP, unlike UDP, is a stream-oriented protocol. It allows the sending process to deliver data as a stream of bytes and allows the receiving process to obtain data as a stream of bytes. TCP creates an environment in which the two processes seem to be connected by an imaginary tube that carries their bytes across the Internet.***



## 24.3.1 TCP Services

Let's describe the services offered by TCP protocol (connection-oriented, reliable, full-duplex) to the processes at the application layer.



## 24.3.2 TCP Features

To provide the services, TCP keeps track of the segments being transmitted or received (for flow and error control) using two fields: sequence number and the acknowledgement number. The bytes of data being transmitted in each connection are numbered by TCP. Numbering is independent in each direction (for full-duplex communication).

**Sequence number:** TCP assigns a sequence number to each segment that is being sent. The sequence number of the first segment is the initial sequence number (ISN) and is an arbitrarily generated random number [0, 2<sup>32</sup>-1]. The sequence of any other segment is the sequence number of the previous segment plus the number of bytes carried by the previous segment, i.e., first byte of data in the segment.

**Acknowledgement number:** the value of the acknowledgement number in a segment defines the number of the next byte a party expects to receive. The acknowledgement number is cumulative.

## Example

Suppose a TCP connection is transferring a file of 5,000 bytes. The first byte is numbered 10,001. What are the sequence numbers for each segment if data are sent in five segments, each carrying 1,000 bytes?

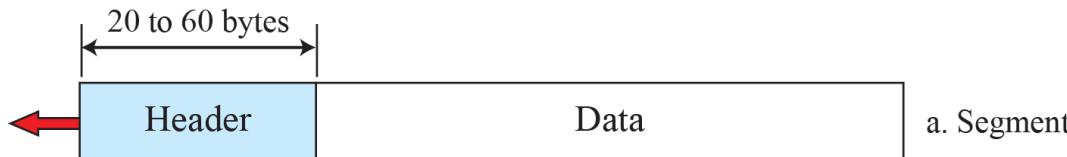
### Solution

The following shows the sequence number for each segment. The value in the sequence number field of a segment defines the number assigned to the first data byte contained in that segment.

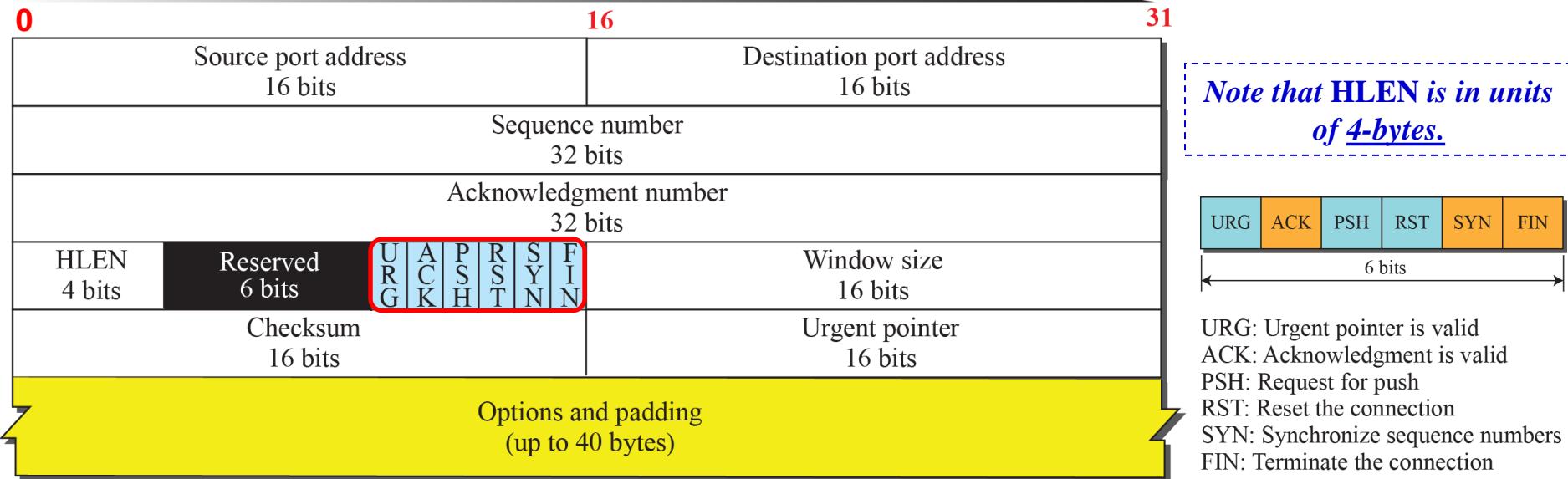
Segment 1	→	Sequence Number:	10,001	<b>Range:</b>	10,001	to	11,000
Segment 2	→	Sequence Number:	11,001	<b>Range:</b>	11,001	to	12,000
Segment 3	→	Sequence Number:	12,001	<b>Range:</b>	12,001	to	13,000
Segment 4	→	Sequence Number:	13,001	<b>Range:</b>	13,001	to	14,000
Segment 5	→	Sequence Number:	14,001	<b>Range:</b>	14,001	to	15,000

## 24.3.3 Segment

A packet in TCP is called a segment.



a. Segment



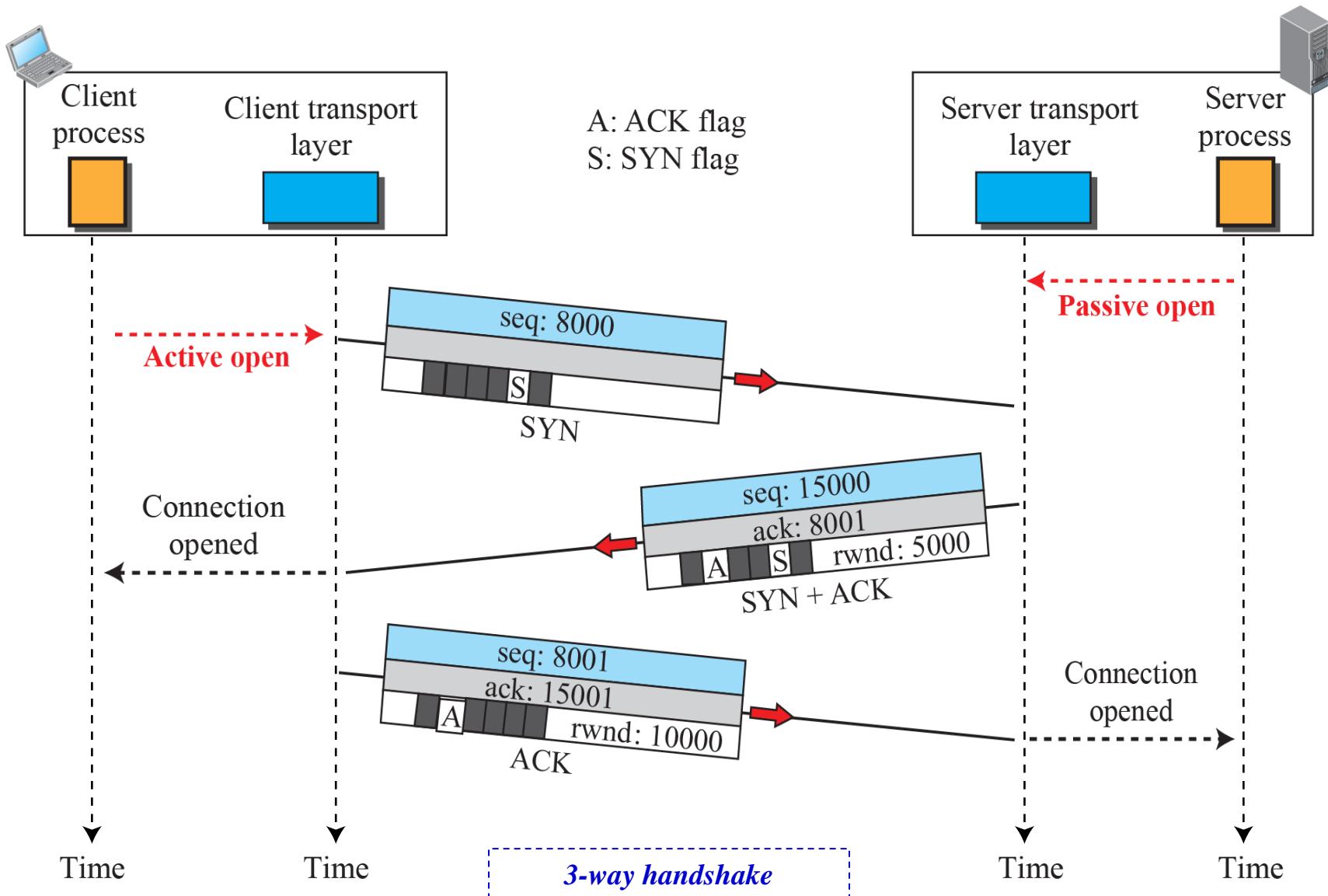
b. Header

## 24.3.4 A TCP Connection

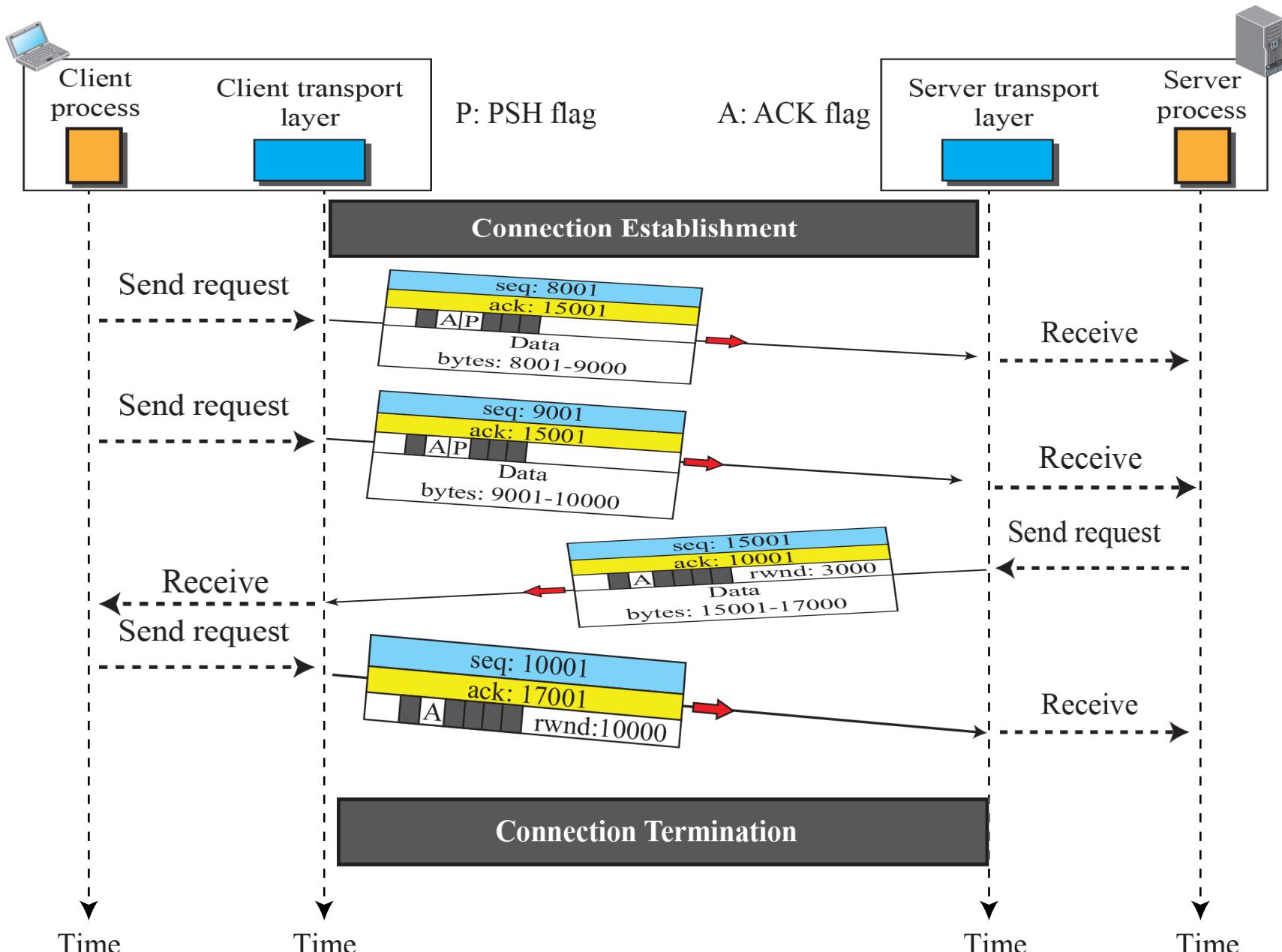
*TCP is connection-oriented. All of the segments belonging to a message are then sent over this logical path. Using a single logical pathway for the entire message facilitates the acknowledgment process as well as retransmission of damaged or lost frames.*

*In TCP, connection-oriented transmission requires three phases: connection establishment, data transfer and connection termination.*

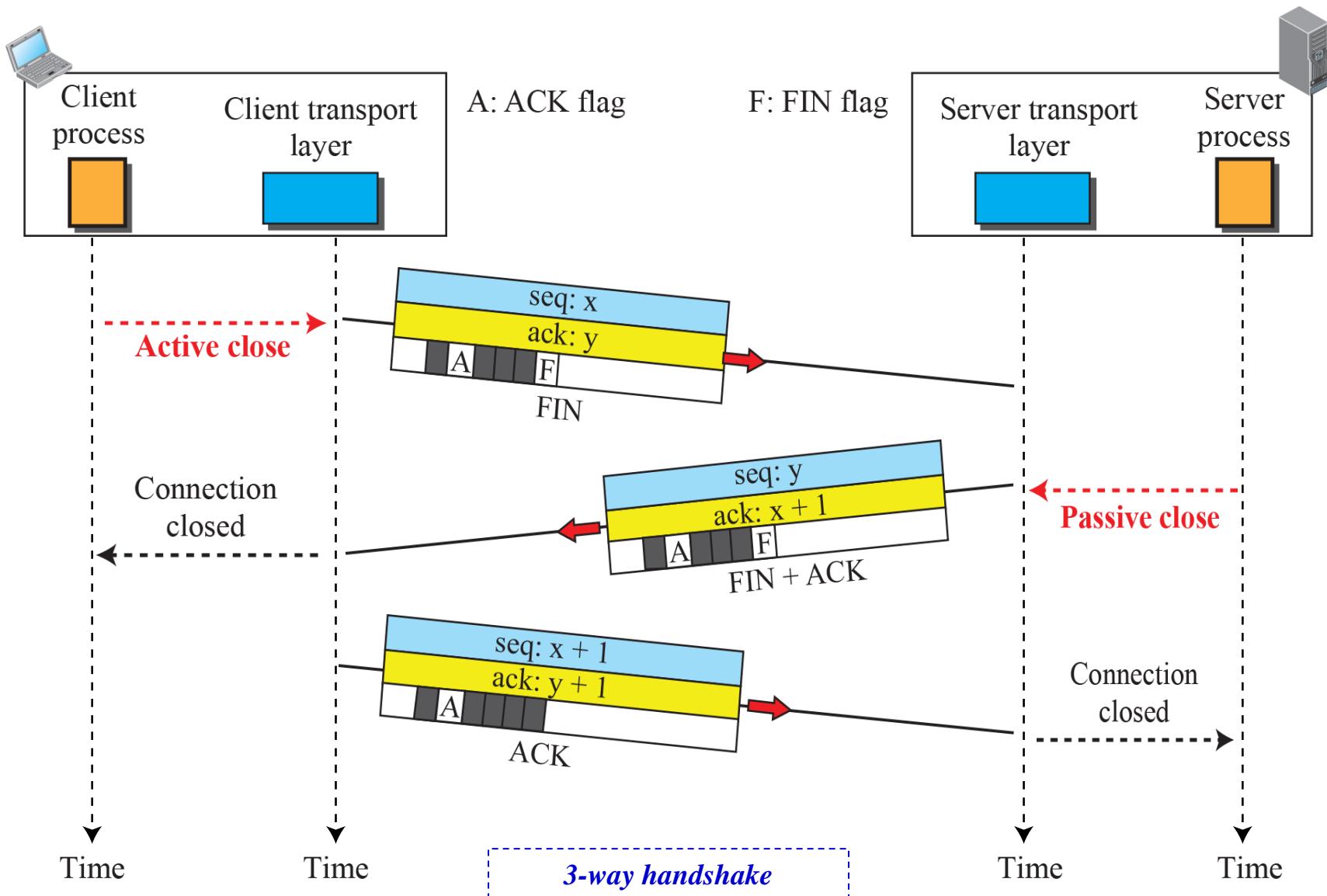
**Figure 24.10: Connection establishment**



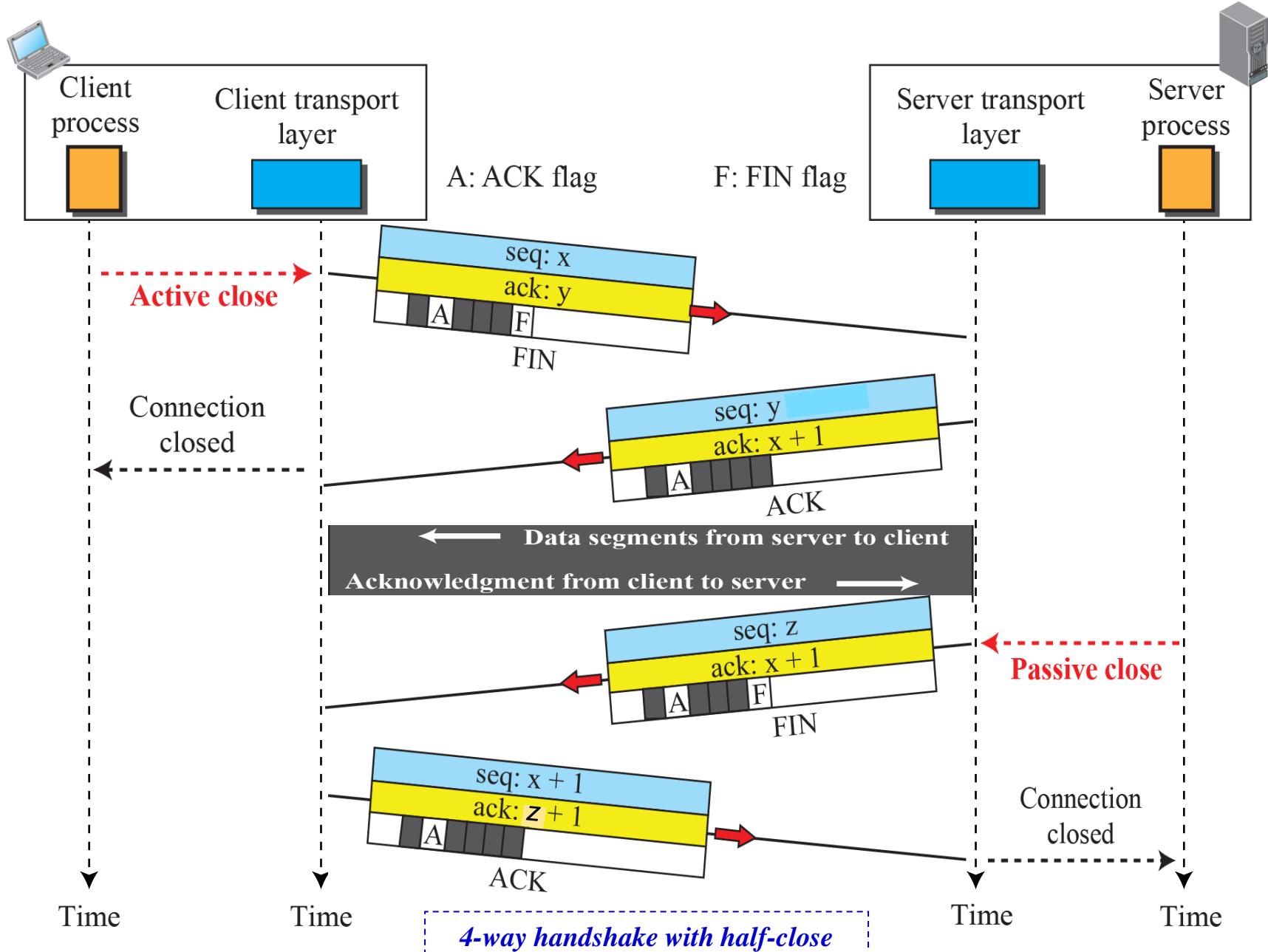
**Figure 24.11: Data transfer**



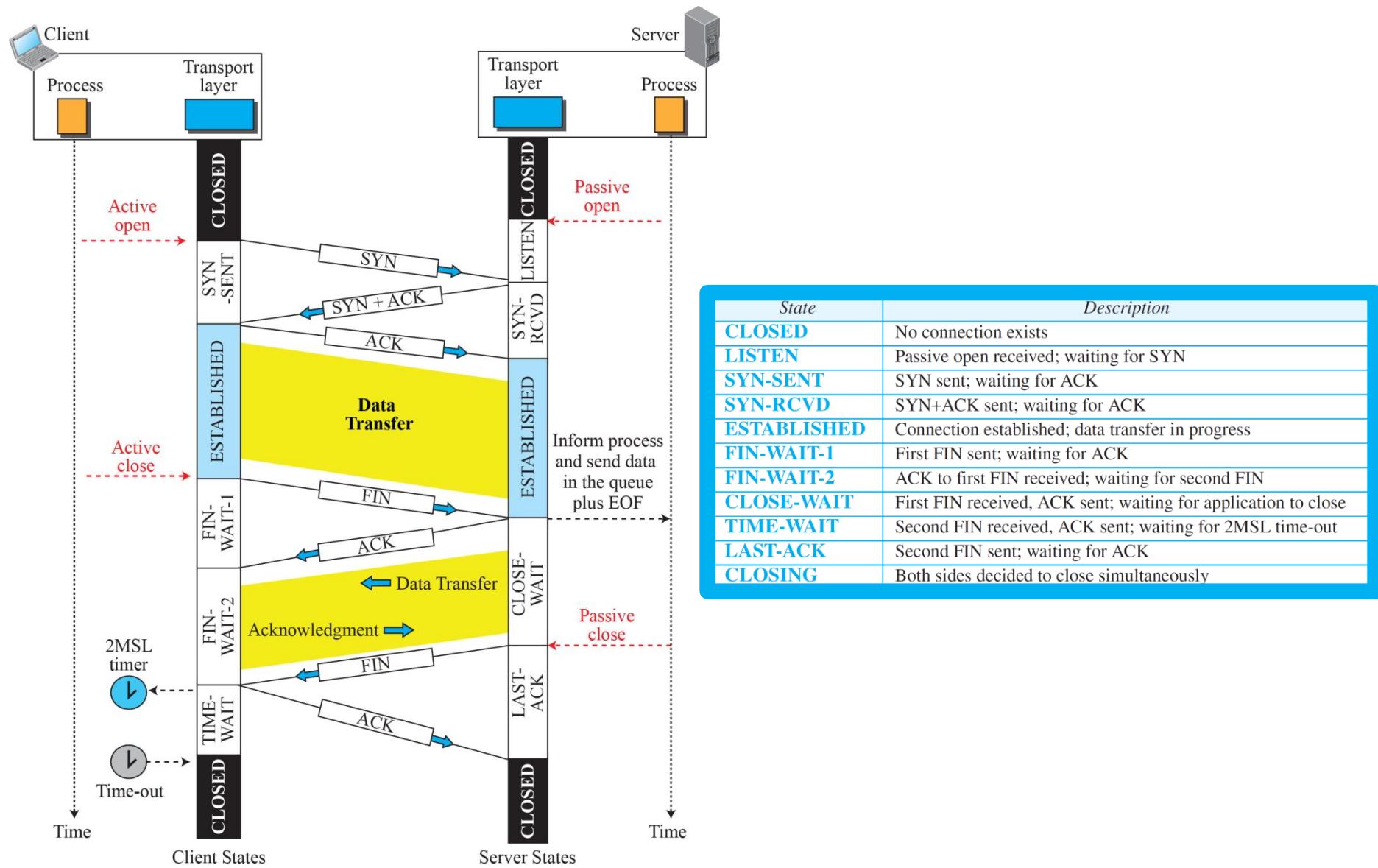
**Figure 24.12: Connection termination**



**Figure 24.13: Half-close**



**Figure 24.16: Time-line diagram for a common scenario**

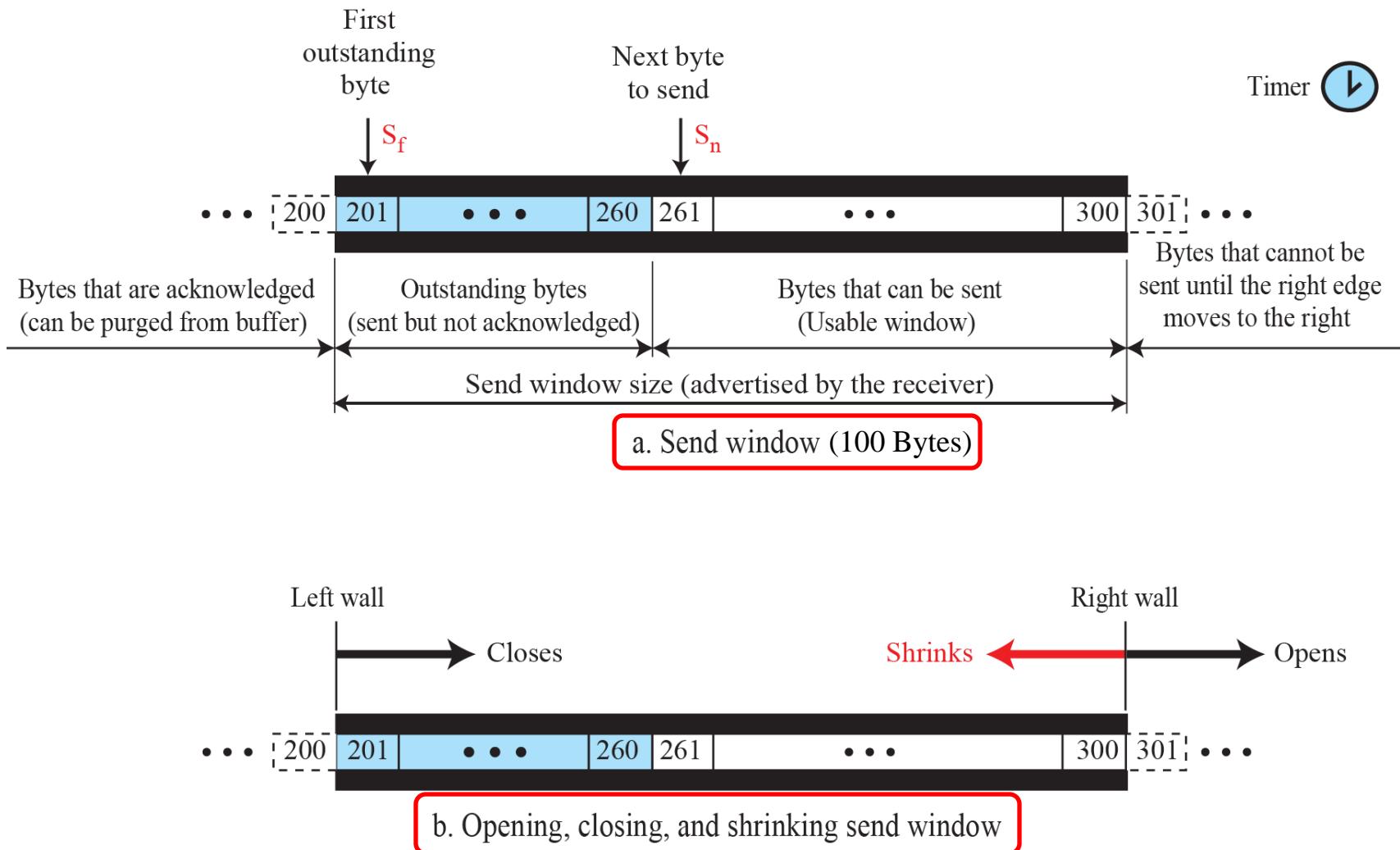


## 24.3.6 Windows in TCP

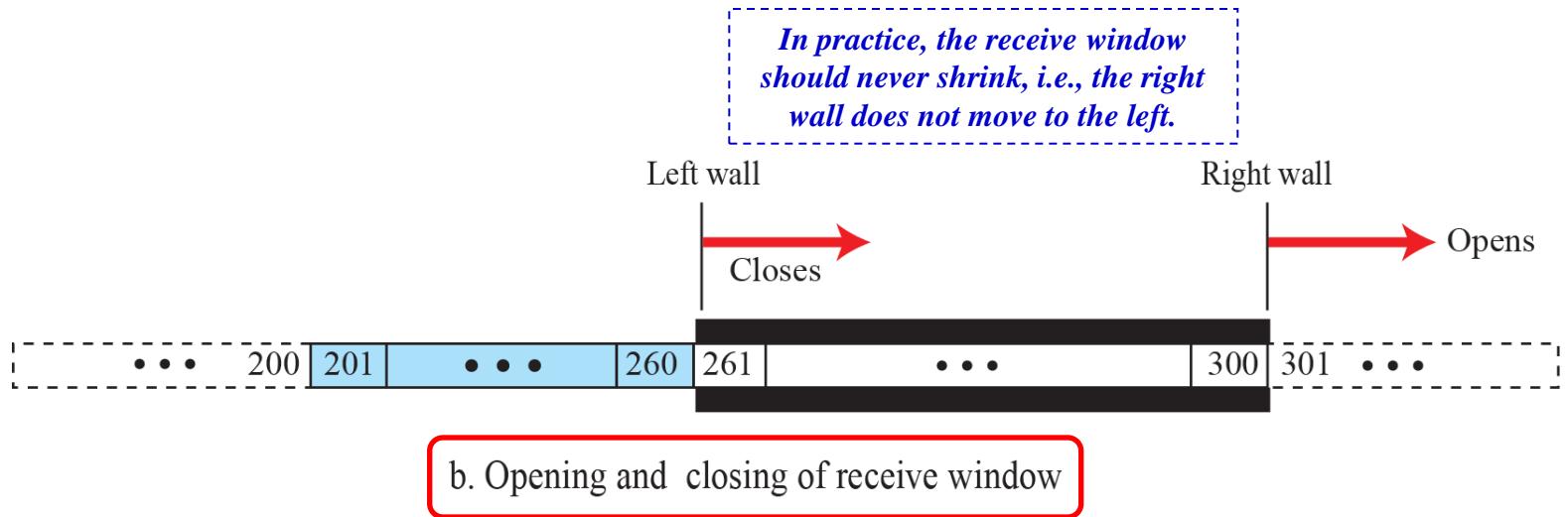
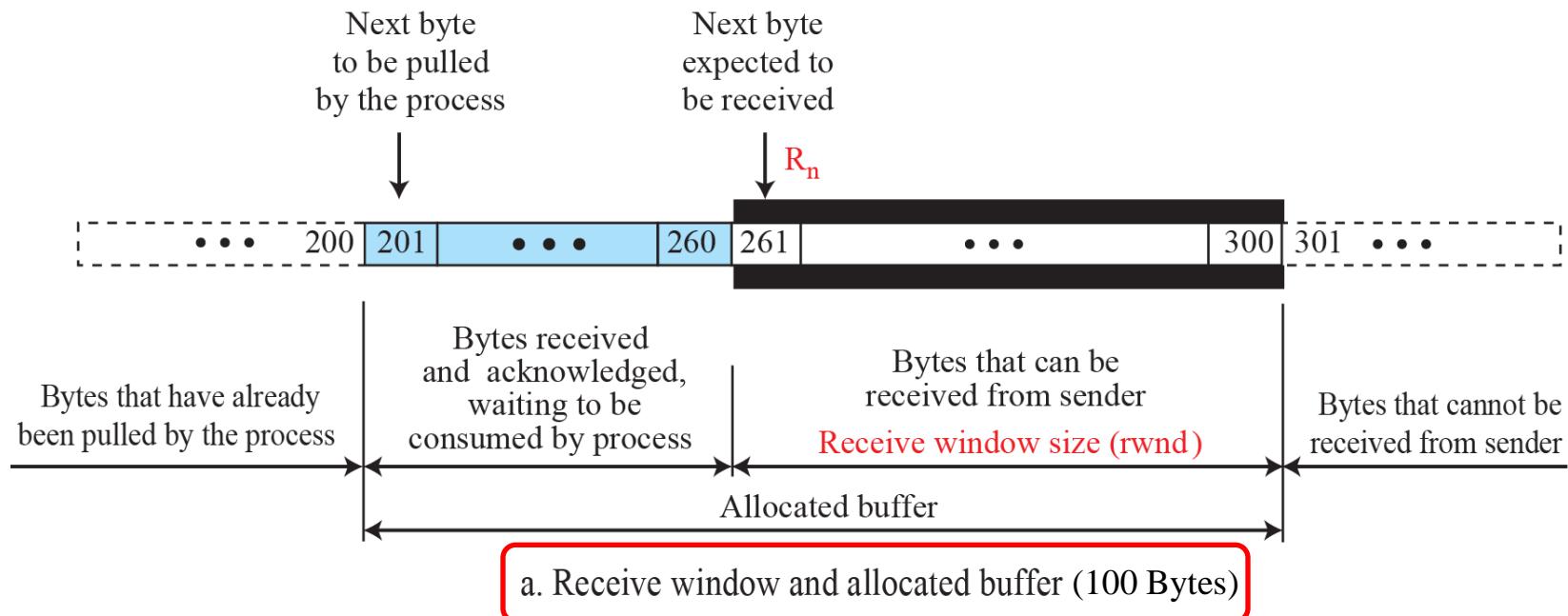
*Before discussing data transfer in TCP and the issues such as flow, error and congestion control, we describe the windows used in TCP. TCP uses two windows, send window and receive window, for each direction of data transfer, which means four windows for a bidirectional communication.*

*To make the discussion simple, we make an unrealistic assumption that communication is only unidirectional. The bidirectional communication can be inferred using two unidirectional communications with piggybacking.*

**Figure 24.17: Send window in TCP**

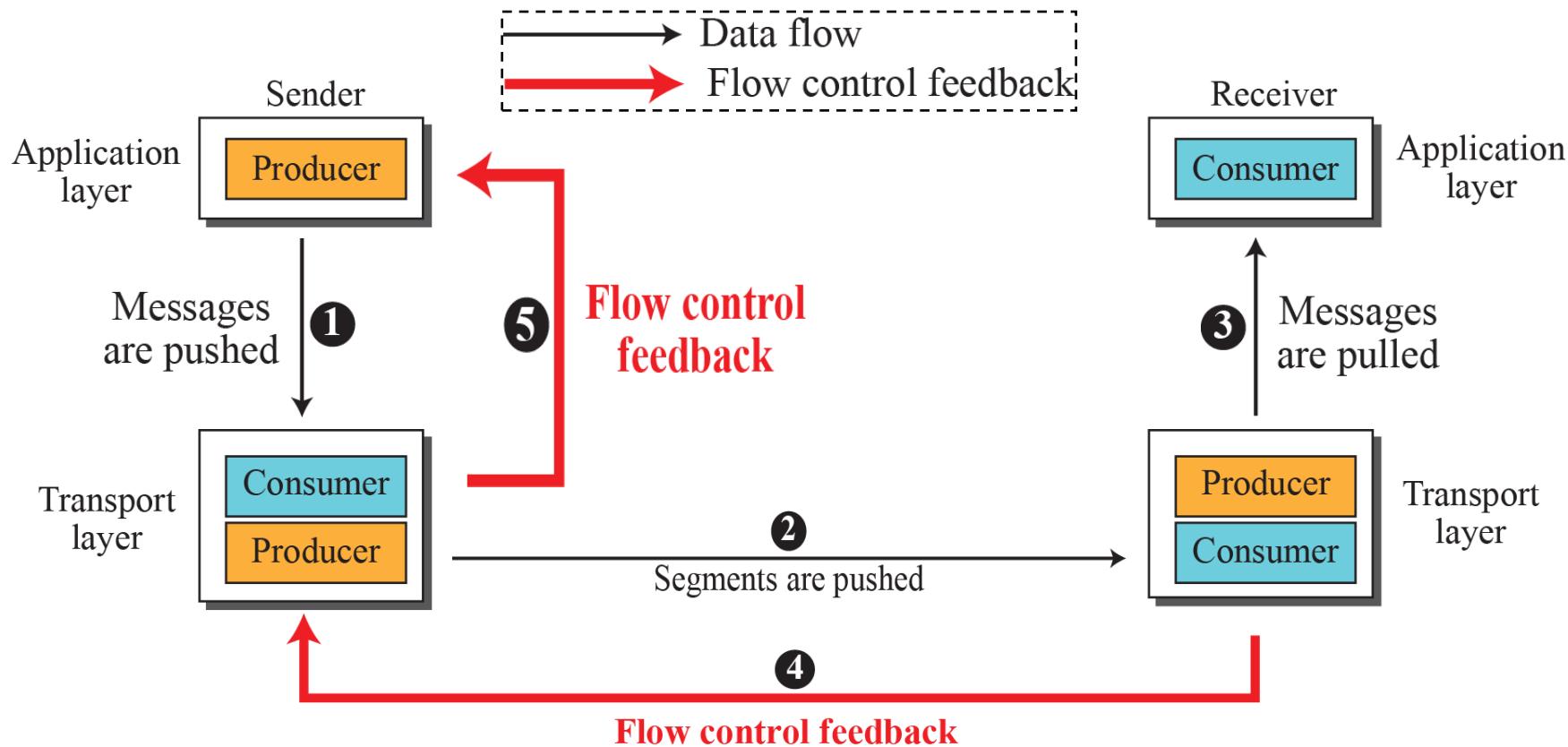


## Figure 24.18: Receive window in TCP



## 24.3.7 Flow Control

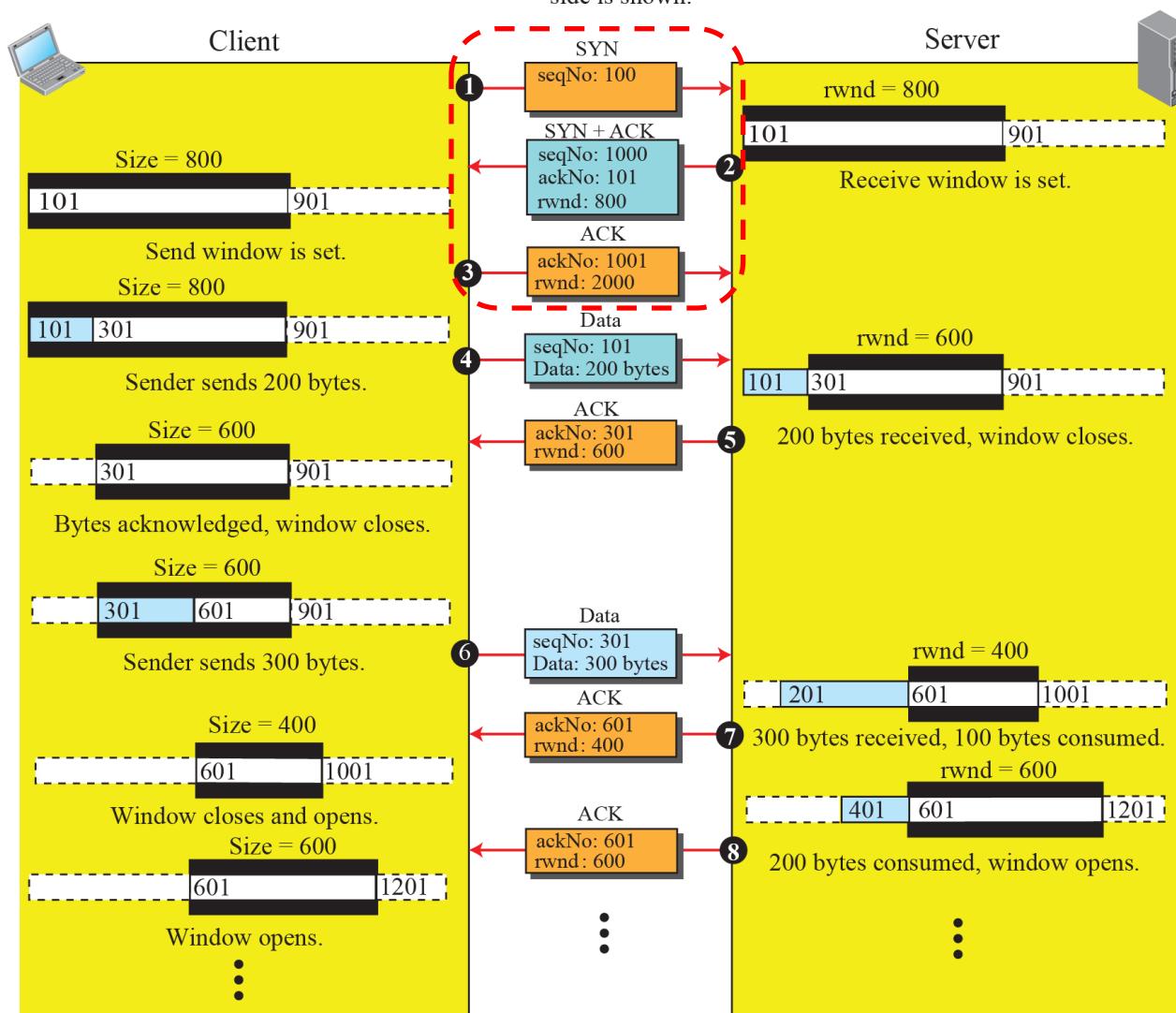
As discussed, flow control balances the rate a producer creates data with the rate a consumer can use the data. TCP separates flow control from error control. We assume that the logical channel between the sending and receiving TCP is error-free.



To achieve flow control, TCP forces the sender and receiver to adjust their window sizes, although the size of the buffer for both parties is fixed when the connection is established. The opening, closing and shrinking of the send window is controlled by the receiver.

## Figure 24.20: An example of flow control

**Note:** We assume only unidirectional communication from client to server. Therefore, only one window at each side is shown.



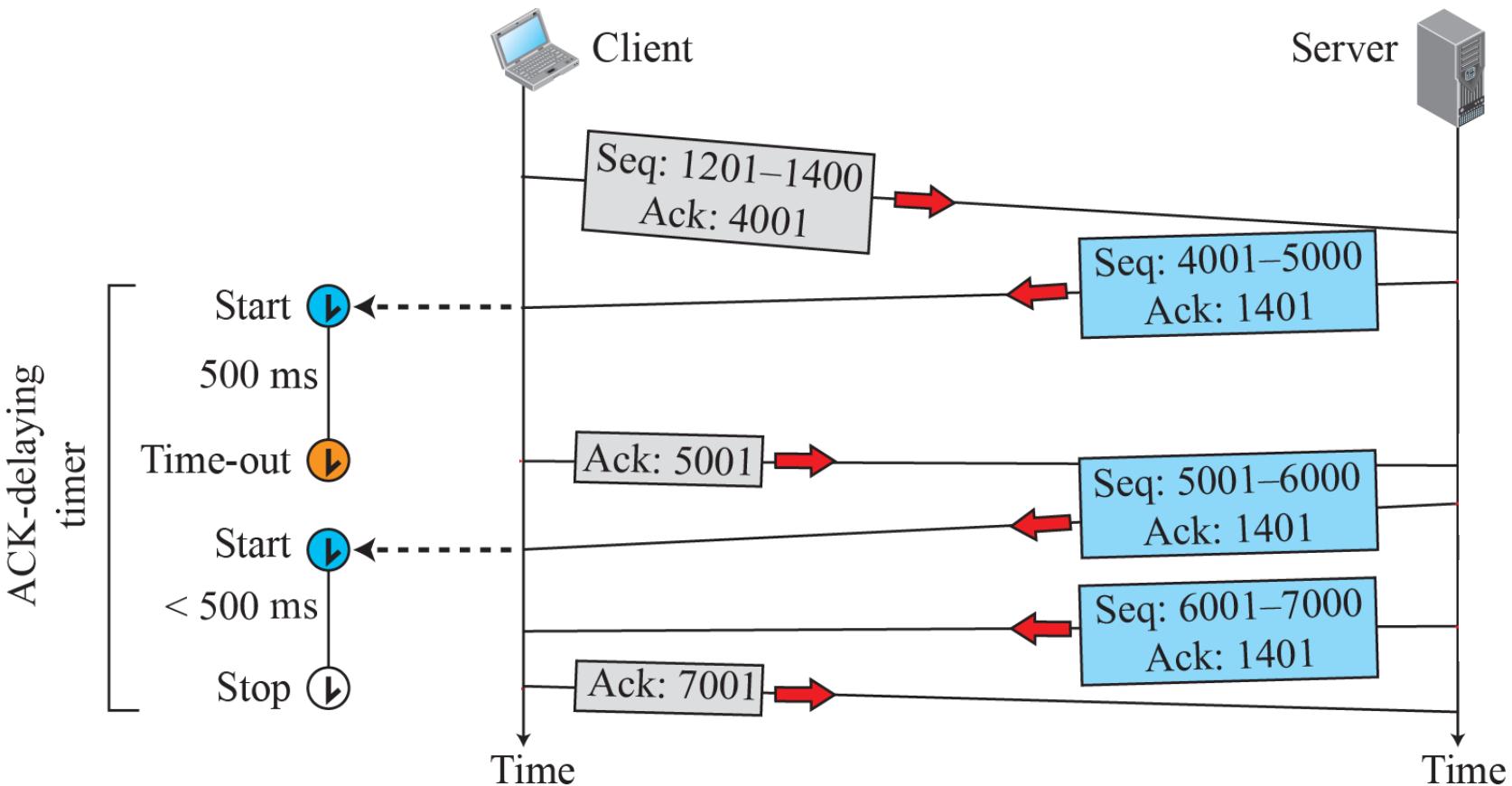
*The send window opens when the receive window size (rwnd) advertised by the receiver allows it to do so, i.e., (new ackNo + new rwnd > last ackNo + last rwnd), and shrinks in the event this situation does not occur.*

## 24.3.8 Error Control

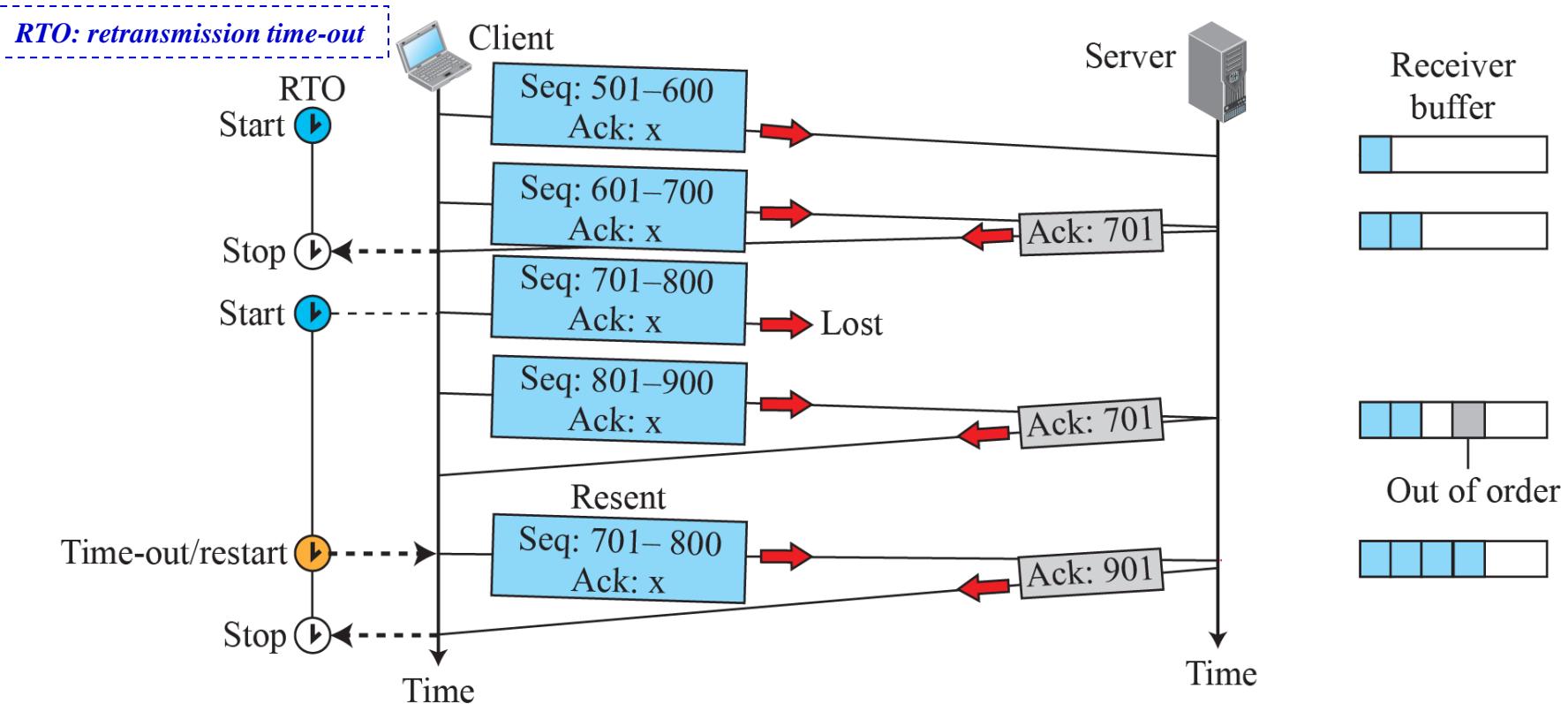
*TCP is a reliable transport-layer protocol. This means that an application program that delivers a stream of data to TCP relies on TCP to deliver the entire stream to the application program on the other end in order, without error and without any part lost or duplicated. Error control in TCP is achieved through the use of three simple tools: checksum, acknowledgement and time-out.*

*Let's consider some examples of error control scenarios that occur during the operation of TCP.*

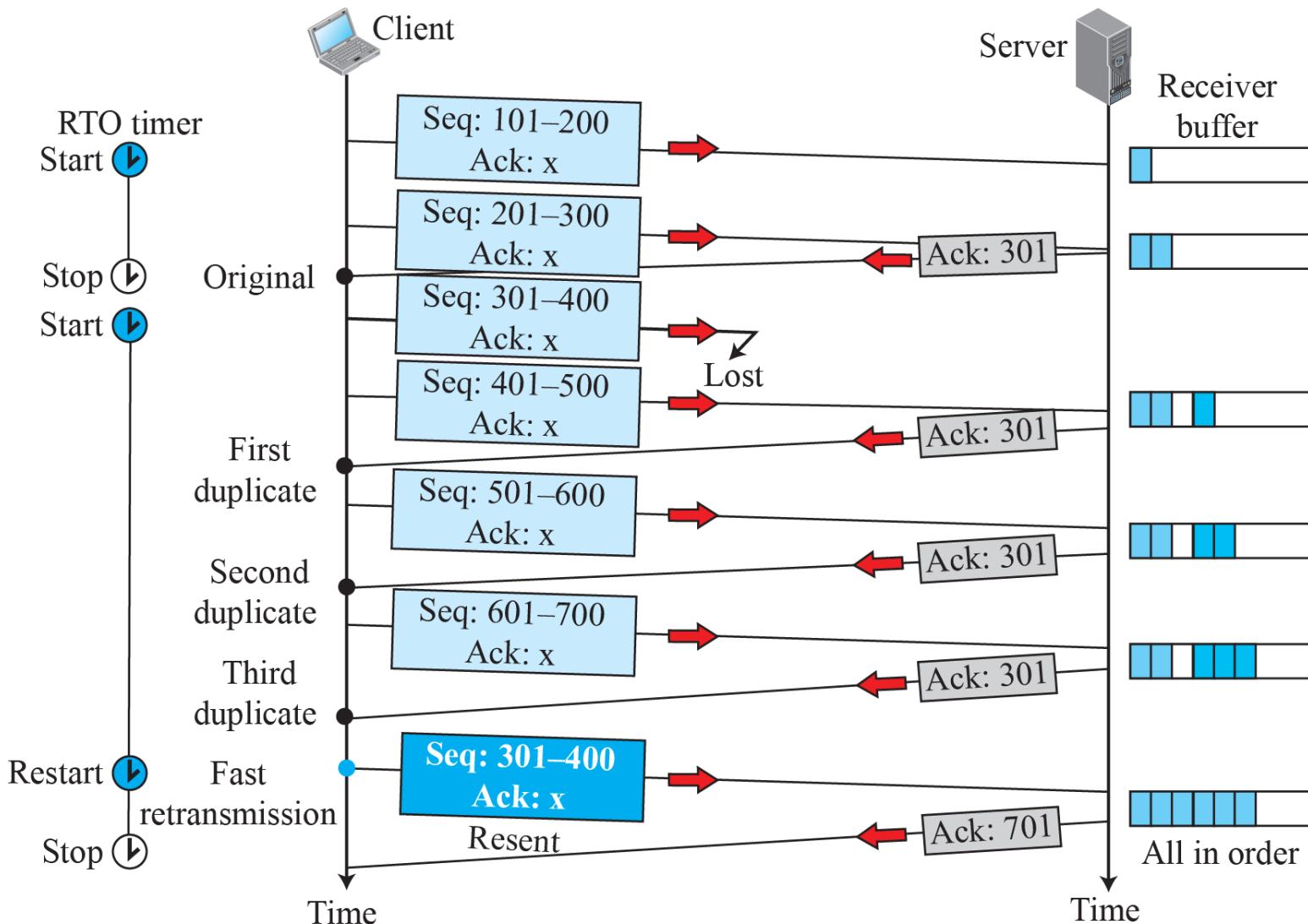
**Figure 24.24: Normal operation**



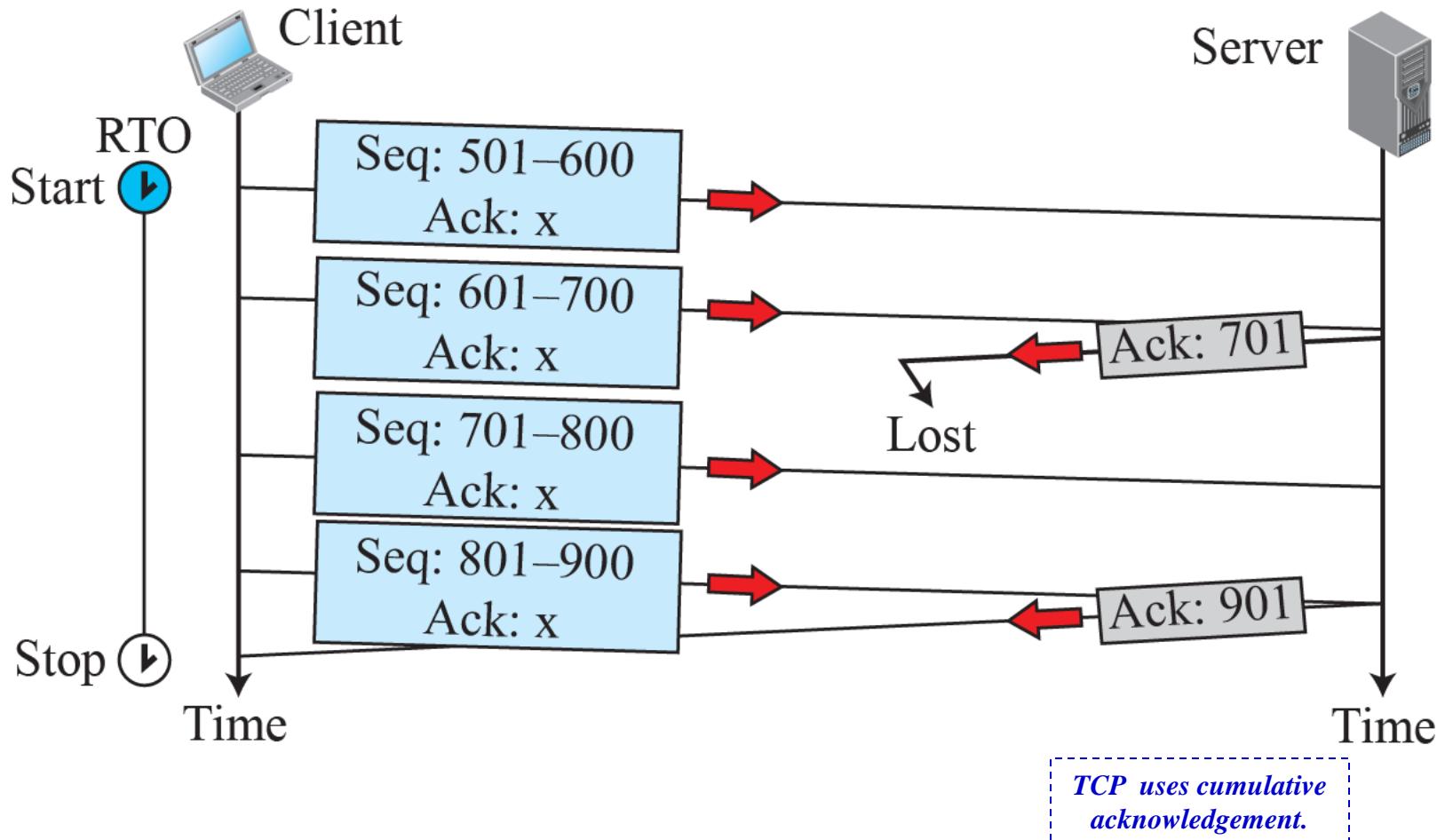
**Figure 24.25: Lost segment**



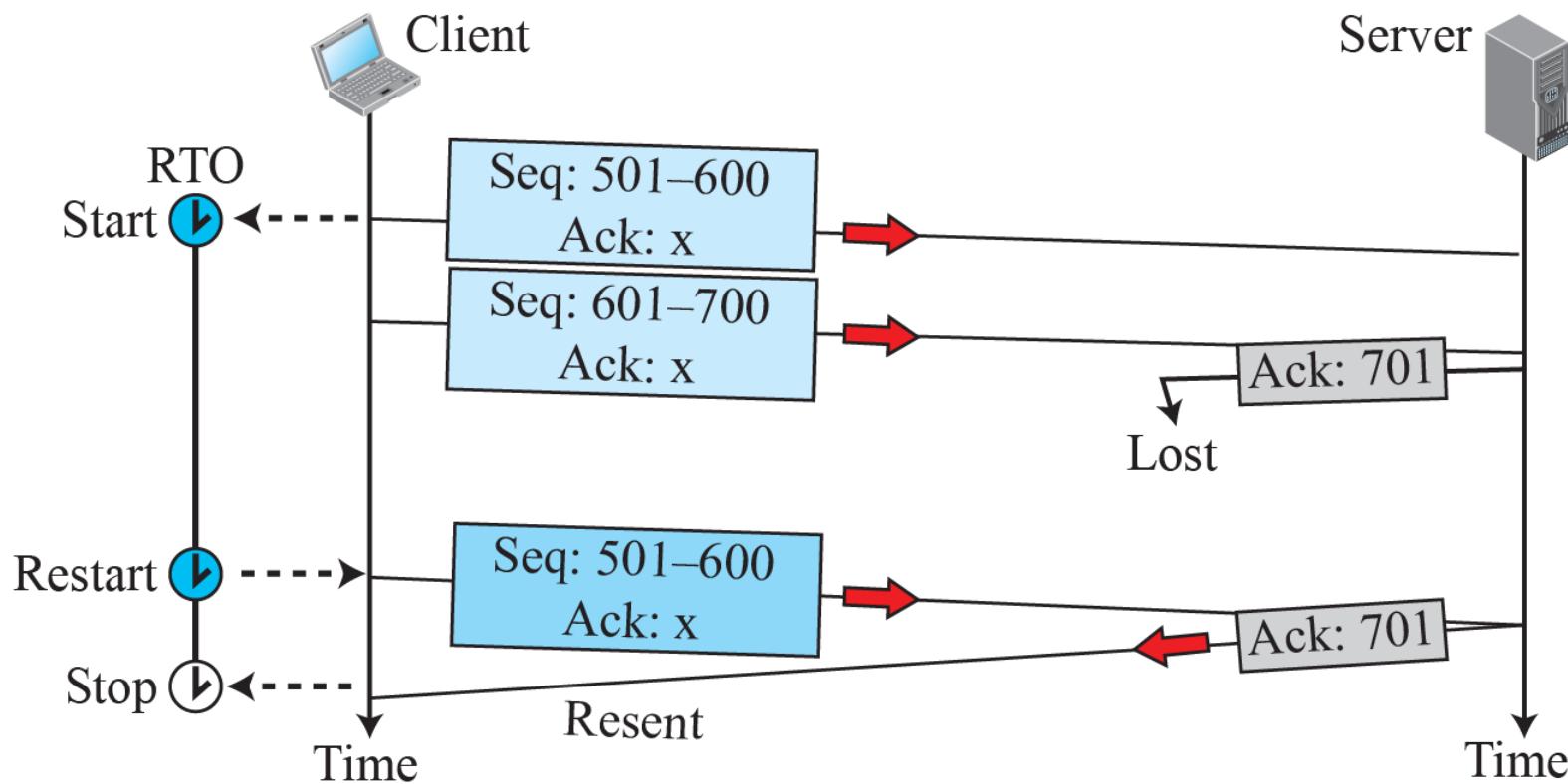
**Figure 24.26: Fast retransmission**



**Figure 24.27: Lost acknowledgment**

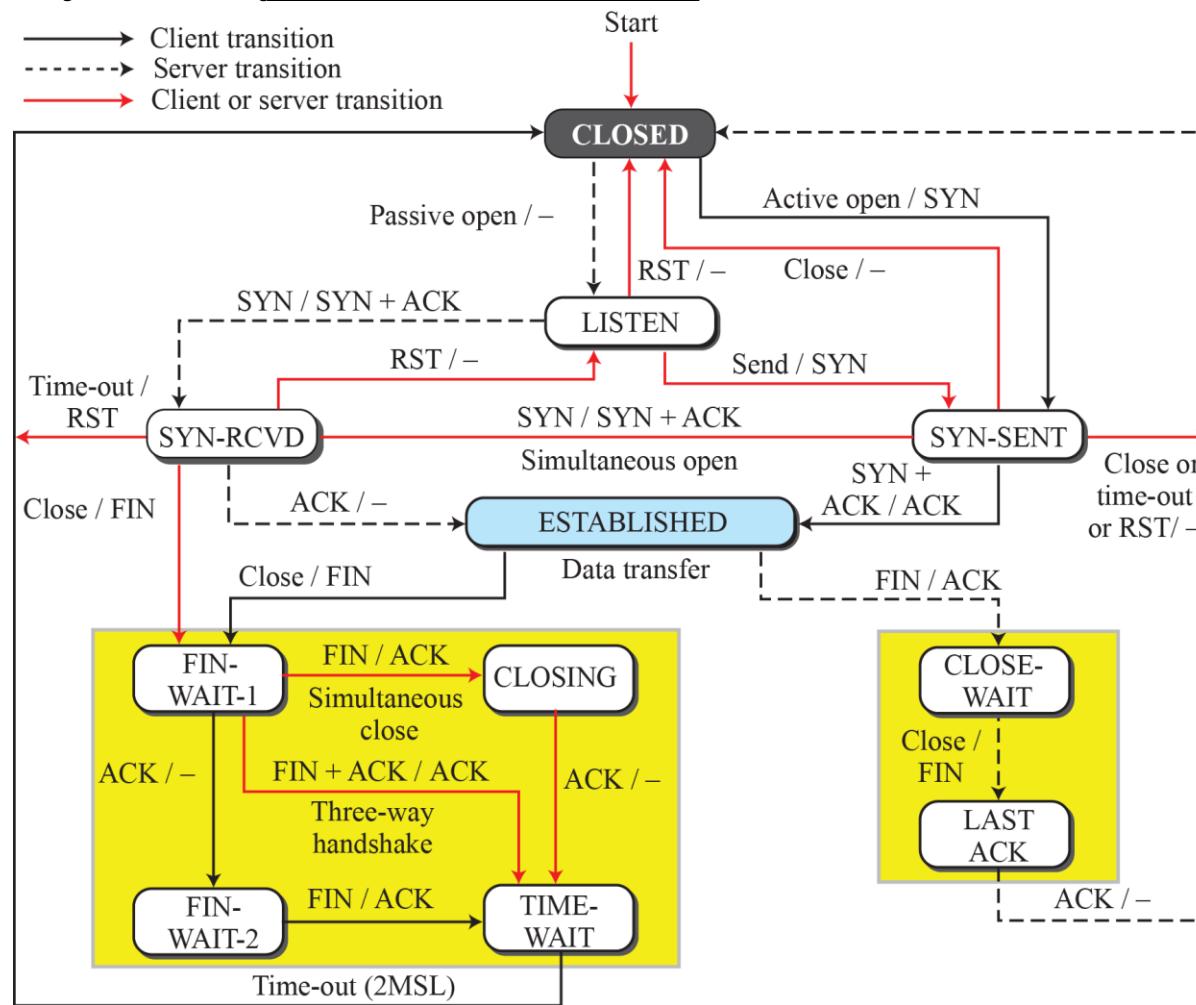


**Figure 24.28: Lost acknowledgment corrected by resending a segment**

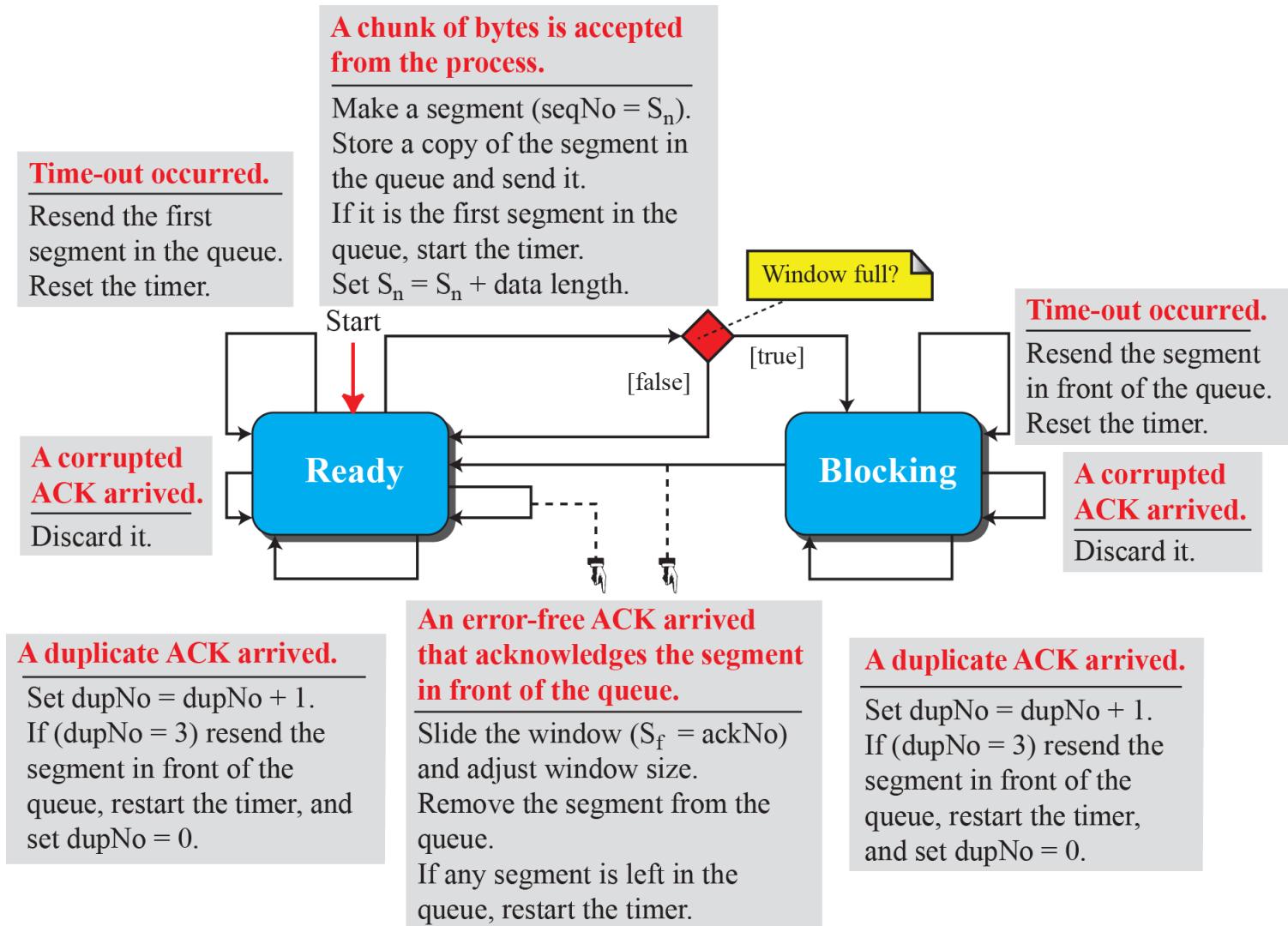


## 24.3.5 State Transition Diagram

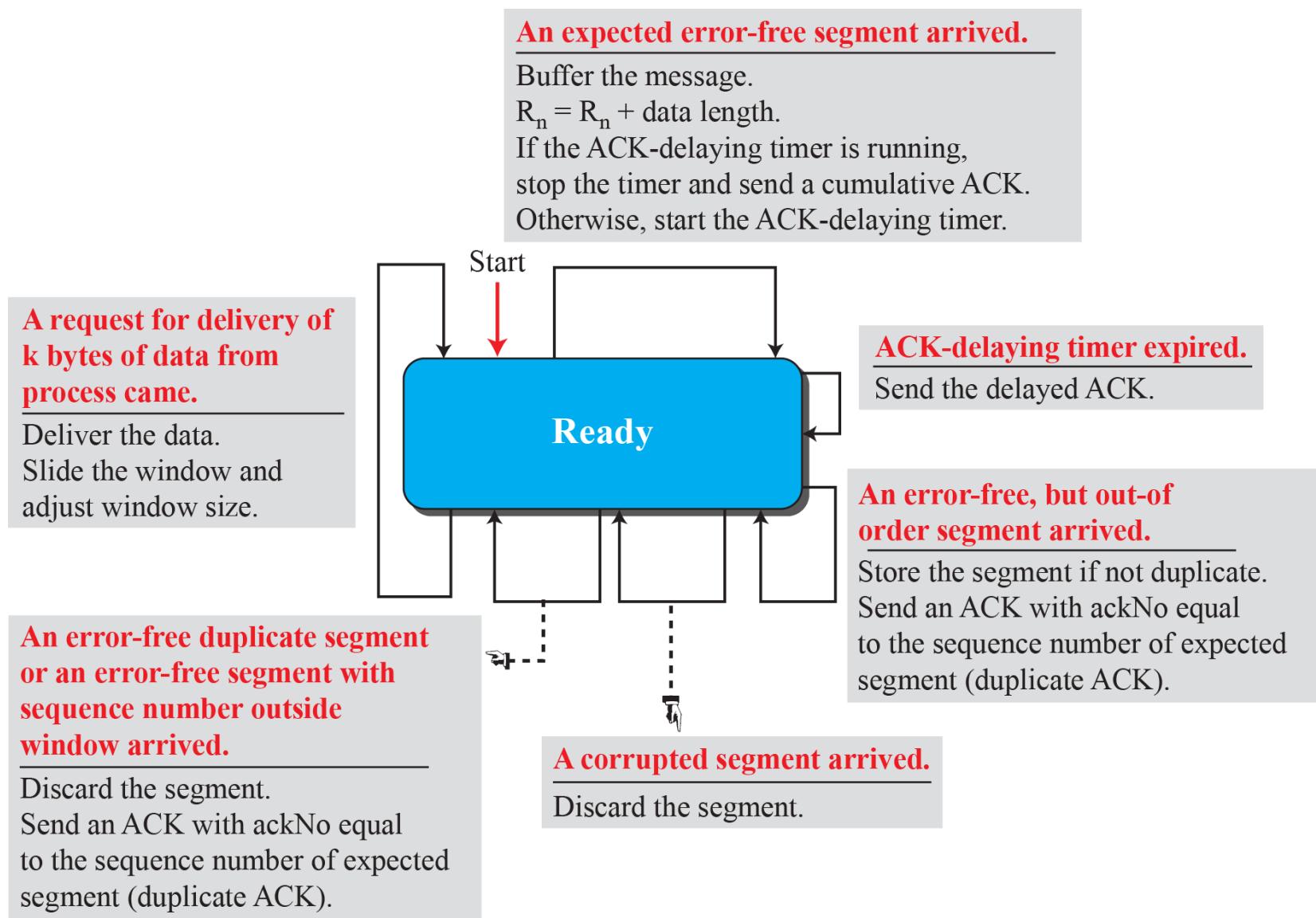
To keep track of all the different events happening during connection establishment, connection termination and data transfer, TCP is specified as a finite state machine (FSM).



**Figure 24.22: Simplified FSM for the TCP sender side**



**Figure 24.23: Simplified FSM for the TCP receiver side**



"That's all Folks!"