Data Communications and Networking Fourth Edition

Forouzan

Chapter 23

Process-to-Process Delivery: UDP, TCP, and SCTP

23-1 PROCESS-TO-PROCESS DELIVERY

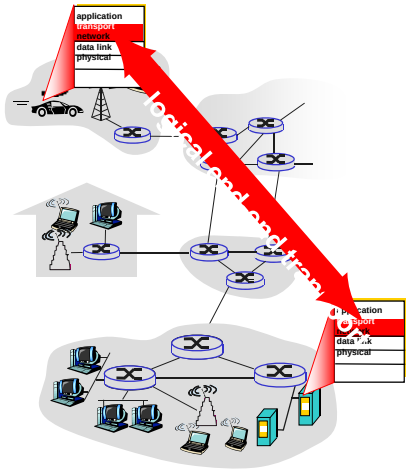
The transport layer is responsible for process-to-process delivery—the delivery of a packet, part of a message, from one process to another. Two processes communicate in a client/server relationship, as we will see later.

Topics discussed in this section:

Client/Server Paradigm
Multiplexing and Demultiplexing
Connectionless Versus Connection-Oriented Service
Reliable Versus Unreliable
Three Protocols

Transport services and protocols

- provide *logical communication* between app processes running on different hosts
- transport protocols run in end systems
  - send side: breaks app messages into segments, passes to network layer
  - rcv side: reassembles segments into messages, passes to app layer
- more than one transport protocol available to apps
  - Internet: TCP and UDP



Transport vs. network layer

- *network layer:* logical communication between hosts
- *transport layer:* logical communication between processes
  - relies on, enhances, network layer services

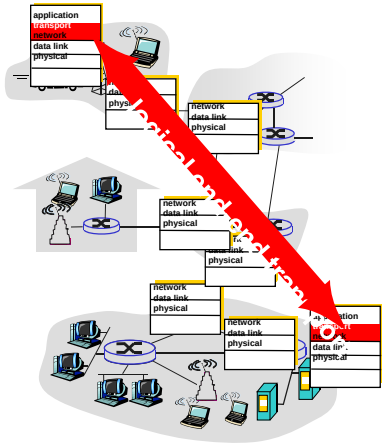
Household analogy:

12 kids sending letters to 12 kids

- processes = kids
- app messages = letters in envelopes
- hosts = houses
- transport protocol = Ann and Bill
- network-layer protocol = postal service

Internet transport-layer protocols

- reliable, in-order delivery (TCP)
  - congestion control
  - flow control
  - connection setup
- unreliable, unordered delivery: UDP
  - no-frills extension of "best-effort" IP
- services not available:
  - delay guarantees
  - bandwidth guarantees



Note

The transport layer is responsible for process-to-process delivery.

Figure 23.1 *Types of data deliveries*

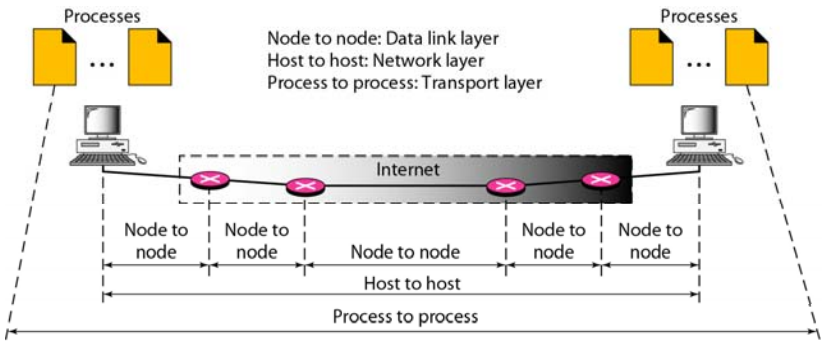


Figure 23.2 *Port numbers 16 bits*

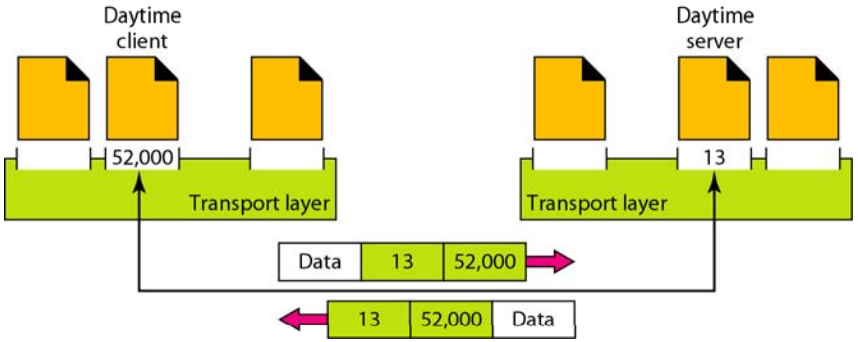


Figure 23.3 *IP addresses versus port numbers*

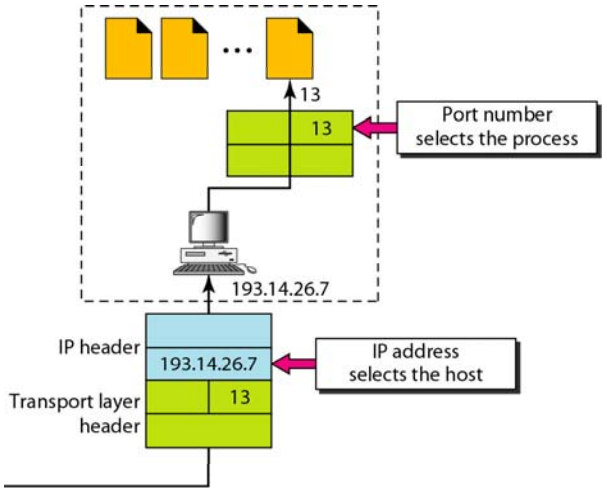


Figure 23.4 *IANA ranges*

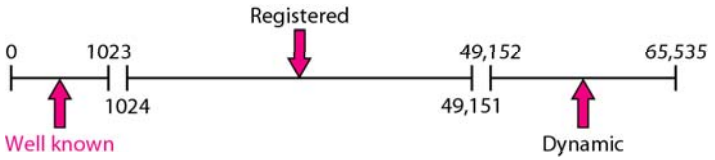
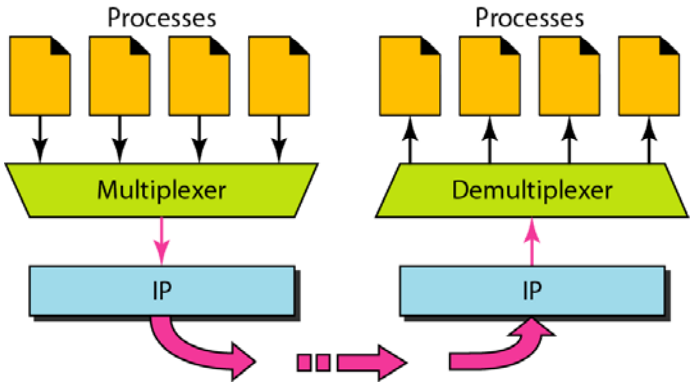


Figure 23.5 *Socket address*

Figure 23.6 *Multiplexing and demultiplexing*



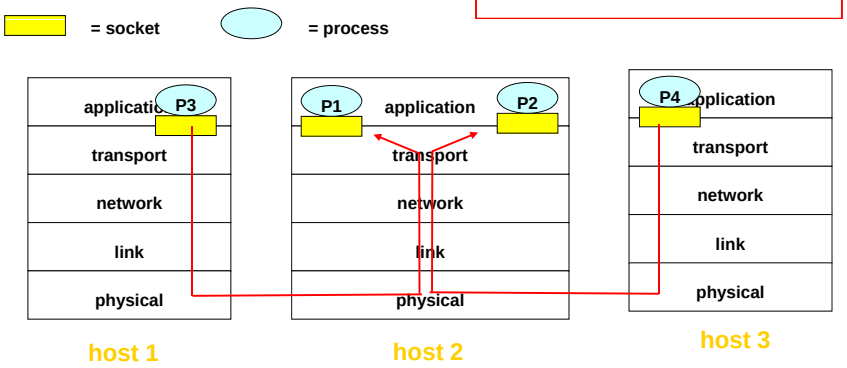
Multiplexing/demultiplexing

Demultiplexing at rcv host:

delivering received segments to correct socket

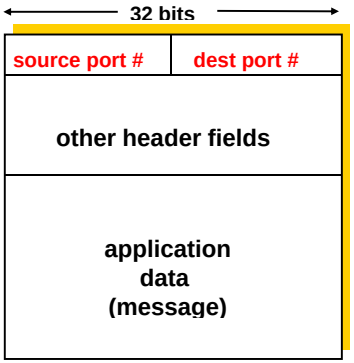
Multiplexing at send host:

gathering data from multiple sockets, enveloping data with header (later used for demultiplexing)



How demultiplexing works

- host receives IP datagrams
  - each datagram has source IP address, destination IP address
  - each datagram carries 1 transport-layer segment
  - each segment has source, destination port number
- host uses IP addresses & port numbers to direct segment to appropriate socket



TCP/UDP segment format

Figure 23.7 *Error control*

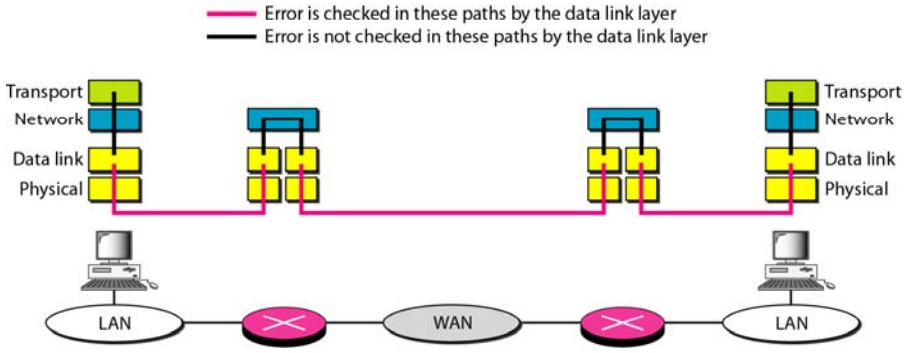
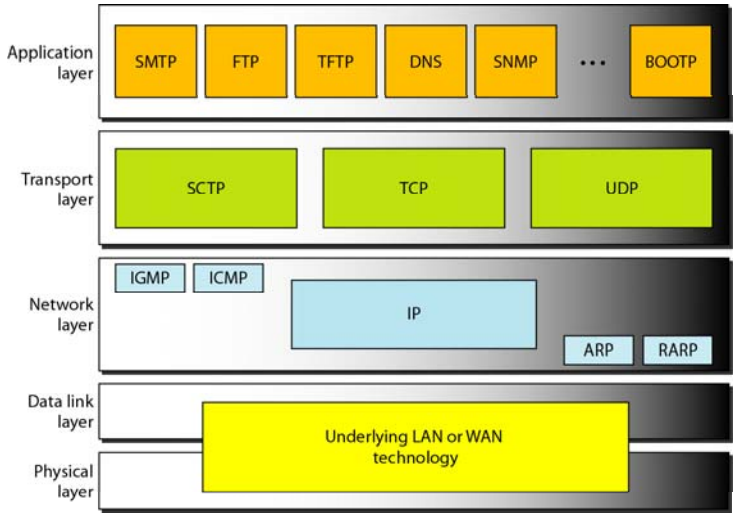


Figure 23.8 *Position of UDP, TCP, and SCTP in TCP/IP suite*



23-2 USER DATAGRAM PROTOCOL (UDP)

The User Datagram Protocol (UDP) is called a connectionless, unreliable transport protocol. It does not add anything to the services of IP except to provide process-to-process communication instead of host-to-host communication.

Topics discussed in this section:

Well-Known Ports for UDP
User Datagram
Checksum
UDP Operation
Use of UDP

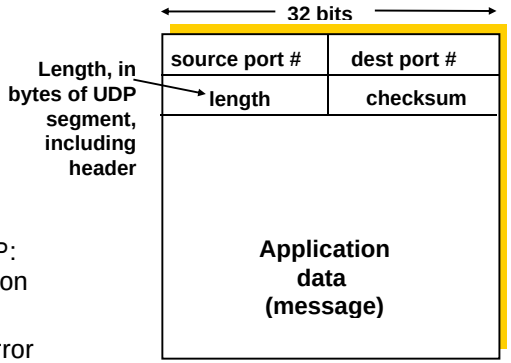
UDP: User Datagram Protocol [RFC 768]

- "no frills," "bare bones" Internet transport protocol
- "best effort" service, UDP segments may be:
  - lost
  - delivered out of order to app
- *connectionless:*
  - no handshaking between UDP sender, receiver
  - each UDP segment handled independently of others

Why is there a UDP?
- no connection establishment (which can add delay)
- simple: no connection state at sender, receiver
- small segment header
- no congestion control: UDP can blast away as fast as desired

UDP: more

- often used for streaming multimedia apps
  - loss tolerant
  - rate sensitive
- other UDP uses
  - DNS
  - SNMP
- reliable transfer over UDP: add reliability at application layer
  - application-specific error recovery!



UDP segment format

UDP checksum

Goal: detect "errors" (e.g., flipped bits) in transmitted segment

Sender:
- treat segment contents as sequence of 16-bit integers
- checksum: addition (1's complement sum) of segment contents
- sender puts checksum value into UDP checksum field

Receiver:
- compute checksum of received segment
- check if computed checksum equals checksum field value:
  - NO - error detected
  - YES - no error detected. *But maybe errors nonetheless?* More later ....

Internet Checksum Example

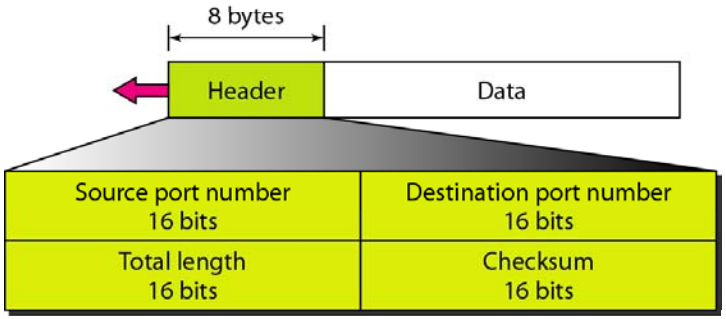
- Note
  - When adding numbers, a carryout from the most significant bit needs to be added to the result
- Example: add two 16-bit integers

```
                1 1 1 0 0 1 1 0 0 1 1 0 0 1 1 0
                1 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1
              ---------------------------------
wraparound    1 1 0 1 1 1 0 1 1 1 0 1 1 1 0 1 1
              ---------------------------------
sum             1 0 1 1 1 0 1 1 1 0 1 1 1 1 0 0
checksum        0 1 0 0 0 1 0 0 0 1 0 0 0 0 1 1
```

Table 23.1 *Well-known ports used with UDP*

Port	*Protocol*	*Description*
7	Echo	Echoes a received datagram back to the sender
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
53	Nameserver	Domain Name Service
67	BOOTPs	Server port to download bootstrap information
68	BOOTPc	Client port to download bootstrap information
69	TFTP	Trivial File Transfer Protocol
111	RPC	Remote Procedure Call
123	NTP	Network Time Protocol
161	SNMP	Simple Network Management Protocol
162	SNMP	Simple Network Management Protocol (trap)

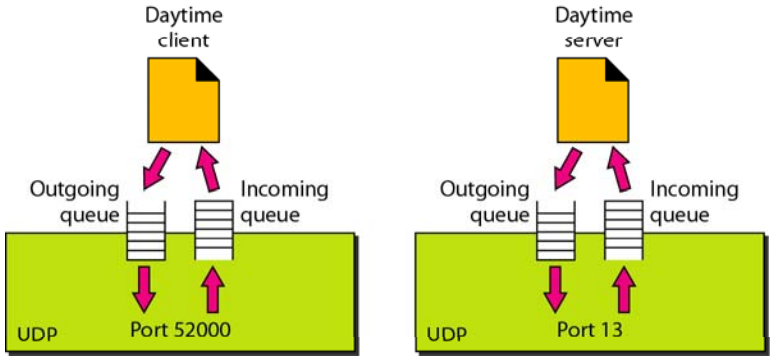
Figure 23.9 *User datagram format*



Note

UDP length
= IP length – IP header's length

Figure 23.12 *Queues in UDP*



UDP is suitable for:

- A process that requires simple request-response communication with little concern for flow and error control – FTP
- A process with internal flow and error control mechanisms – TFTF
- Multicasting
- Management processes – SNMP
- Some rout updating protocols - RIP

23-3 TCP

TCP is a connection-oriented protocol; it creates a virtual connection between two TCPs to send data. In addition, TCP uses flow and error control mechanisms at the transport level.

Topics discussed in this section:

TCP Services
TCP Features
Segment
A TCP Connection
Flow Control
Error Control

TCP: Overview

RFCs: 793, 1122, 1323, 2018, 2581

- point-to-point:
  - one sender, one receiver
- reliable, in-order *byte steam:*
  - no “message boundaries”
- pipelined:
  - TCP congestion and flow control set window size
- *send & receive buffers*

- full duplex data:
  - bi-directional data flow in same connection
  - MSS: maximum segment size
- connection-oriented:
  - handshaking (exchange of control msgs) init’s sender, receiver state before data exchange
- flow controlled:
  - sender will not overwhelm receiver

TCP segment structure

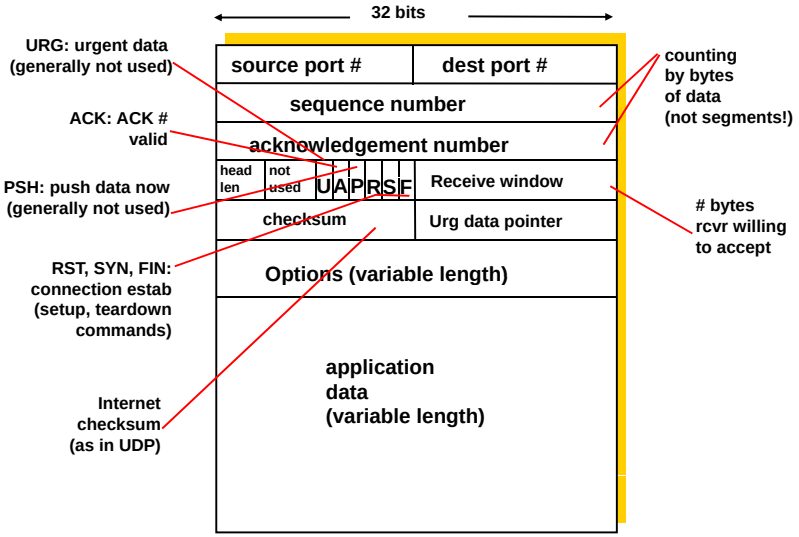


Table 23.2 *Well-known ports used by TCP*

Port	*Protocol*	*Description*
7	Echo	Echoes a received datagram back to the sender
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	FTP, Data	File Transfer Protocol (data connection)
21	FTP, Control	File Transfer Protocol (control connection)
23	TELNET	Terminal Network
25	SMTP	Simple Mail Transfer Protocol
53	DNS	Domain Name Server
67	BOOTP	Bootstrap Protocol
79	Finger	Finger
80	HTTP	Hypertext Transfer Protocol
111	RPC	Remote Procedure Call

Figure 23.13 *Stream delivery*

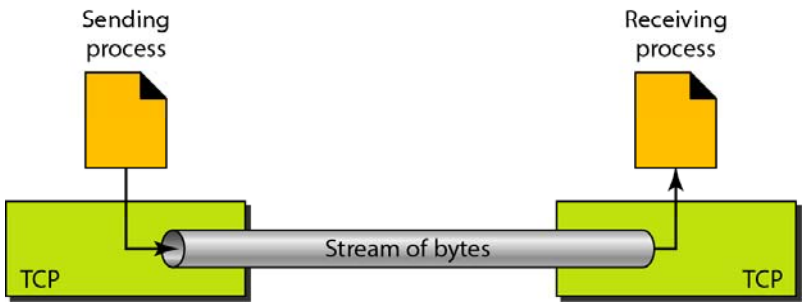


Figure 23.14 *Sending and receiving buffers*

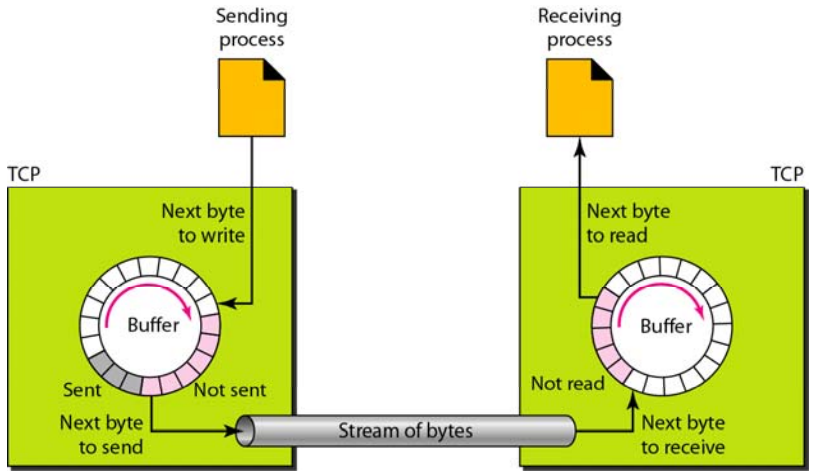
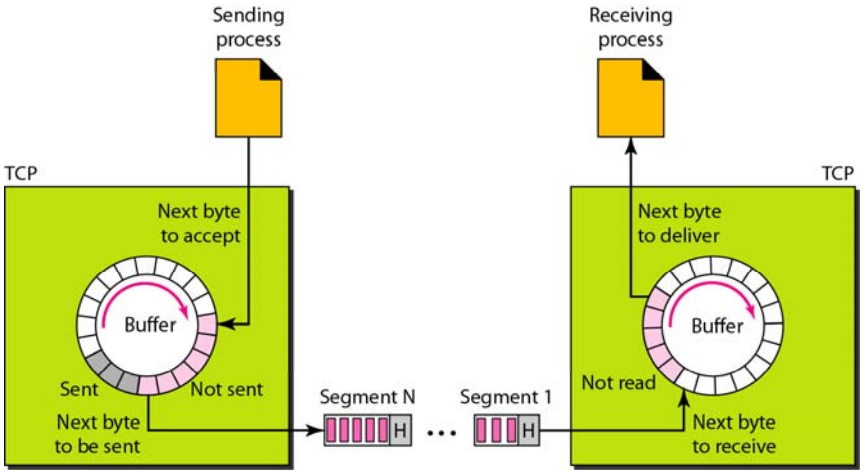


Figure 23.15 *TCP segments*



TCP provides:

- Full duplex communication
- Connection oriented service:
  - The two TCPs establish a connection between them
  - Data are exchanged in both directions
  - The connection is terminated
- Reliable service
- Flow control
- Error Control
- Congestion control

Note

The bytes of data being transferred in each connection are numbered by TCP.
• Sequence number
• Acknowledgment number
The numbering starts with a randomly generated number.

Example 23.3

The file is 5000 bytes, the first byte is numbered 10,001 each segment caries 1000 byte

The following shows the sequence number for each segment:

Segment 1	→	Sequence Number: 10,001 (range: 10,001 to 11,000)
Segment 2	→	Sequence Number: 11,001 (range: 11,001 to 12,000)
Segment 3	→	Sequence Number: 12,001 (range: 12,001 to 13,000)
Segment 4	→	Sequence Number: 13,001 (range: 13,001 to 14,000)
Segment 5	→	Sequence Number: 14,001 (range: 14,001 to 15,000)

TCP seq. #'s and ACKs

Seq. #'s:

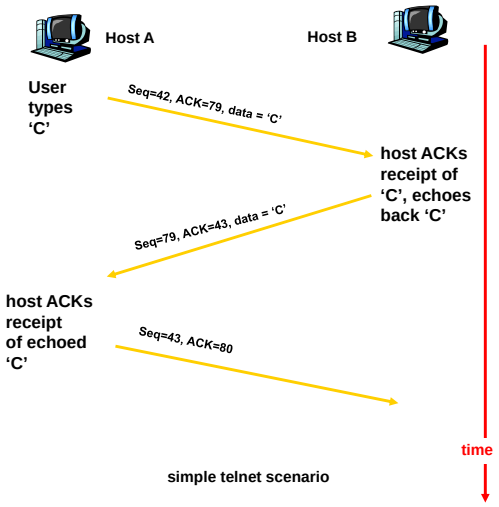
- byte stream "number" of first byte in segment's data

ACKs:

- seq # of next byte expected from other side
- cumulative ACK

Q: how receiver handles out-of-order segments

- A: TCP spec doesn't say, - up to implementer



simple telnet scenario

TCP Round Trip Time and Timeout

Q: how to set TCP timeout value?

- longer than RTT
  - but RTT varies
- too short: premature timeout
  - unnecessary retransmissions
- too long: slow reaction to segment loss

Q: how to estimate RTT?

- `SampleRTT`: measured time from segment transmission until ACK receipt
  - ignore retransmissions
- **`SampleRTT`** will vary, want estimated RTT "smoother"
  - average several recent measurements, not just current **`SampleRTT`**

Note

The value in the sequence number field of a segment defines the number of the first data byte contained in that segment.

Note

The value of the acknowledgment field in a segment defines the number of the next byte a party expects to receive. The acknowledgment number is cumulative.

Figure 23.16 *TCP segment format*

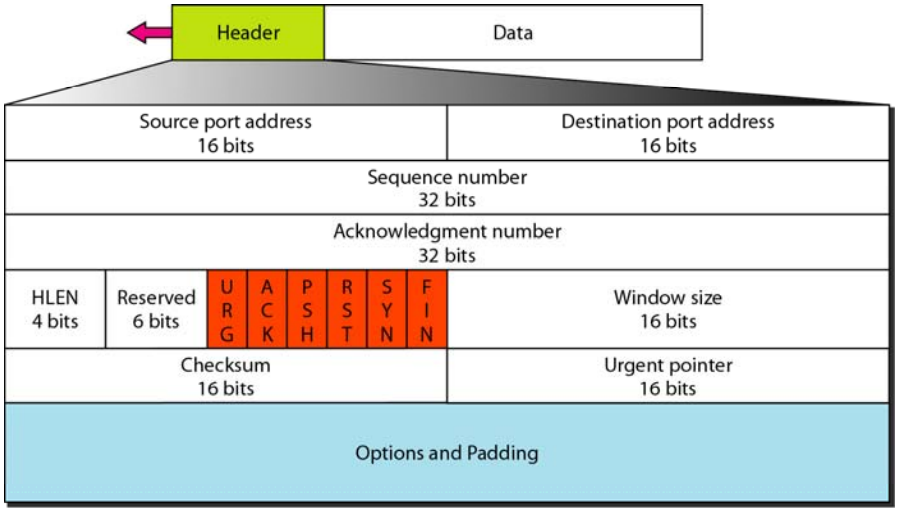


Figure 23.17 *Control field*

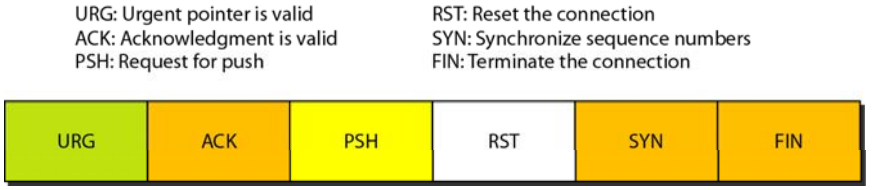
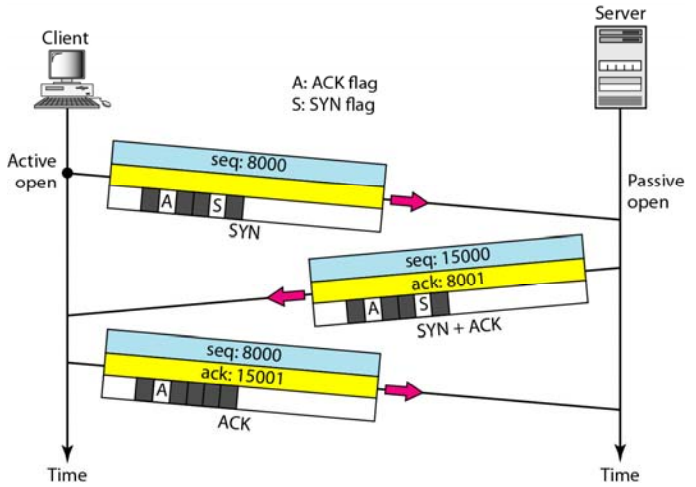


Table 23.3 *Description of flags in the control field*

Flag	*Description*
URG	The value of the urgent pointer field is valid.
ACK	The value of the acknowledgment field is valid.
PSH	Push the data.
RST	Reset the connection.
SYN	Synchronize sequence numbers during connection.
FIN	Terminate the connection.

Figure 23.18 *Connection establishment using three-way handshaking*



Note

A SYN segment cannot carry data, but it consumes one sequence number.

Note

A SYN + ACK segment cannot carry data, but does consume one sequence number.

Note

An ACK segment, if carrying no data, consumes no sequence number.

- **Simultaneous open**
- **SYN flooding attack**

Figure 23.19 *Data transfer*

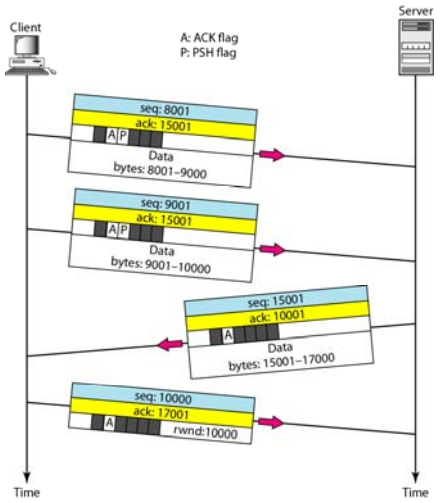
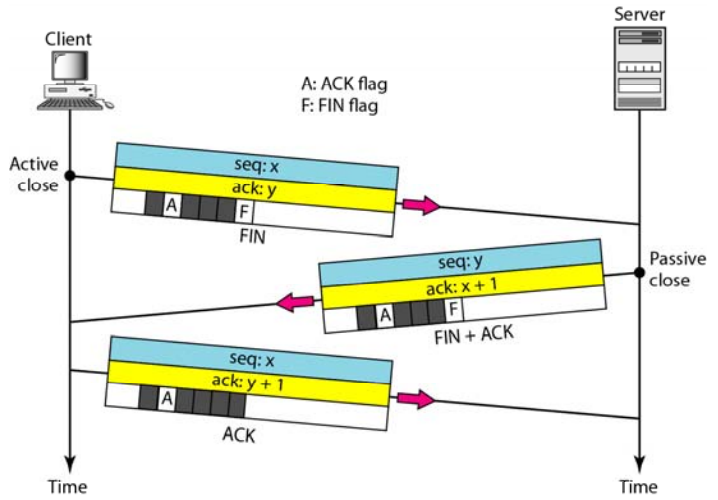


Figure 23.20 *Connection termination using three-way handshaking*



Note

The FIN segment consumes one sequence number if it does not carry data.

Note

The FIN + ACK segment consumes one sequence number if it does not carry data.

Figure 23.21 *Half-close*

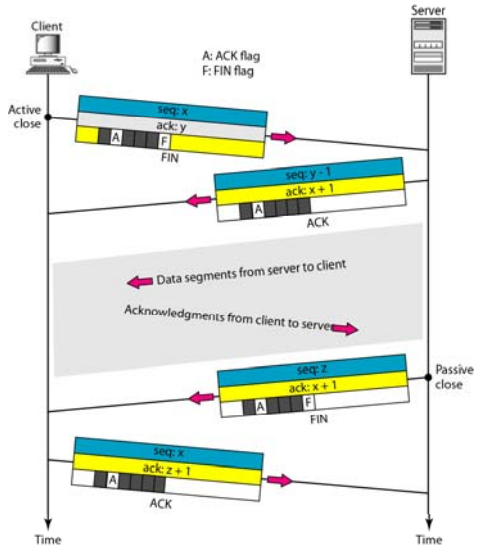
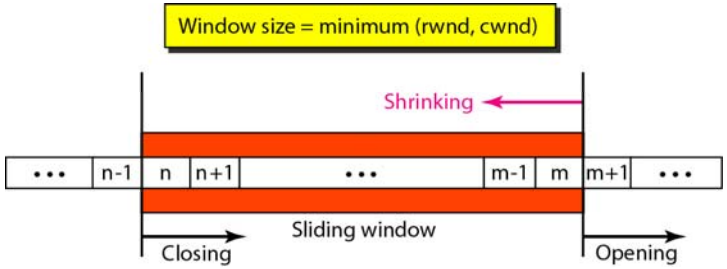


Figure 23.22 *Sliding window*



Note

A sliding window is used to make transmission more efficient as well as to control the flow of data so that the destination does not become overwhelmed with data.
TCP sliding windows are byte-oriented.

Example 23.4

What is the value of the receiver window (rwnd) for host A if the receiver, host B, has a buffer size of 5000 bytes and 1000 bytes of received and unprocessed data?

Solution

The value of rwnd = 5000 − 1000 = 4000. Host B can receive only 4000 bytes of data before overflowing its buffer. Host B advertises this value in its next segment to A.

Example 23.5

What is the size of the window for host A if the value of rwnd is 3000 bytes and the value of cwnd is 3500 bytes?

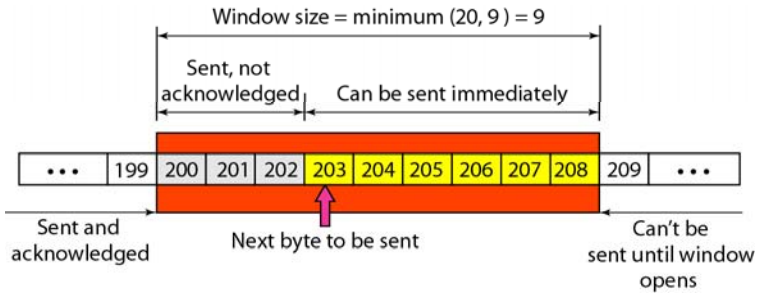
Solution

The size of the window is the smaller of rwnd and cwnd, which is 3000 bytes.

Example 23.6

Figure 23.23 shows an unrealistic example of a sliding window. The sender has sent bytes up to 202. We assume that cwnd is 20 (in reality this value is thousands of bytes). The receiver has sent an acknowledgment number of 200 with an rwnd of 9 bytes (in reality this value is thousands of bytes). The size of the sender window is the minimum of rwnd and cwnd, or 9 bytes. Bytes 200 to 202 are sent, but not acknowledged. Bytes 203 to 208 can be sent without worrying about acknowledgment. Bytes 209 and above cannot be sent.

Figure 23.23 *Example 23.6*



Note

Some points about TCP sliding windows:

- The size of the window is the lesser of rwnd and cwnd.
- The source does not have to send a full window's worth of data.
- The window can be opened or closed by the receiver, but should not be shrunk.
- The destination can send an acknowledgment at any time as long as it does not result in a shrinking window.
- The receiver can temporarily shut down the window; the sender, however, can always send a segment of 1 byte after the window is shut down.

Note

ACK segments do not consume sequence numbers and are not acknowledged.

Note

In modern implementations, a retransmission occurs if the retransmission timer expires or three duplicate ACK segments have arrived.

Note

No retransmission timer is set for an ACK segment.

- **Retransmission after RTO**
- **Retransmission after three duplicate ACK segments.**

- **Data may arrive out of order and be temporarily stored by the receiving TCP, but TCP guarantees that no out-of-order segment is delivered to the process.**

Figure 23.24 *Normal operation*

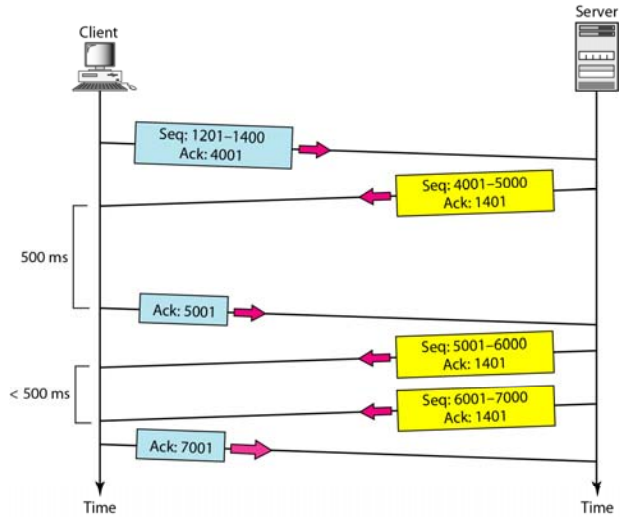
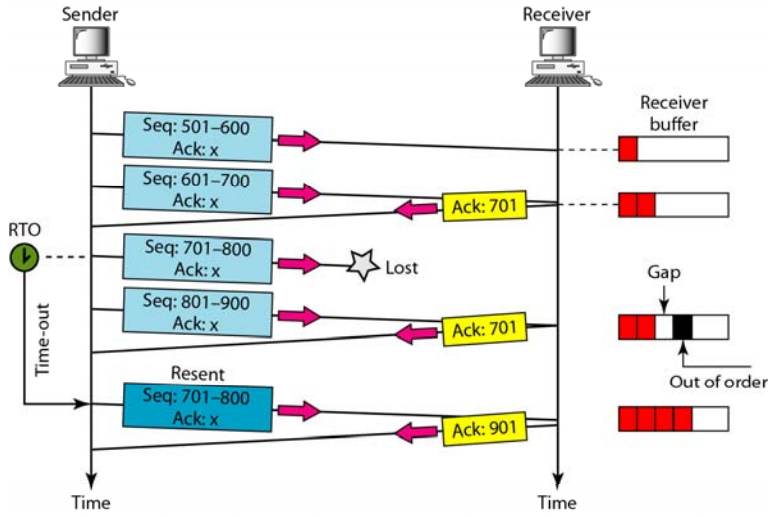


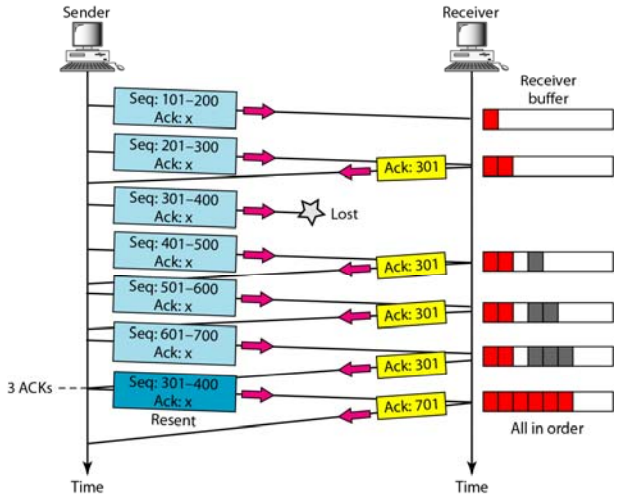
Figure 23.25 *Lost segment*



Note

The receiver TCP delivers only ordered data to the process.

Figure 23.26 *Fast retransmission*



23-4 SCTP

Stream Control Transmission Protocol (SCTP) is a new reliable, message-oriented transport layer protocol. SCTP, however, is mostly designed for Internet applications that have recently been introduced. These new applications need a more sophisticated service than TCP can provide.

Topics discussed in this section:

SCTP Services and Features
Packet Format
An SCTP Association
Flow Control and Error Control

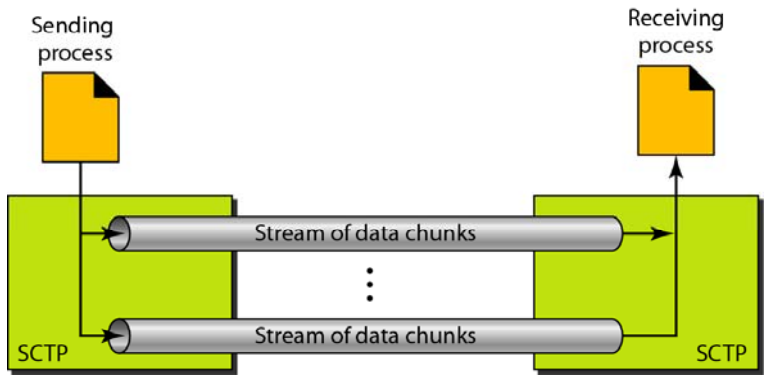
Note

SCTP is a message-oriented, reliable protocol that combines the best features of UDP and TCP.

Table 23.4 *Some SCTP applications*

Protocol	*Port Number*	*Description*
IUA	9990	ISDN over IP
M2UA	2904	SS7 telephony signaling
M3UA	2905	SS7 telephony signaling
H.248	2945	Media gateway control
H.323	1718, 1719, 1720, 11720	IP telephony
SIP	5060	IP telephony

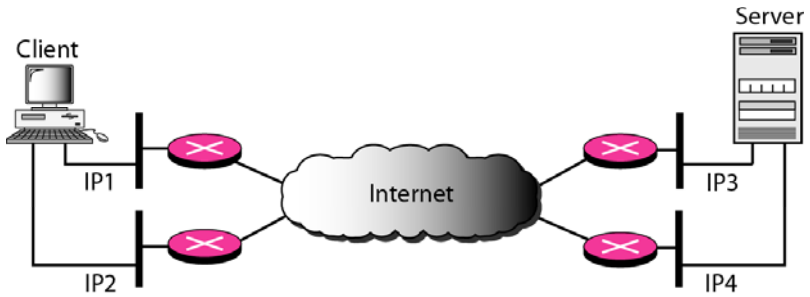
Figure 23.27 *Multiple-stream concept*



Note

An association in SCTP can involve multiple streams.

Figure 23.28 *Multihoming concept - fault tolerance*



Note

SCTP association allows multiple IP addresses for each end.

- **SCTP provides Full-duplex communication**
- **Connection-oriented service**
- **Reliable service**
- **In SCTP, a data chunk is numbered using a TSN.**

Note

To distinguish between different streams, SCTP uses an SI.

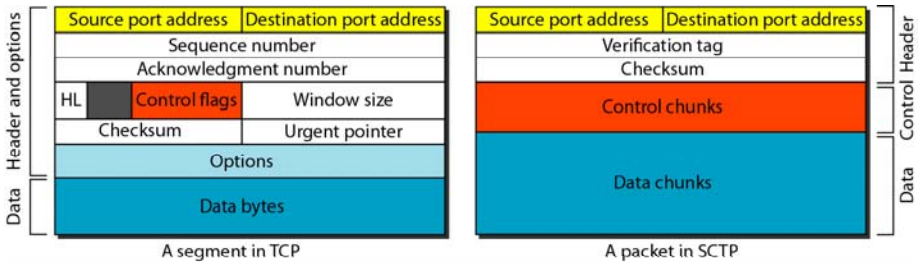
Note

To distinguish between different data chunks belonging to the same stream, SCTP uses SSNs.

Note

TCP has segments; SCTP has packets.

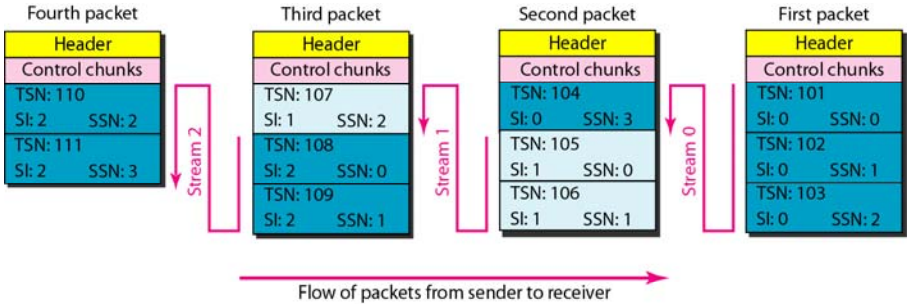
Figure 23.29 *Comparison between a TCP segment and an SCTP packet*



Note

In SCTP, control information and data information are carried in separate chunks.

Figure 23.30 *Packet, data chunks, and streams*



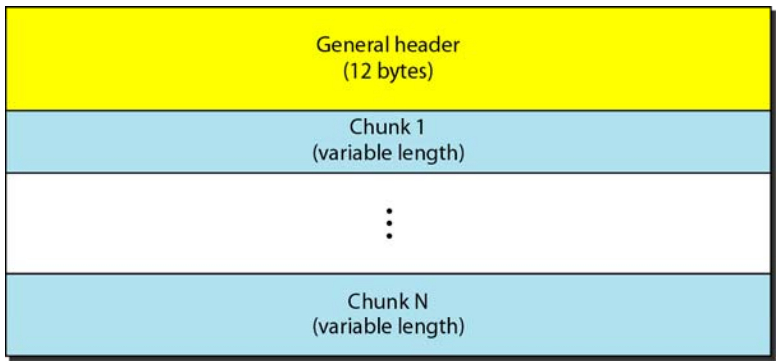
Note

Data chunks are identified by three items: TSN, SI, and SSN. TSN is a cumulative number identifying the association; SI defines the stream; SSN defines the chunk in a stream.

Note

In SCTP, acknowledgment numbers are used to acknowledge only data chunks; control chunks are acknowledged by other control chunks if necessary.

Figure 23.31 *SCTP packet format*



Note

In an SCTP packet, control chunks come before data chunks.

Figure 23.32 *General header*

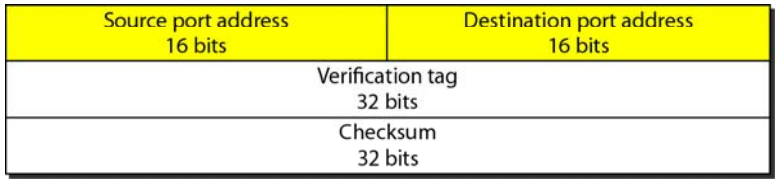


Table 23.5 *Chunks*

Type	*Chunk*	*Description*
0	DATA	User data
1	INIT	Sets up an association
2	INIT ACK	Acknowledges INIT chunk
3	SACK	Selective acknowledgment
4	HEARTBEAT	Probes the peer for liveliness
5	HEARTBEAT ACK	Acknowledges HEARTBEAT chunk
6	ABORT	Aborts an association
7	SHUTDOWN	Terminates an association
8	SHUTDOWN ACK	Acknowledges SHUTDOWN chunk
9	ERROR	Reports errors without shutting down
10	COOKIE ECHO	Third packet in association establishment
11	COOKIE ACK	Acknowledges COOKIE ECHO chunk
14	SHUTDOWN COMPLETE	Third packet in association termination
192	FORWARD TSN	For adjusting cumulative TSN

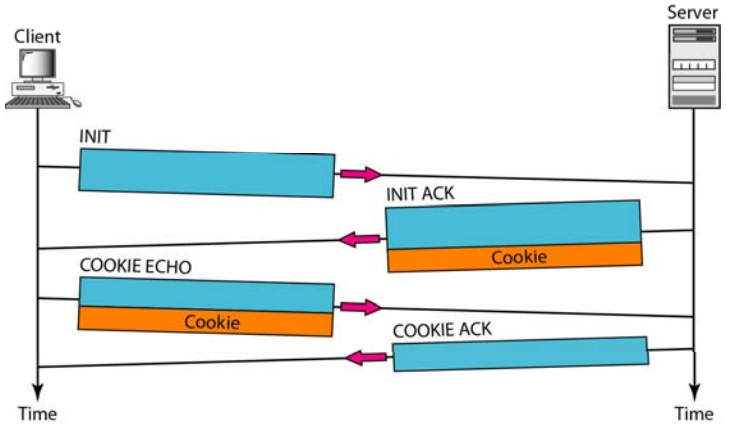
Note

A connection in SCTP is called an association.

Note

No other chunk is allowed in a packet carrying an INIT or INIT ACK chunk. A COOKIE ECHO or a COOKIE ACK chunk can carry data chunks.

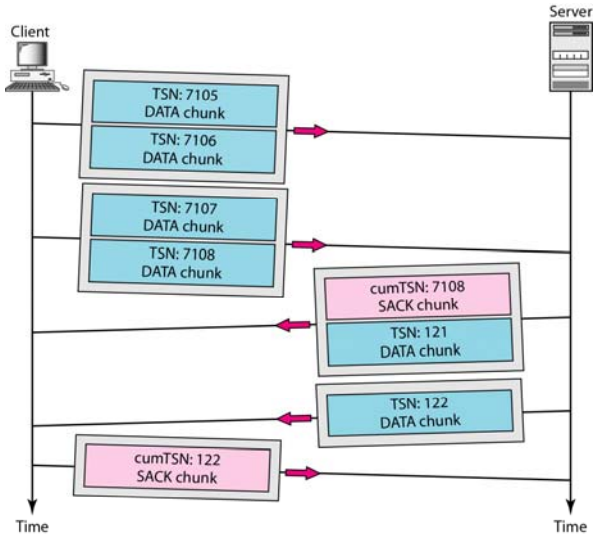
Figure 23.33 *Four-way handshaking*



Note

In SCTP, only DATA chunks consume TSNs; DATA chunks are the only chunks that are acknowledged.

Figure 23.34 *Simple data transfer*



Note

The acknowledgment in SCTP defines the cumulative TSN, the TSN of the last data chunk received in order.

Figure 23.35 *Association termination*

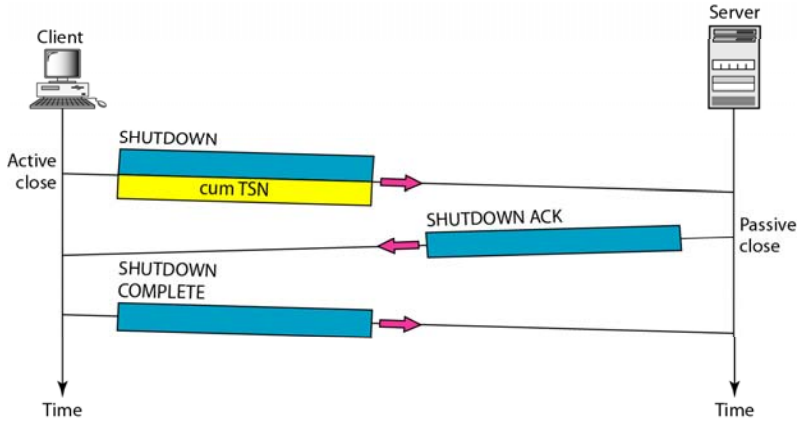


Figure 23.36 *Flow control, receiver site*

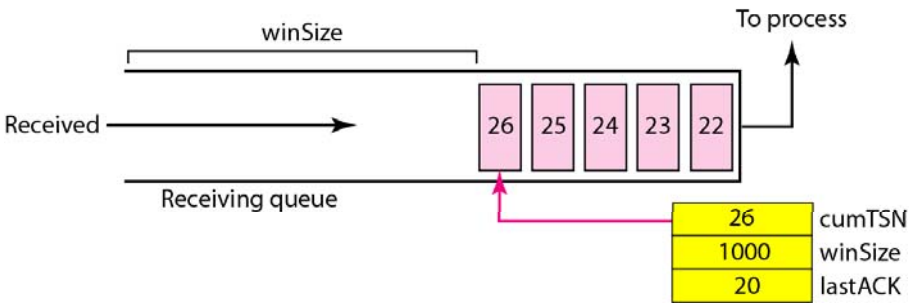


Figure 23.37 *Flow control, sender site*

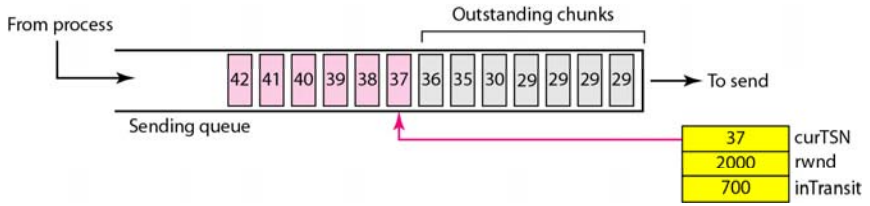


Figure 23.38 *Flow control scenario*

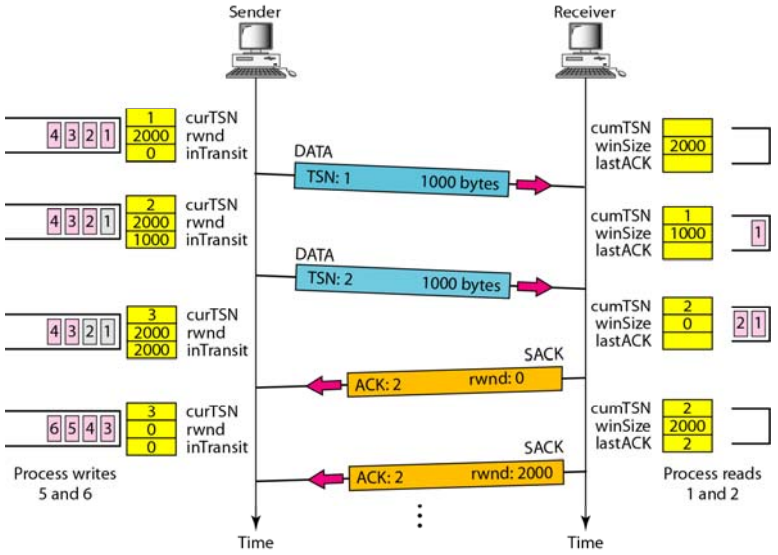


Figure 23.39 *Error control, receiver site*

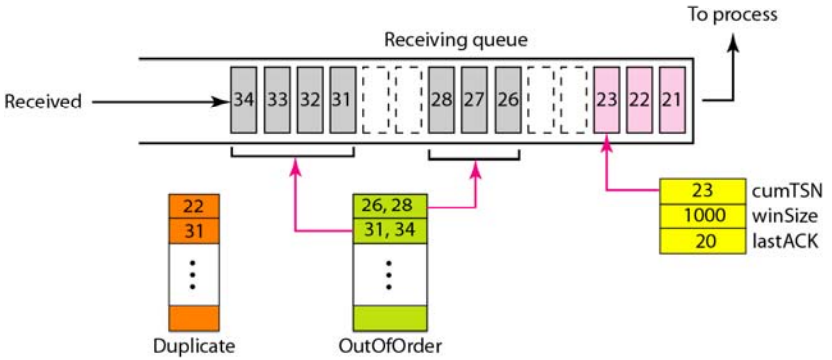


Figure 23.40 *Error control, sender site*