Data Communication and Computer Network

Transport Layer (TCP, UDP)

Transport Layer functions

- ❑ Possible transport layer functions:
  - ➢ Process-to-process delivery
  - ➢ Establish and maintain end-to-end connections among processes (explicitly establish and terminate connections)
  - ➢ Guarantee reliable, in-order end-to-end transfer
  - ➢ Provide end-to-end flow control
  - ➢ Congestion control

Uses port numbers to identify processes.

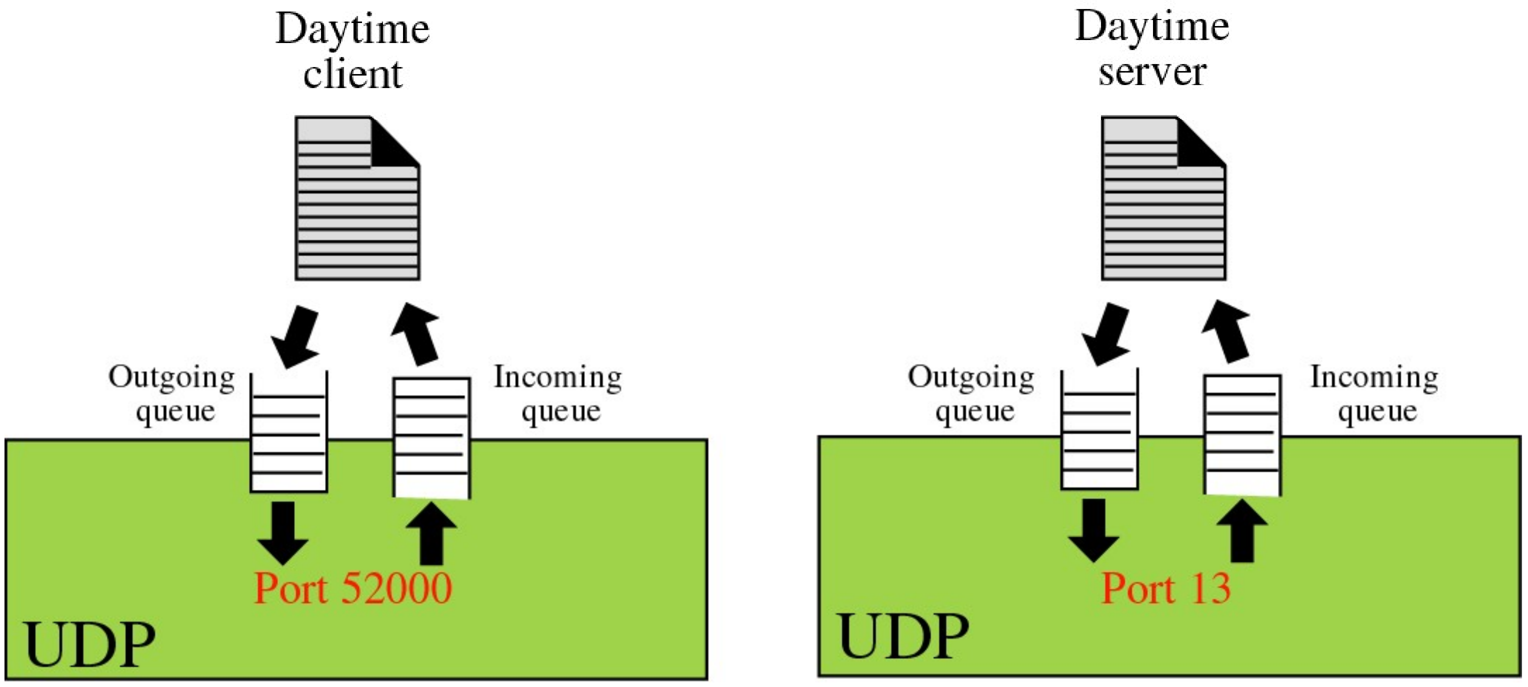
TCP numbers packets and expects acknowledgments (ACKs).

Uses a sliding window mechanism in TCP.

- ❑ UDP provides only the first function
- ❑ TCP provides all of the above

Ports

- Port: a 16-bit integer, used to identify an application
- Ports typically implemented as a pair of queues: an incoming queue and an outgoing queue



Classification of Ports

- Well-known / reserved ports: ports up to 1023
  - Normally used for protocols with wide applicability
  - Registered by ICANN (Internet Corporation for Assigned Names and Numbers)
  - E.g. ftp – 21, 20, telnet – 23, web server – 80
- Registered ports: ports 1024 – 49151
  - Used to avoid port collisions between user-level applications developed independently when installed on the same machine
  - Registered by IANA (Internet Assigned Numbers Authority)
  - E.g. vlc media player – 1234, RADIUS authentication protocol - 1812
- Dynamic or private ports: ports 49152 – 65535

  Can be used by any application
  - cannot be registered with IANA
  - temporary purposes and for automatic allocation

Endpoints and Connections

- ❑ Endpoint
  - ➢ a 2-tuple <host IP, port>
  - ➢ Commonly called a socket
- ❑ Connection
  - ➢ Defined by two endpoints
  - ➢ Two connections will have at least one endpoint different (but can have one endpoint same)
- ❑ Messages de-multiplexed based on
  - ➢ Connections in TCP (TCP maintains connections)
  - ➢ Endpoint in UDP (no connection maintained)

Transmission Control Protocol (TCP)

Transmission Control Protocol

- Guarantees reliable, in-order delivery of a stream of bytes between sender and receiver applications
- Connection oriented (establishment, termination)
- Full-duplex protocol
  - Each TCP connection supports a pair of byte streams, one flowing in each direction
- Flow-control mechanism for both byte streams
  - Receiver can limit how many bytes the sender can send at a given time
- Congestion-control mechanism
  - Control how fast sender can send data, to prevent the sender from overloading the network

Stream, segment, sequence number

- Data sent by application process is viewed as a stream (sequence) of bytes

- Segment – the unit of transfer between TCP modules on two hosts
  - stream of bytes divided into segments, each segment given a TCP header, and given to IP module to send

- Each TCP segment has a sequence number to specify position of the first byte in the segment within the byte stream

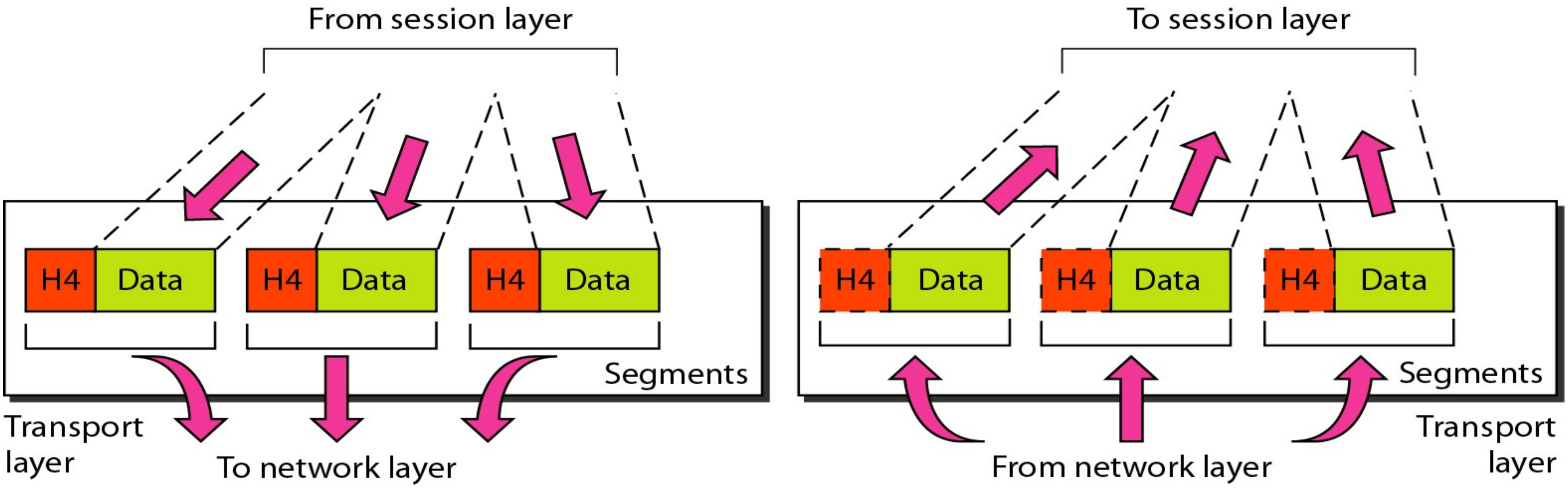
Maximum Segment Size (MSS)

- TCP *usually* sets MSS to the size of the largest segment that can be sent, without causing the local IP to fragment it

- For example, if both nodes lie in same network,
  - MSS = (MTU of underlying network – IP header size – TCP header size)

- A TCP module can negotiate with the TCP module at the other end to specify the MSS that it is willing to receive (MSS announcement in SYN using TCP option during three way handshaking)

TCP Segmentation & Reassembly

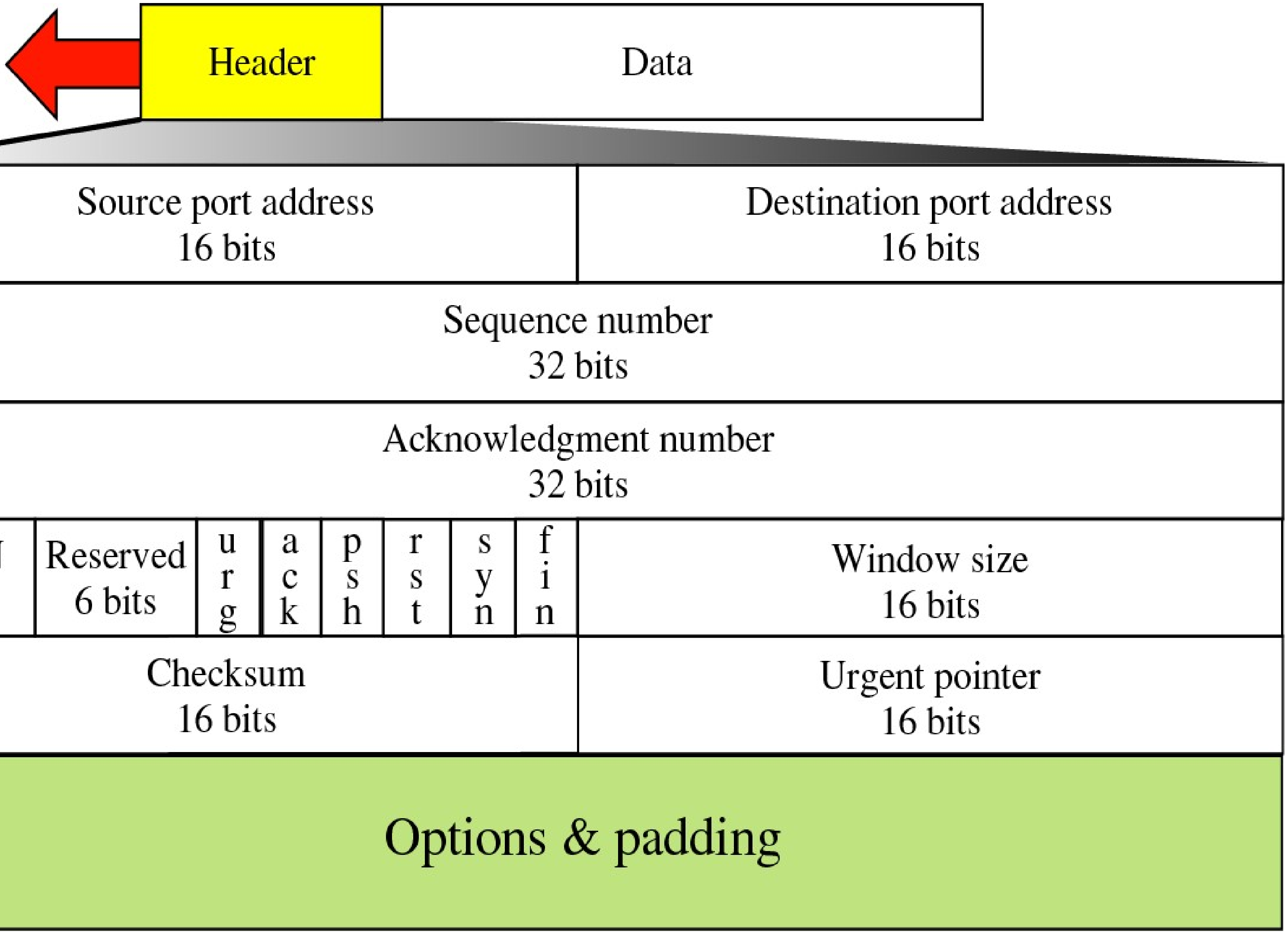


Maximum Segment Lifetime (MSL)

- How late can a segment arrive at destination?
  - Routers using IP discard packets if TTL expires
  - TCP assumes each packet has a maximum lifetime
  - MSL currently taken to be 120 seconds
  - Just an estimate used by TCP, not guaranteed by IP

Why do we need MSL?
Because TCP wants to avoid duplicate segments from a previous connection being accepted in a new connection (which can happen if ports/IPs are reused quickly).

TCP segment header



Header

Data

Source port address
16 bits

Destination port address
16 bits

Sequence number
32 bits

Acknowledgment number
32 bits

HLEN
4 bits

Reserved
6 bits

urg

ack

psh

rst

syn

fin

Window size
16 bits

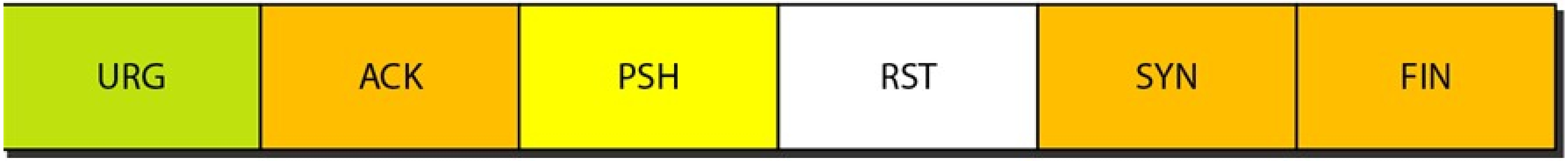
Checksum
16 bits

Urgent pointer
16 bits

Options & padding

Flags in TCP header

- **URG** – urgent pointer is valid in this segment
- **ACK** – the acknowledgement number is valid
- **PSH** – sender application requests sending TCP module to send all data accumulated in buffer
- **RST** – indicates the connection has been reset
- **SYN** – synchronize the sequence numbers to establish a connection
- **FIN** – application has finished sending data, wants to close connection



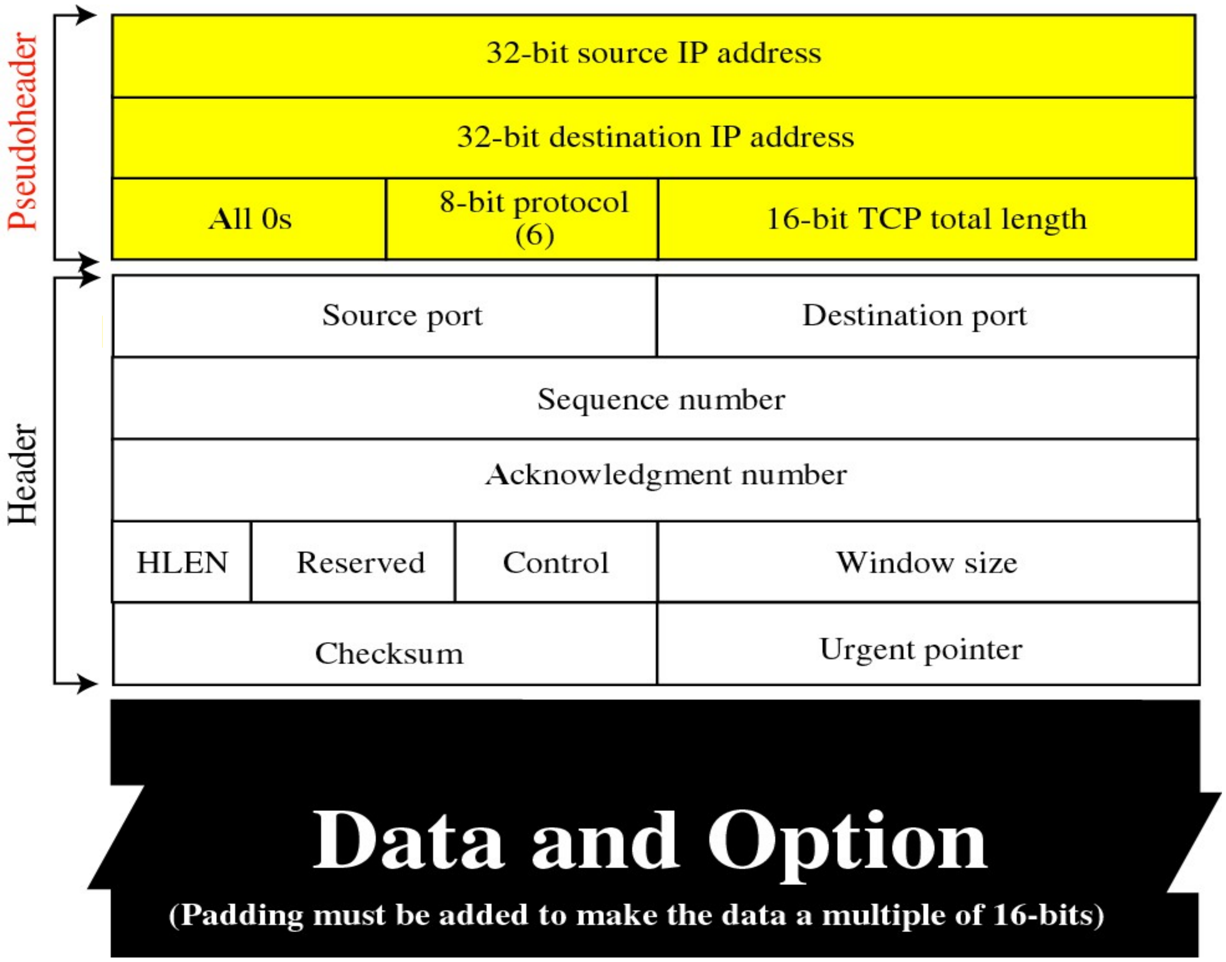
Checksum in TCP

- TCP checksum algorithm similar to that in IP
- TCP computes checksum over data, TCP header and a pseudo-header

TCP checksum is stronger than IP checksum.

- TCP pseudo-header consists of
  - TCP length field
  - Three fields from the IP header: protocol number (6 for TCP), source IP address, destination IP address
  - One octet of zeroes to pad the segment to an exact multiple of 16 bits

TCP pseudo-header



Pseudoheader

32-bit source IP address

32-bit destination IP address

All 0s

8-bit protocol (6)

16-bit TCP total length

Header

Source port

Destination port

Sequence number

Acknowledgment number

HLEN

Reserved

Control

Window size

Checksum

Urgent pointer

Data and Option

(Padding must be added to make the data a multiple of 16-bits)

TCP pseudo-header (contd.)

- ❑ Pseudo-header (and the octet used for padding) are NOT transmitted along with the TCP segment, nor are they included in the length

- ❑ Receiver TCP module also prepends pseudo-header and compares checksum

- ❑ Why use pseudo-header?
  - ➢ To verify that this segment has been delivered between the correct two endpoints

Connection Establishment

- Purpose: both sides needs to know
  - that the other side is ready for data transfer, the port used by the other side, MSS, etc
  - the other side's Initial Sequence Number (ISN)
- First byte address cannot always be 0 or 1
  - Protect against two incarnations of the same connection re-using the same ISN too soon
  - TCP specification: each side of a connection selects an ISN at random
  - For example, ISN set from a 32-bit clock that ticks every 4 microseconds (wraps around once every 4.55 hours)

Connection establishment (contd.)

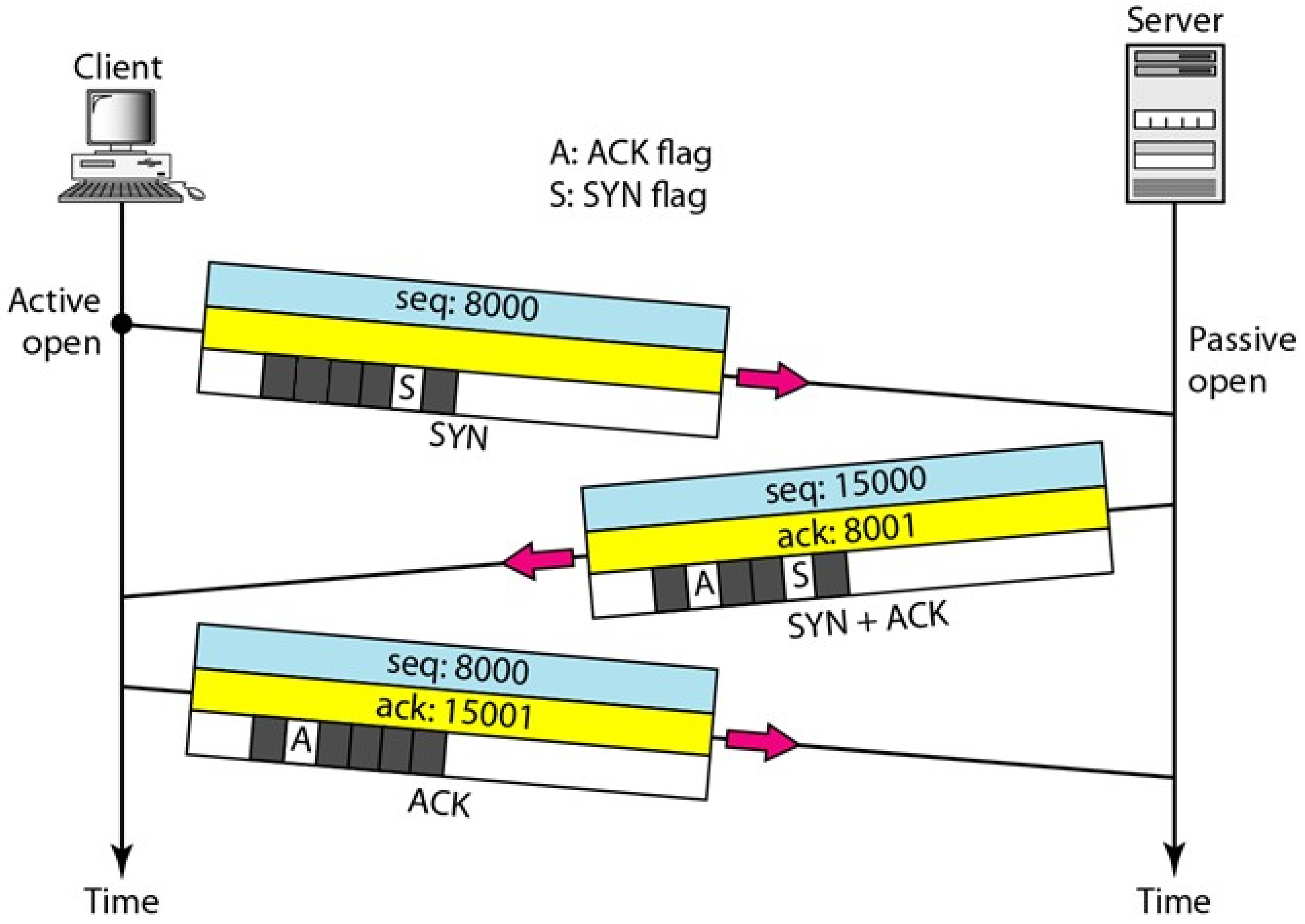
3-way handshake

- ❑ Client sends SYN segment (no data)
  - ➢ specifies port numbers, etc
  - ➢ Sequence Number: client’s ISN isn_c
- ❑ Server responds with a SYN + ACK segment
  - ➢ Sequence Number: server’s ISN isn_s
  - ➢ Acknowledgement number: isn_c + 1
- ❑ Client sends ACK segment
  - ➢ Acknowledgement number: isn_s + 1
- ❑ The side sending the first SYN is said to perform an active open. The other side performs a passive open

Why "Passive Open"?
The server is referred to as passive because it does not initiate the connection.

Why "Active Open"?
The client in the connection setup is called the "active" participant because it actively initiates the connection.

Connection establishment using three-way handshaking



Basic TCP data transfer

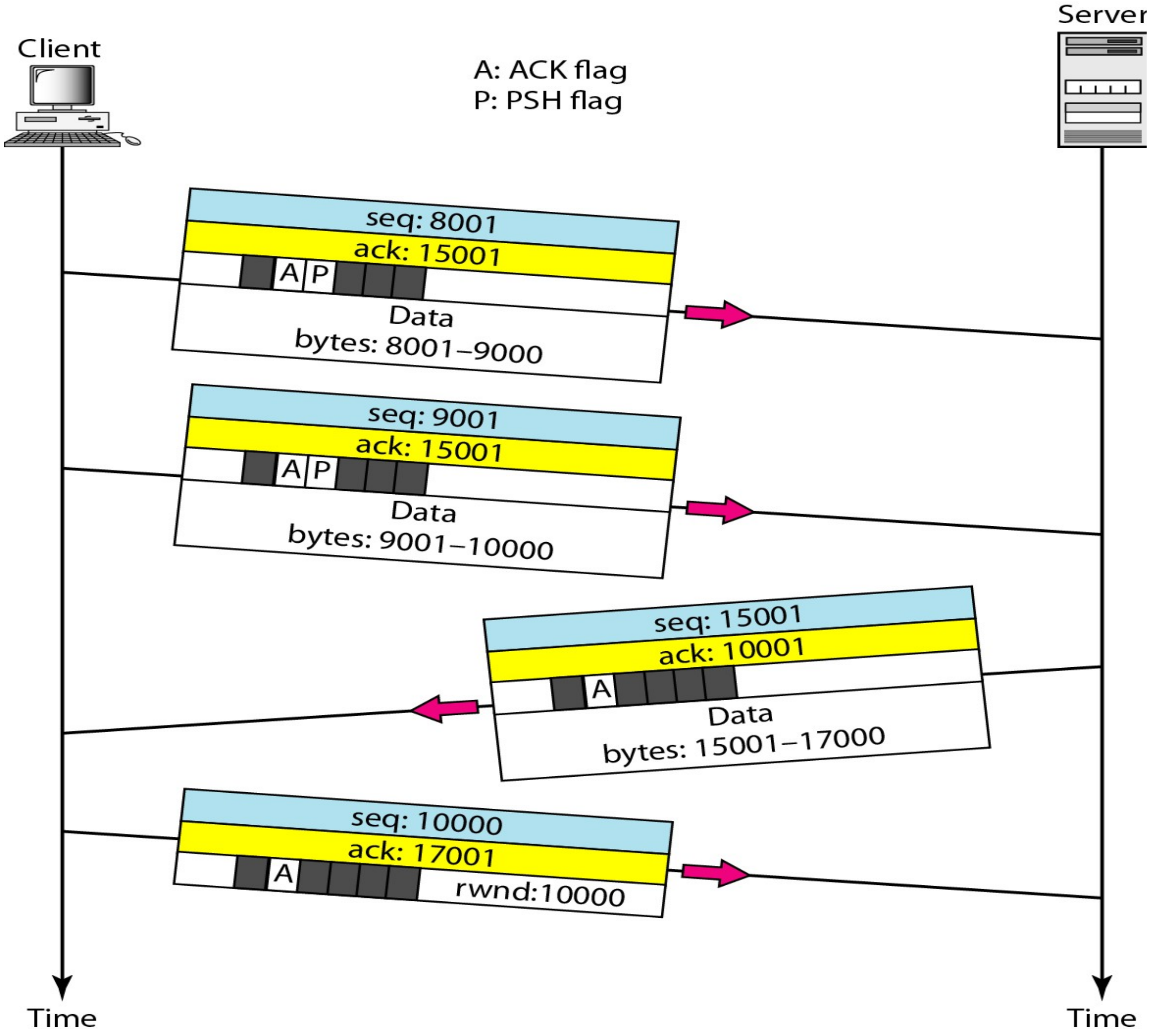
- ❑ Once connection is established between sender TCP module and receiving TCP module …
- ❑ TCP modules send and receive TCP segments with data
  - ➢ Each segment contains a sequence number, source and destination port
- ❑ Acknowledgments sent by TCP modules to other side
  - ➢ Indicated by ACK flag set to 1 and a valid acknowledgement number field: the next sequence number the receiver expects
  - ➢ TCP uses cumulative, positive ACK
  - ➢ ACK frame may have no data, or ACK may be piggybacked over data segments
- ❑ When data transfer over, close connection

Cumulative ACK:

TCP uses cumulative positive acknowledgment, meaning that an acknowledgment for a specific sequence number also acknowledges all the previous sequence numbers.

For example, if the receiver sends an acknowledgment with the number 1000, it means that the receiver has successfully received all data up to, but not including, sequence number 1000.

Once data transfer is over



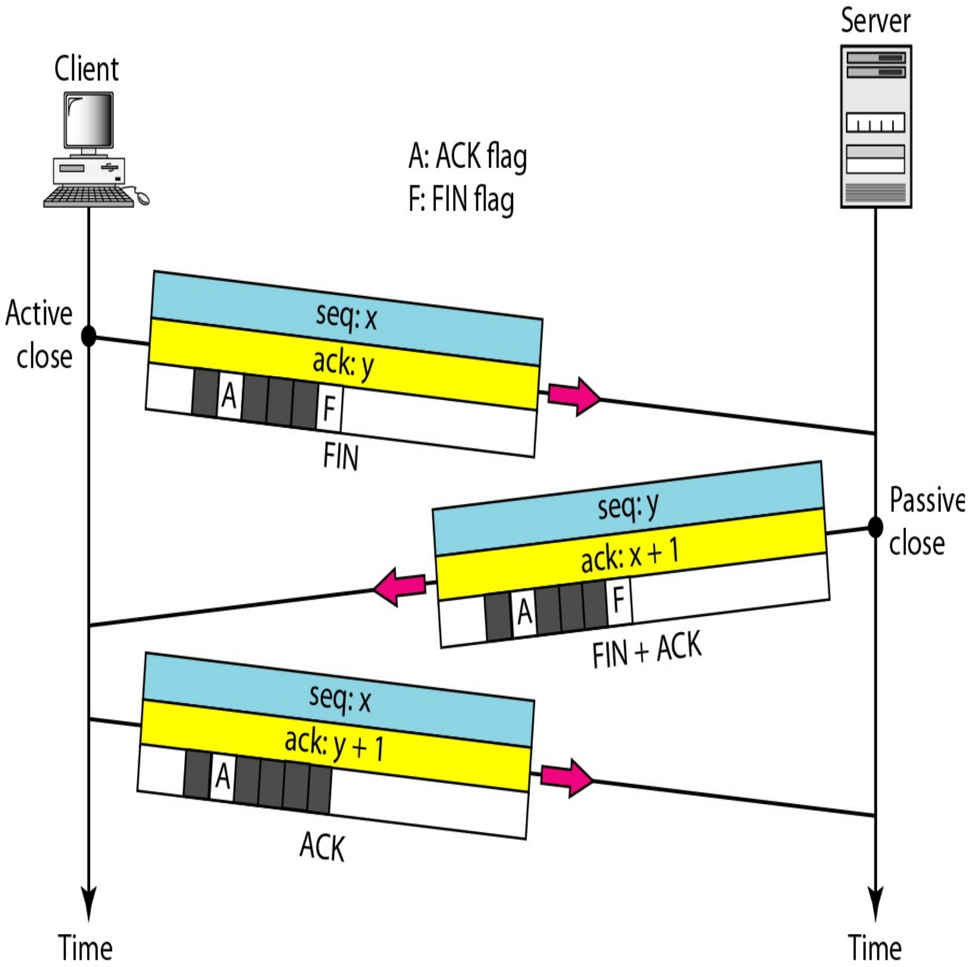
Connection Termination

- Any of the two parties involved in exchanging data (client or server) can close the connection, although it is usually initiated by the client.
- Most implementations today allow two options for connection termination:
  - three-way handshaking and
  - four-way handshaking with a half-close option.

Normal Connection Termination

- The client TCP, after receiving a close command from the client process, sends a FIN segment in which the FIN flag is set.
  - A FIN segment can include the last chunk of data sent by the client or it can be just a control segment
- The server TCP, after receiving the FIN segment, informs its process of the situation and sends a FIN +ACK segment, to confirm the receipt of the FIN segment from the client and at the same time to announce the closing of the connection in the other direction.
  - This segment can also contain the last chunk of data from the server. If it does not carry data, it consumes only one sequence number.
- The client TCP sends the last segment, an ACK segment, to confirm the receipt of the FIN segment from the TCP server.

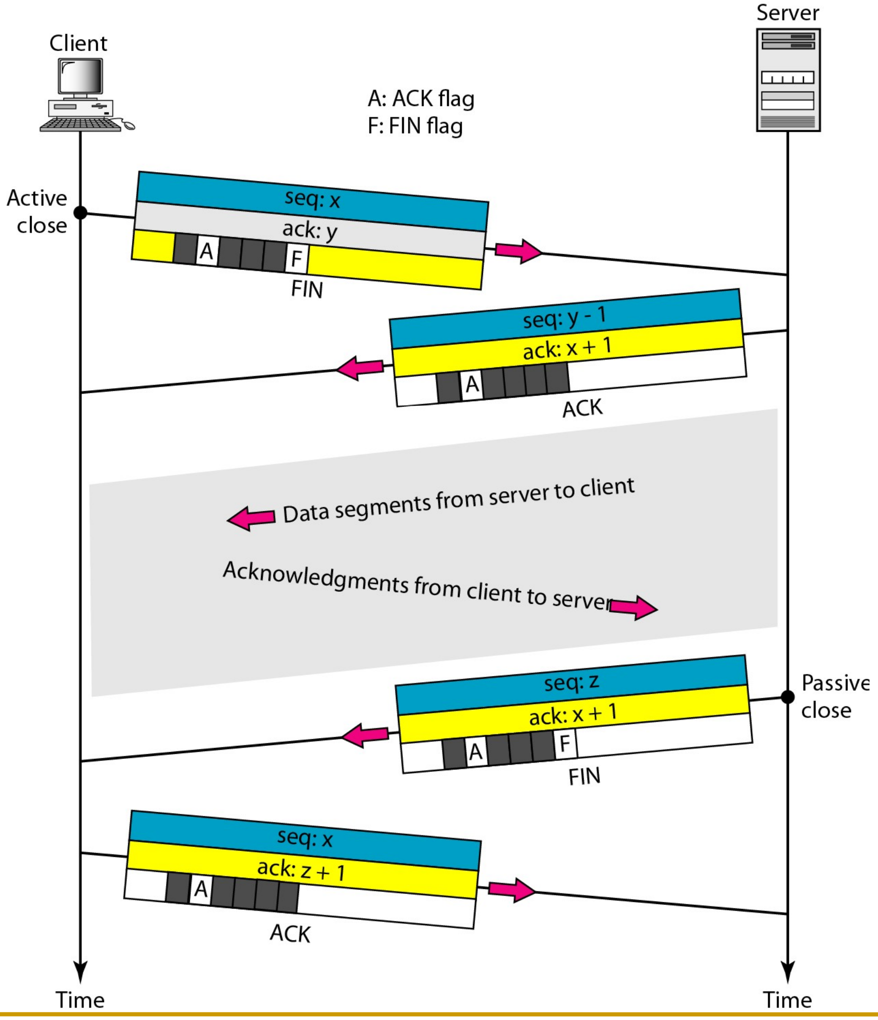
Connection termination using three-way handshaking (normal)



Half-close Connection Termination

- A duplex connection treated as two one-way connections, each to be closed separately
- Either side can initiate termination
  - TCP module sends FIN segment with a sequence number
  - Indicates that the application process on the side sending FIN will not send any more data
- The other side acknowledges FIN segment
  - TCP module receiving FIN sends an ACK segment with ACK number = FIN segment sequence number + 1
  - This side can go on sending data
- The above process repeats when the other side wants to close connection

Connection Termination - four-way handshaking (Half-close)



CLOSE-WAIT and TIME-WAIT states

- CLOSE-WAIT: Indicates that the remote endpoint (other side of the connection) has closed the connection
  - Waiting for a connection termination request from the local user.

- TIME-WAIT: Indicates that local endpoint (this side) has closed the connection.
  - Waiting for enough time to pass to be sure that all remaining packets on the connection have expired.

TIME-WAIT state

- After both sides have sent FIN segments and both sides have been ACK-ed by the other side, connection placed in TIME-WAIT state
- Connection remains in the TIME-WAIT state for 2*MSL (Maximum Segment Lifetime) before data structures for this connection are removed.
  - MSL is typically two minutes
- Why TIME-WAIT state?
  - a transitional state where a connection is kept open for a short period after the application closes a socket to ensure that any delayed or out-of-order packets are handled correctly.

States for TCP Connection

State	Description
CLOSED	There is no connection.
LISTEN	The server is waiting for calls from the client.
SYN-SENT	A connection request is sent; waiting for acknowledgment.
SYN-RCVD	A connection request is received.
ESTABLISHED	Connection is established.
FIN-WAIT-1	The application has requested the closing of the connection.
FIN-WAIT-2	The other side has accepted the closing of the connection.
TIME-WAIT	Waiting for retransmitted segments to die.
CLOSE-WAIT	The server is waiting for the application to close.
LAST-ACK	The server is waiting for the last acknowledgment.

RST flag

- Used to reset a connection (abnormal close)

- TCP module on one side sends a segment with the RST bit set
  - TCP module on the other side responds to a RST segment by aborting the connection immediately, and informs the application that a reset occurred

- RST flag cannot be set by the applications, it is set by the TCP modules

Out of Band Data

- Sender application may want some data to be delivered to the receiving application urgently (before what was sent previously)
  - TCP module sends segment with URG flag set to 1
  - Location of urgent data identified by the Urgent Pointer field

- When this segment reaches receiver TCP module
  - Receiver TCP module will notify receiver application and hands over the urgent data

TCP Flow Control and Error Control

Sliding window flow control

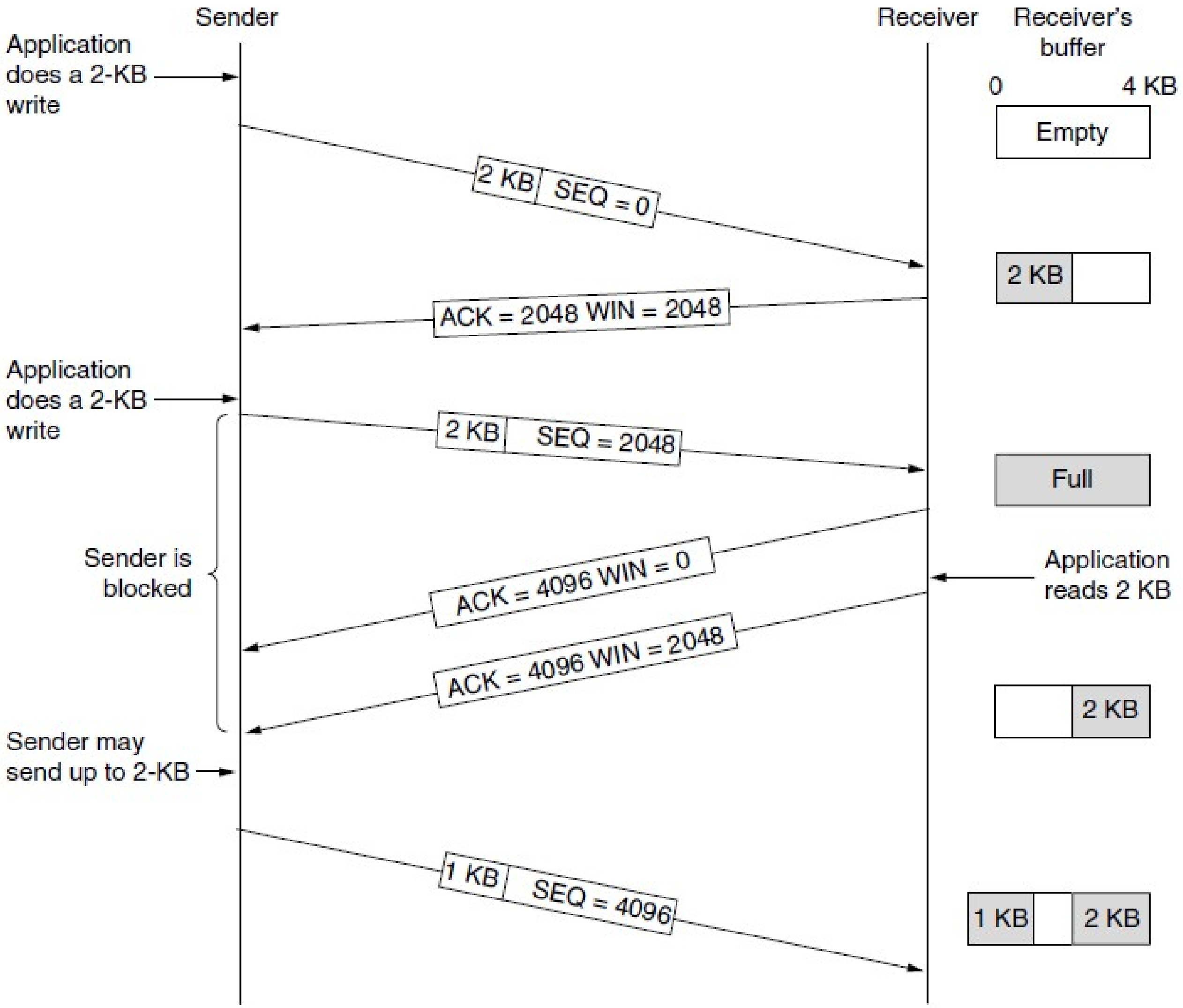
- TCP uses a byte-level sliding window flow control scheme with cumulative positive ACKs
  - Sender TCP module maintains a window of size *w* and start of window *X*
  - Can send up to *n* bytes starting from byte *X* without receiving an acknowledgement
- Window size at sender determines how much unacknowledged data can the sender send
- Also takes care of error control (segments re-transmitted if corrupted or lost)

Problem at the receiver

- ❑ Data remains in buffer of receiver TCP module till the receiver application reads data
- ❑ So, depending on how fast the application is reading the data, the receiver's window size may change

- ❑ Solution: advertised window
  - ➢ In every segment (data or ACK or both) sent to the sender, receiver tells its current window size
  - ➢ Sender uses this value instead of a fixed receiver window size

Advertised window – example



Silly Window Syndrome occurs when the receiver advertises a very small window size, causing the sender to send small segments with very few bytes of data, leading to inefficiency and high overhead due to the large number of small segments.

Silly window syndrome

- ❑ Receiver TCP module advertising small window, sender TCP module sending segments with few bytes of data, …
  - ➢ large overhead
- ❑ Avoiding silly window at receiver
  - ➢ Once zero window size has been advertised, receiver must wait until 'significant amount' of buffer empty
  - ➢ Receiver delays sending ACK if the window size is not sufficiently large to advertise (up to 500 msec)
- ❑ Avoiding silly window at sender
  - ➢ When to send segments? Nagle's algorithm

Nagle’ s algorithm

When the application produces additional data to send:

```
if  no data in flight
    send segment immediately (with as much data as allowed by advertised window W)
else        # there is un-ACKed data in flight
    buffer new data until available data >= MSS, and then send segment with as much data as allowed by W
    if ACK received while waiting to send
        send all data accumulated in buffer (or as much as is allowed by W) now
    end if
end if
```

Push flag in TCP header

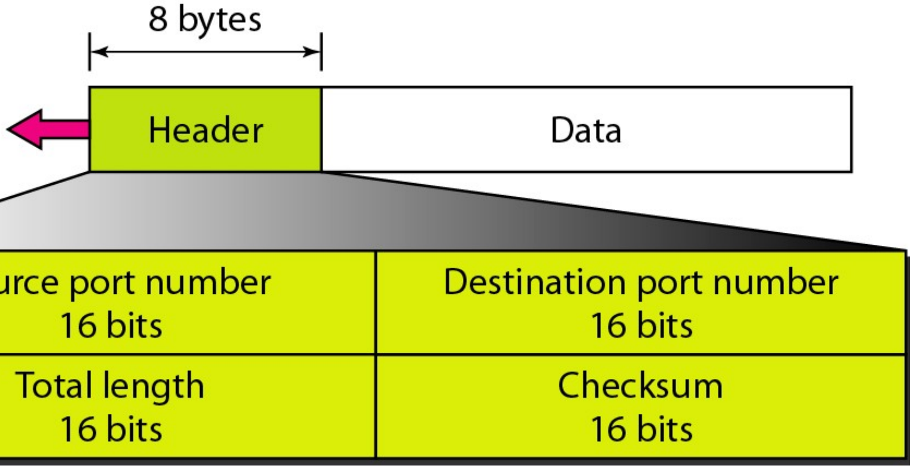
- Normally, TCP module decides when to send a segment

- Sender application can tell sender TCP module to send (flush) whatever bytes it has collected
  - TCP module sends segment with PSH flag set to 1
  - When segment with PSH=1 reaches receiver TCP module, data given to receiving application immediately

User Datagram Protocol (UDP)

UDP

- ❑ Simply extends the unreliable host-to-host delivery service of underlying network into a process-to-process communication channel
- ❑ Identification for each process: <host IP, port>
- ❑ Connectionless, each UDP segment handled independent of others
- ❑ UDP does NOT implement flow control, reliable or ordered delivery



8 bytes

Header

Data

Source port number 16 bits	Destination port number 16 bits
Total length 16 bits	Checksum 16 bits

UDP header

Fields in the UDP header

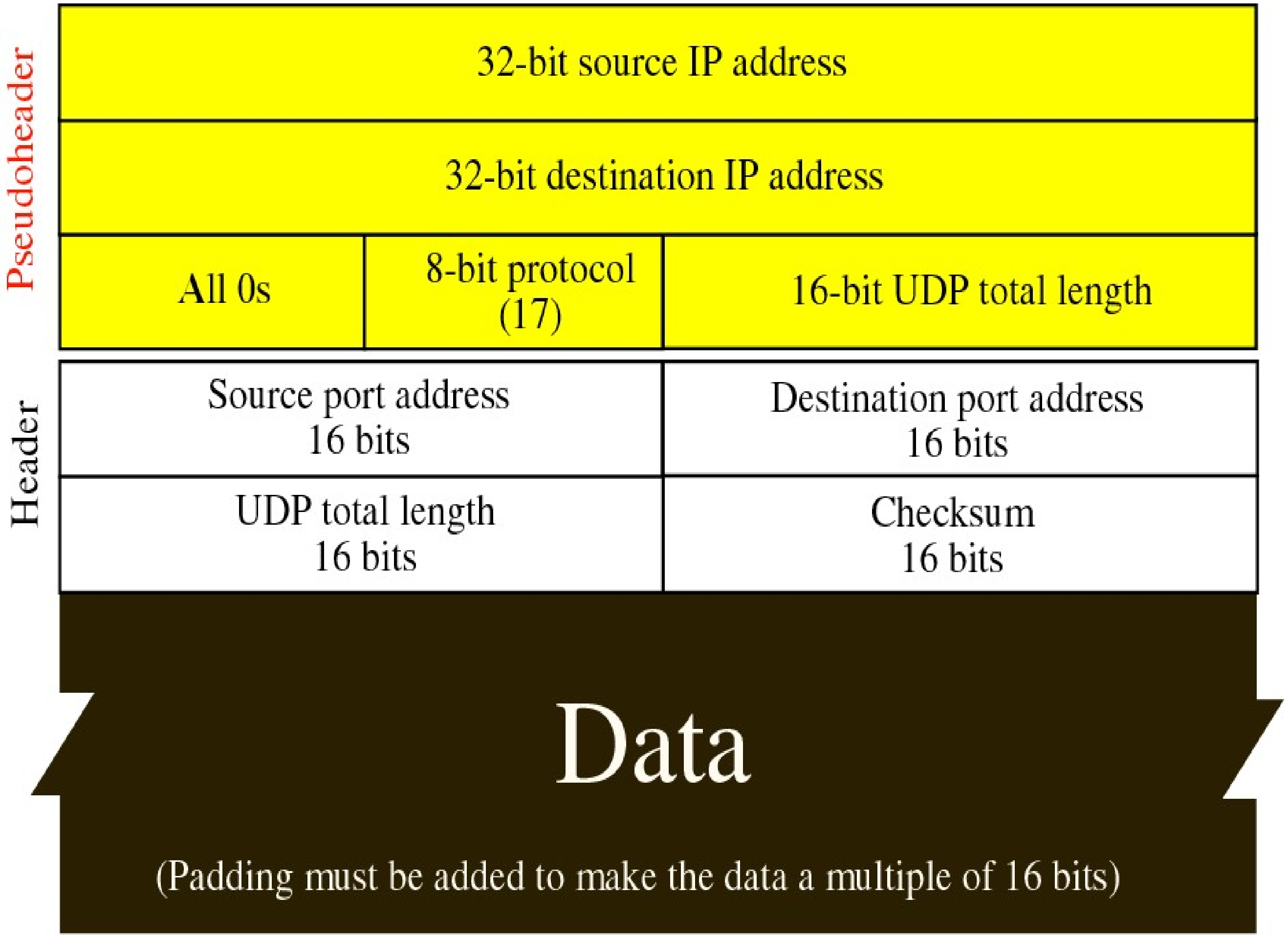
- Source port, destination port

  16-bit UDP protocol port numbers used to demultiplex datagrams (along with IP address)

- Length field: number of octets in the UDP datagram, including the UDP header and data
  - Minimum value is 8

- Checksum: same method as in TCP
  - Checksum computed over the UDP header, the contents of the message body, and a pseudo-header

Pseudo-header added to UDP datagram



32-bit source IP address

32-bit destination IP address

All 0s | 8-bit protocol (17) | 16-bit UDP total length

Source port address 16 bits | Destination port address 16 bits

UDP total length 16 bits | Checksum 16 bits

Data

(Padding must be added to make the data a multiple of 16 bits)

Why is there a UDP?

- Since TCP is decidedly better than UDP, why is there a UDP at all?
  - UDP is suitable for purposes where error checking and correction are either not necessary or are performed in the application; UDP avoids the overhead of such processing at the level of the network interface.
  - Time-sensitive applications often use UDP because dropping packets is preferable to waiting for delayed packets (VoIP), which may not be an option in a real-time system
- Some advantages of UDP
  - Low overhead
  - Simple, requires very little resources
  - Fine control over when data is sent

References

- *Data Communications & Networking, 5th Edition, Behrouz A. Forouzan*
- *Computer Networks, Andrew S. Tanenbaum and David J. Wetherall*
- *Wikipedia*