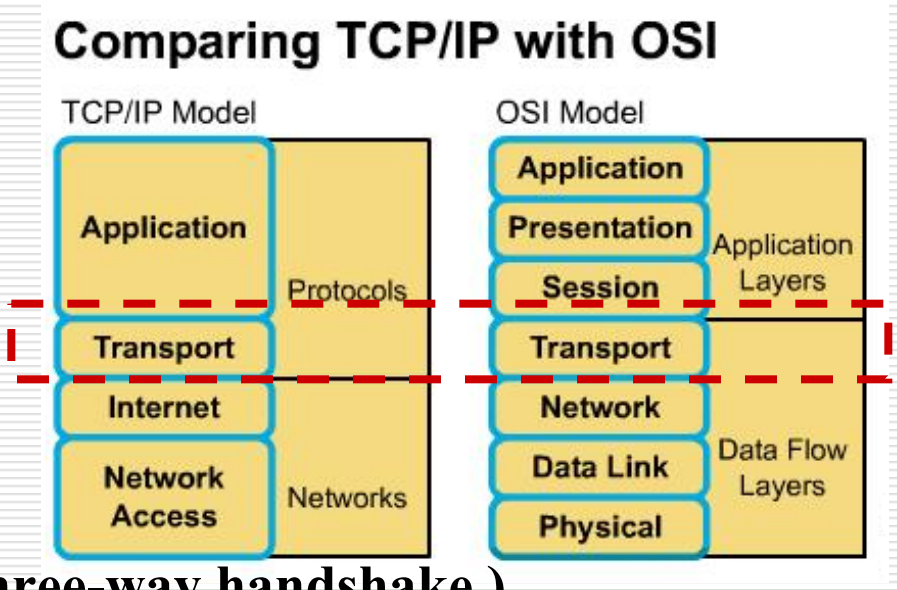
# Chapter6 Transport Layer

王昊翔：hxwang@scut.edu.cn

School of Computer Science & Engineering ,SCUT

# Outline

- Overview of Transport layer
- General concept
- Transport layer protocol
  - UDP
  - TCP
    - Segment format
    - Connection establish (three-way handshake )
    - Connection release
    - How to provide data reliable transport?



# Overview of transport layer

- The transport layer is the heart of the whole protocol hierarchy.
- Its task is to provide reliable, cost-effective data transport from the source machine to the destination machine, independently of the physical network or networks currently in use.
- The services, design, protocols, and performance of the transport layer will be discussed.

# Services Provided to the Upper Layers

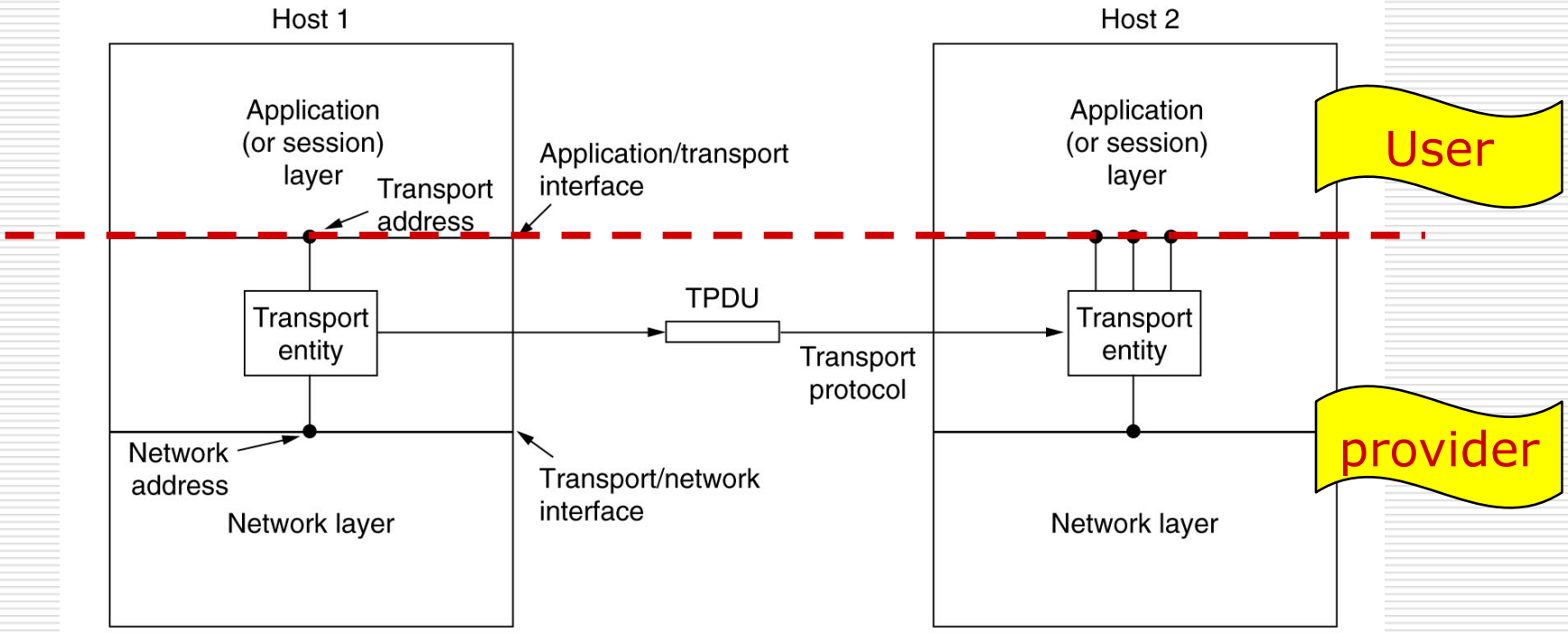
- The ultimate goal of the transport layer is to provide efficient, reliable, and cost-effective service to its users.
- Users are normally processes in the application layer.
- The hardware and/or software that does the work is called the transport entity(传输实体).
- The transport entity can be located in:
  - operating system kernel
  - a separate user process
  - a library package linked into the network applications
  - the network cards.
  - etc.

# Transport Services

- There are two types of transport services:
  - connection-oriented
  - Connectionless
- This is very similar to the network layer service.
- Why are there two distinct layers?
  - The network layer mostly runs on the routers, which are operated by the carrier. So the users have no real control over the network layer.
  - Putting on top of the network layer another layer can make the users have some control over the quality of the service(QoS).
  - The transport service primitives can be designed to be independent of the network service primitives, which may vary considerably from network to network.

# Network, Transport, And Application Layers

- The transport layer provides a seamless interface between the Application layer and the Network layer
- The bottom four layers can be seen as the transport service provider, whereas the upper layers are the transport service user.



# Transport Service Primitives

- The transport service primitives allow application programs to access the transport service.
- Two main differences between the transport service and the network service:
  1. The network service is intended to model the service offered by real (unreliable) networks. The (connection-oriented) transport service is reliable.
  2. The network service is used only by the transport entities. The transport service is used by application programs directly and must be convenient and easy to use.

# Primitives for a Simple Transport Service

| Primitive | Packet sent | Meaning |
|---|---|---|
| LISTEN | (none) | Block until some process tries to connect |
| CONNECT | CONNECTION REQ. | Actively attempt to establish a connection |
| SEND | DATA | Send information |
| RECEIVE | (none) | Block until a DATA packet arrives |
| DISCONNECT | DISCONNECTION REQ. | This side wants to release the connection |

# An Example of Using Transport Service Primitives

- **Connection establishment:**
  - **The server executes a LISTEN primitive, which blocks the server until a client turns up.**
  - **A client executes a CONNECT primitive, which blocks the client, and sends a packet (encapsulating a CONNECTION REQUEST TPDU) to the server via the underlying network layer.**
  - **When the packet arrives at the server side, the transport entity checks to see that whether the server is blocked on a LISTEN (i.e., interested in handling requests). It then unblocks the server and sends a CONNECTION ACCEPTED TPDU back to the client.**
  - **When this TPDU arrives at the client side, the client is unblocked and the connection is established.**

# An example of using transport service primitives (cont'd)

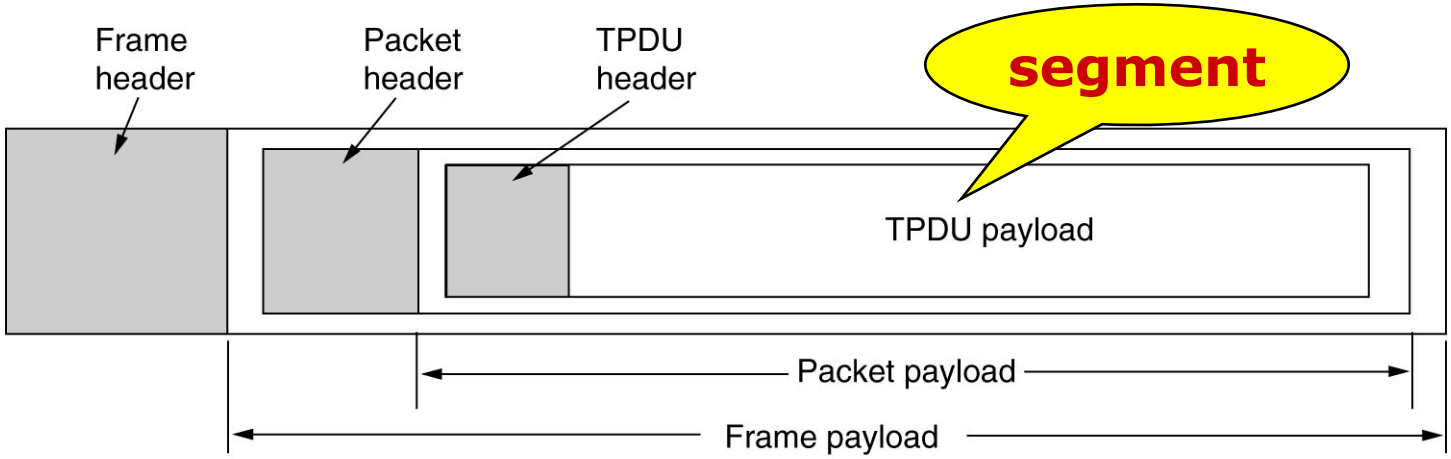
- **Data exchange**:
  - Either party can do a (blocking) RECEIVE to wait for the other party to do a SEND.
  - When the DATA TPDU arrives, the receiver is unblocked.
  - As long as both sides can keep track of whose turn it is to send, this scheme works fine.
  - Every packet sent will be (eventually) acknowledged by the underlying network layer.
  - These acknowledgements, timers, and re-transmissions are managed by the transport entities using the network layer protocol and are not visible to the transport users.

# An example of using transport service primitives (cont'd)

- **Connection release:**
  - **Asymmetric disconnection: either transport user can issue a DISCONNECT primitive, which results in a DISCONNECT TPDU being sent to the remote transport entity. Upon arrival, the connection is released.**
  - **Symmetric disconnection: When one side does a DISCONNECT, that means it has no more data to send, but is still willing to accept data from its partner. A connection is released when both sides have done a DISCONNECT.**

# TPDU（Segment）

- **TPDU (Transport Protocol Data Unit) are messages sent from transport entity to transport entity.**
- **TPDUs are contained in packets (exchanged by network layer).**
- **Packets are contained in frames (exchanged by the data link layer).**



# Transport layer protocol

- UDP
  - User datagram protocol
- TCP
  - Transport control protocol

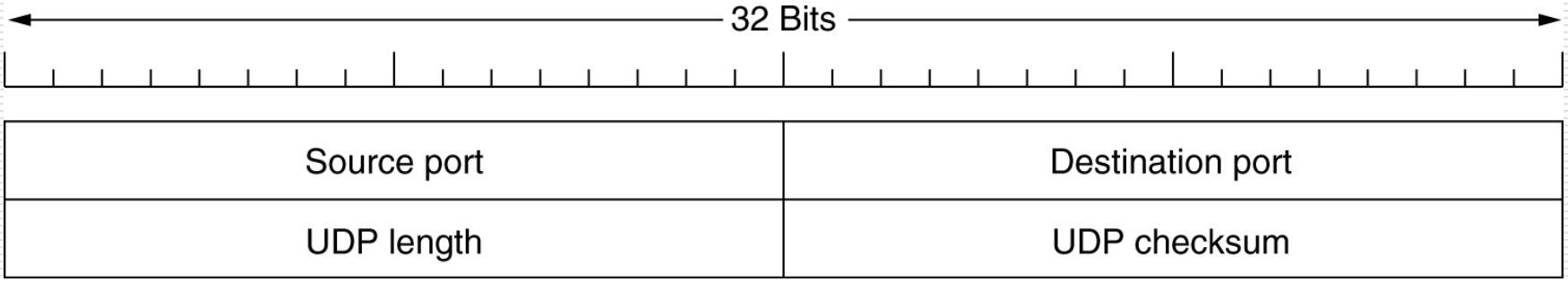
# User Datagram Protocol (UDP)

- UDP is a connectionless transport protocol used in the Internet.
- UDP is used to send encapsulated datagrams without having to establish a connection.
- UDP is described in RFC 768.
- Many client-server applications (e.g. DNS) that have one request and one response use UDP.

Why need UDP?

# UDP Header

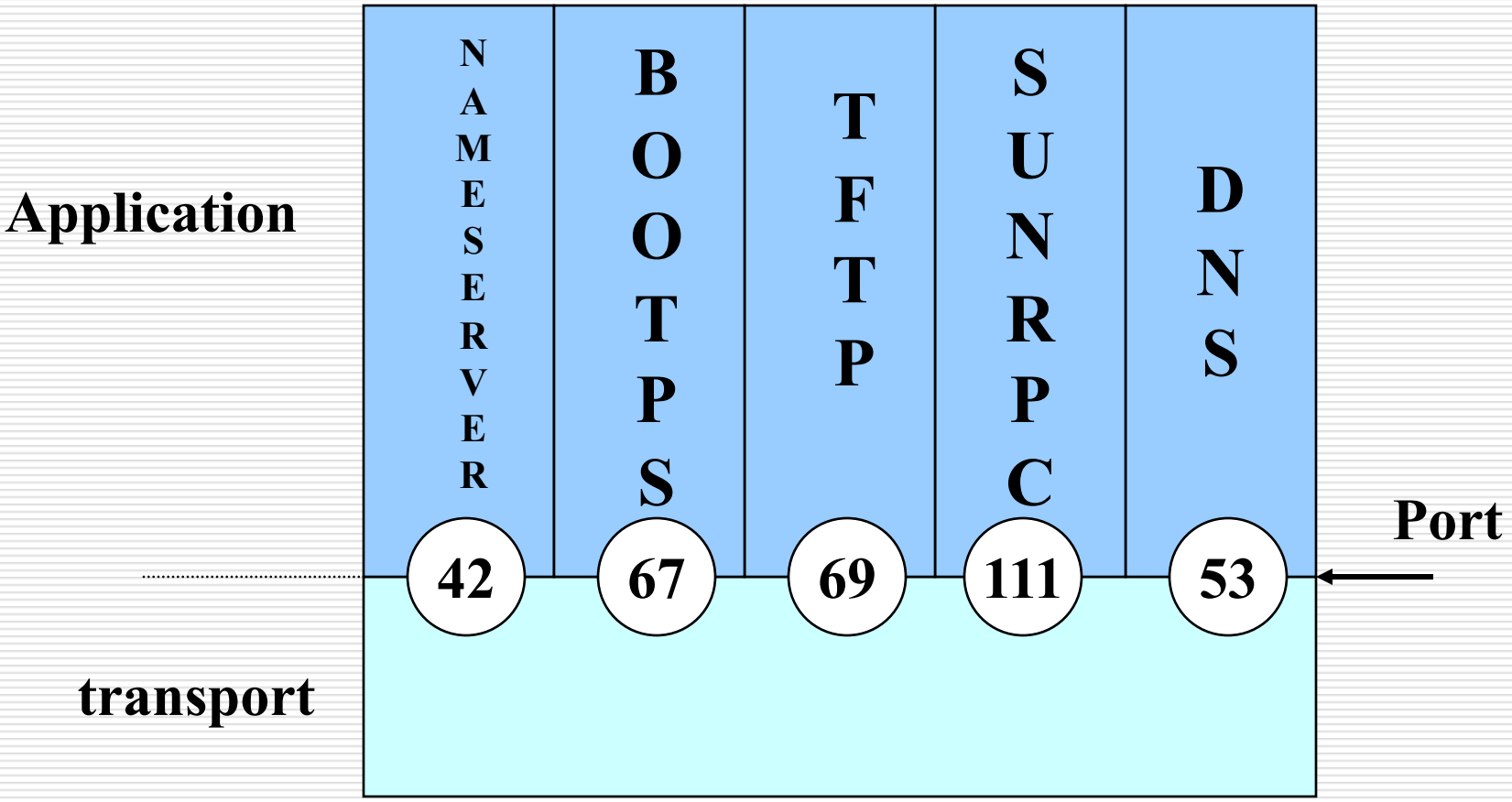
- UDP transmits segments consisting of an 8-byte header followed by the payload.
- The UDP length field includes the 8-byte header and the data.
- The UDP checksum is optional and stored as 0 if not computed (a true computed 0 is stored as all 1s).
- It should be noted that the advantage of UDP over IP packets is the use of the source and destination ports.



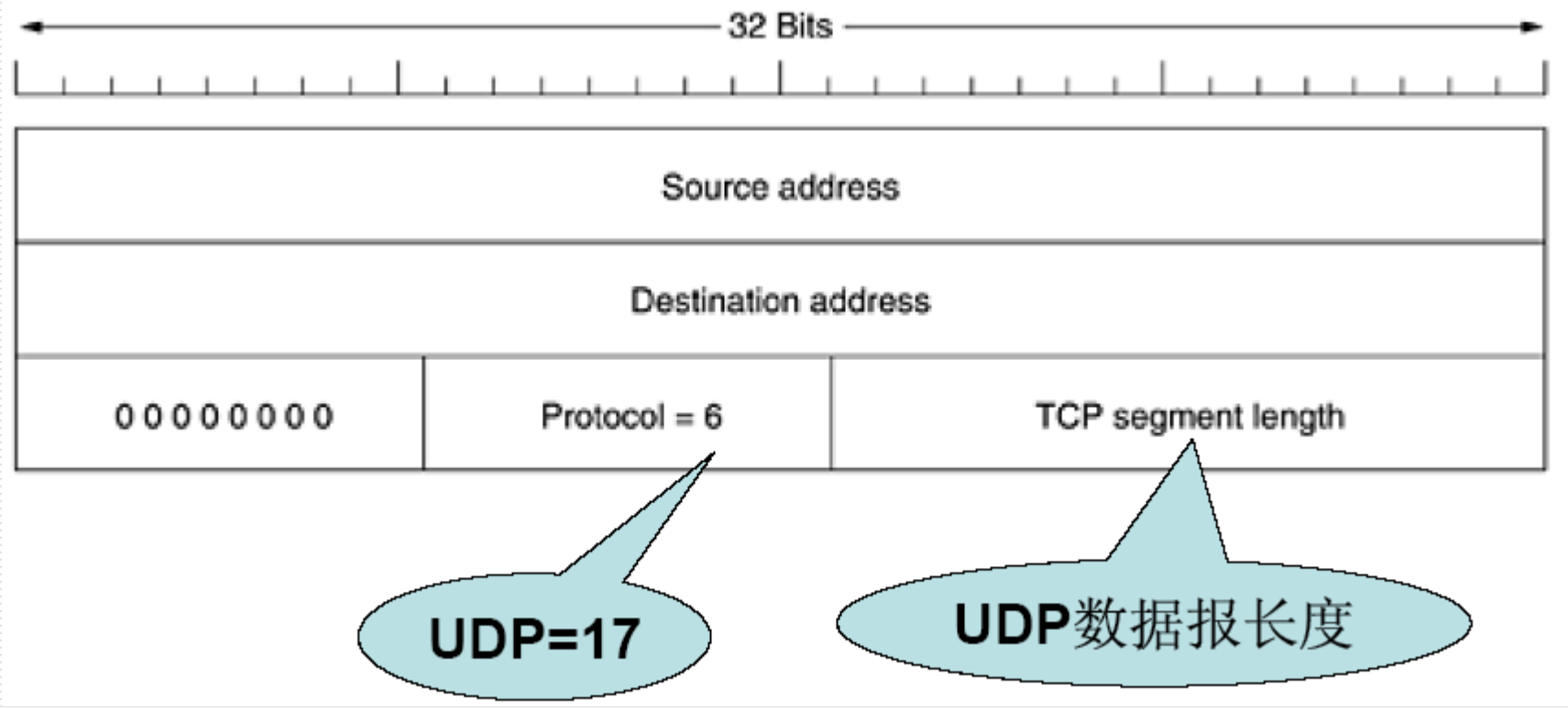
# Port definition

- 16 bit can have result in $2^{16}$ port
  - Port range：0~65535
    - <255 : standards
    - 255~1023 : applications；
    - >1023 : free 。
- Free port (自由端口)
  - Local assignment
  - Random port dynamically

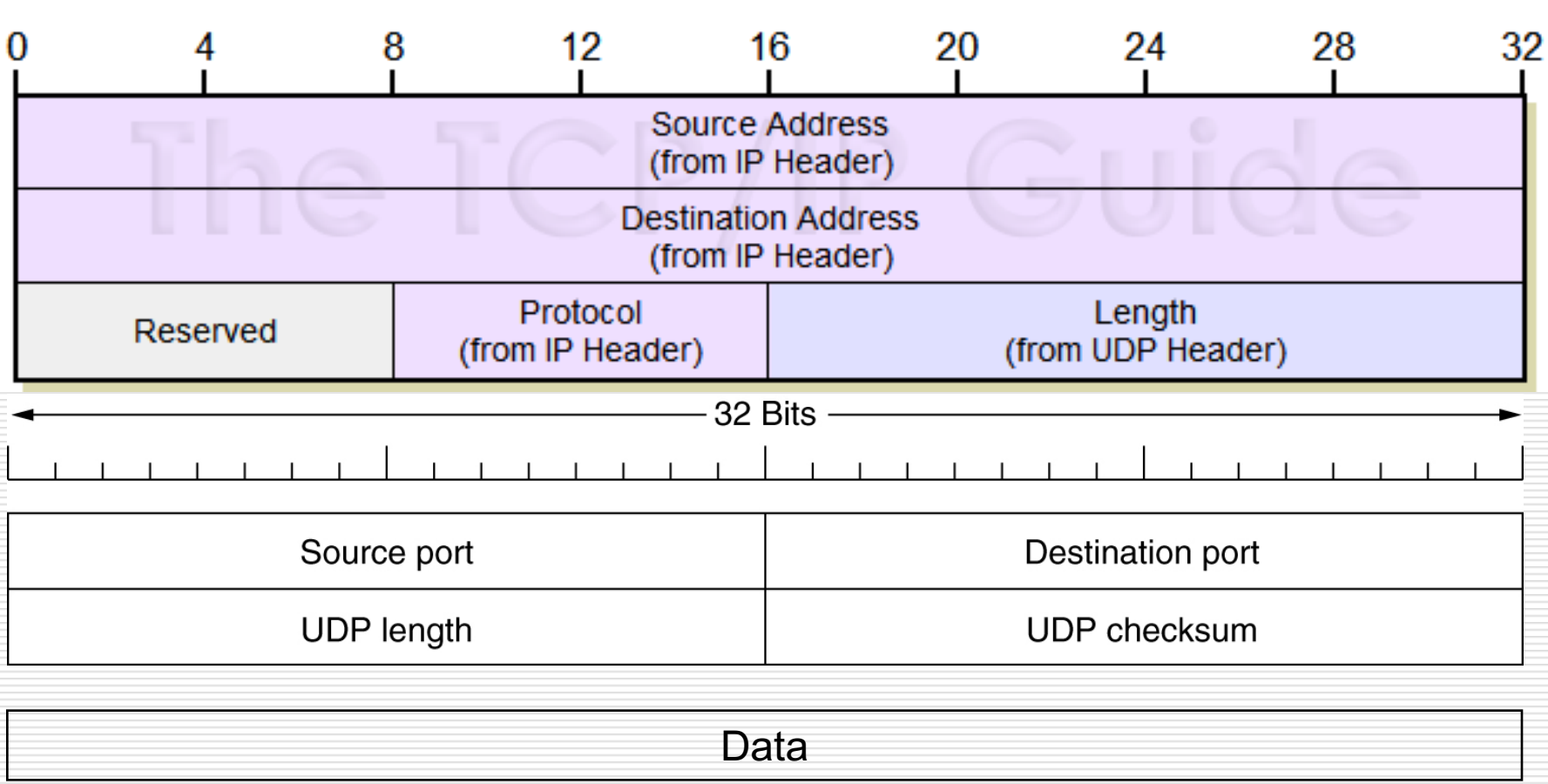
# UDP reserved port



# TCP/UDP pseudo header(伪报头)



# UDP Checksum



# Transmission Control Protocol

- **TCP (Transmission Control Protocol) was specifically designed to provide a reliable end-to-end byte stream over an unreliable internetwork.**
- **TCP must dynamically adapt to different topologies, bandwidths, delays, packet sizes, etc and must be robust to failures.**

# TCP (cont'd)

- **Each machine supporting TCP has a TCP entity, either a user process or part of the OS kernel that manages TCP streams and interfaces to the IP layer**
  - **A TCP entity accepts user data streams from local processes, breaks them up into pieces not exceeding 64K bytes (in practice, usually about 1460 bytes in order to fit in a single Ethernet frame with the IP and TCP headers), and sends each piece as a separate IP datagram.**
  - **When IP datagrams containing TCP data arrive at a machine, they are given to the TCP entity, which reconstructs the original byte streams.**

Encapsulation

De-Encap.

# TCP Service Model

- TCP service is obtained by having both the sender and receiver create end points, called sockets.
- Each socket has a number (address) consisting of the IP address of the host and a 16-bit number local to that host, called port.
- Port numbers below 1024 are called well-known ports and are reserved for standard services.

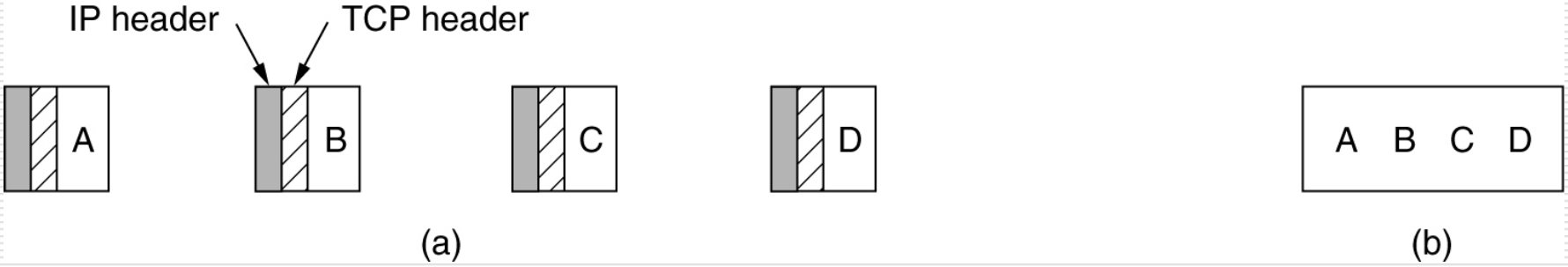
# Some Assigned Ports

- The list of well-known ports is given at http://www.iana.org .
- Over 300 have been assigned.

| Port | Protocol | Use |
|---|---|---|
| 21 | FTP | File transfer |
| 23 | Telnet | Remote login |
| 25 | SMTP | E-mail |
| 69 | TFTP | Trivial file transfer protocol |
| 79 | Finger | Lookup information about a user |
| 80 | HTTP | World Wide Web |
| 110 | POP-3 | Remote e-mail access |
| 119 | NNTP | USENET news |

# TCP Service Model (cont'd)

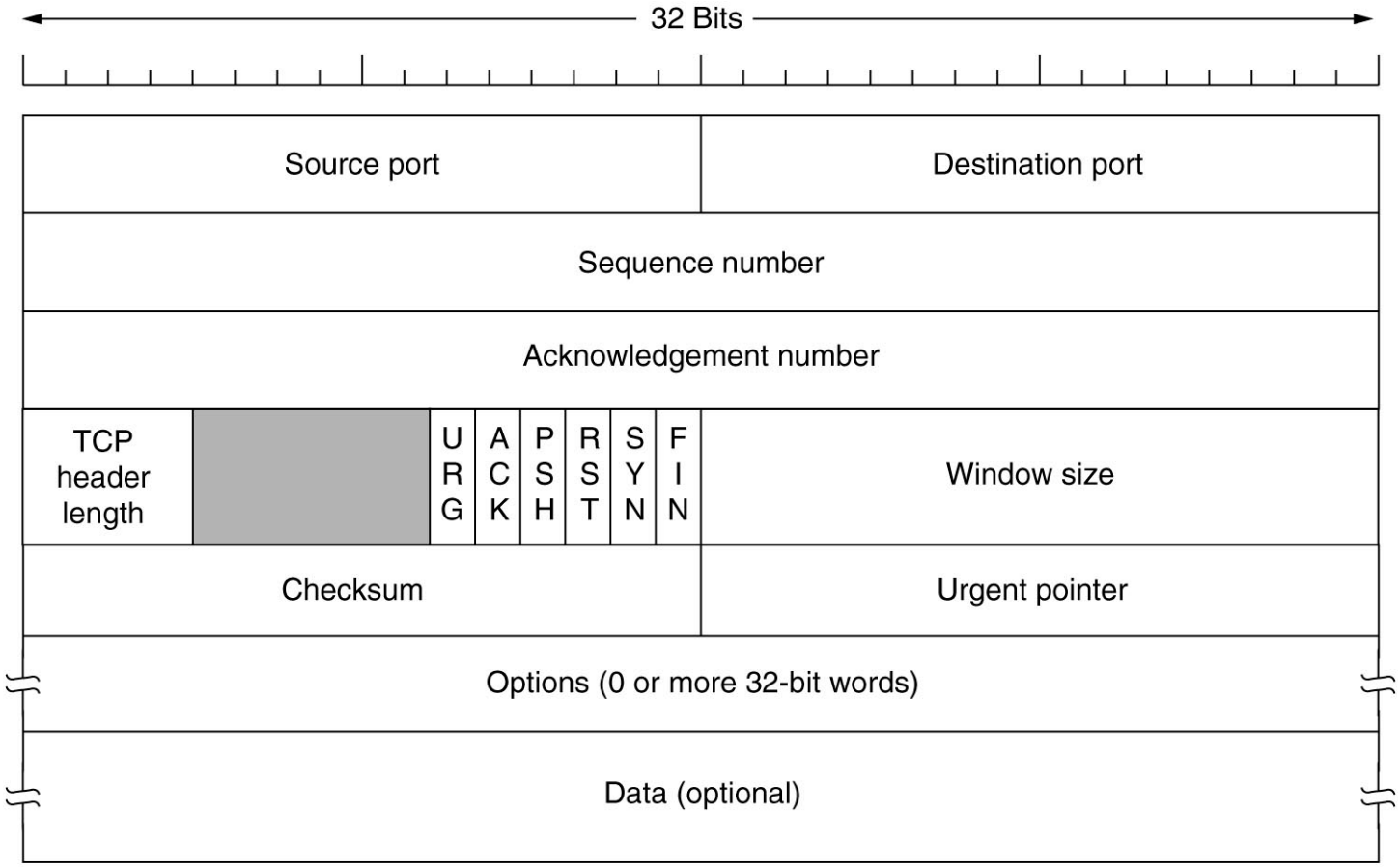
- All TCP connections are full-duplex (which means that traffic can go in both directions at the same time) and point-to-point (which means each connection has exactly two end points).
- TCP does not support multicasting or broadcasting.
- A TCP connection is a byte stream, not a message stream.



# TCP Protocol

- The sending and receiving TCP entities exchange data in the form of segments.
- A segment consists of a fixed 20-byte header (plus an optional part) followed by zero or more data bytes.
- The TCP software decides how big segments should be. Two limits restrict the segment size:
  - Each TCP segment must fit in the 65,535 bytes IP payload.
  - Each TCP segment must fit in the MTU (Maximum Transfer Unit) of the underlying network (e.g., 1500 bytes – the Ethernet payload size).
- The basic protocol used by TCP entities is the sliding window protocol.

# TCP TPDU(segment) format

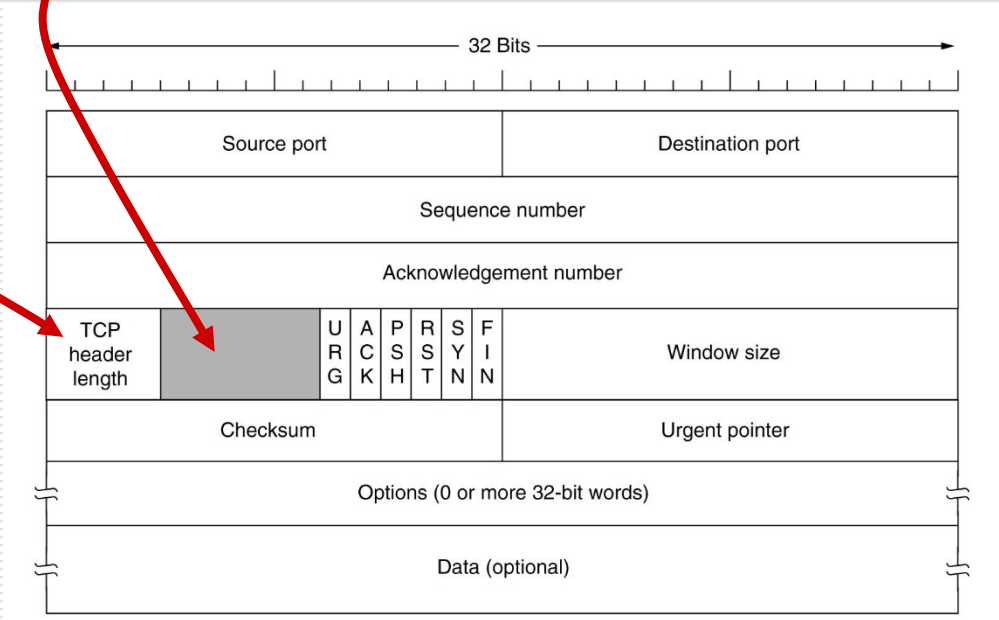


# TCP Header

- **Source port** and **Destination port** fields identify the local end points of the connection. (16 + 16 bits)
- **Sequence number** – byte number (32 bits)
  - ISNs(initial sequence numbers ):random produce
  - SYN: A segment with ISNs and SYN control-bit
- **Acknowledgement number** - the next byte expected. (32 bits)

# TCP Header(cont'd)

- **TCP header length - tells how many 32-bit words are contained in the TCP header. (4 bits)**
- **a 6-bit field that is not used.**



# TCP Header (cont'd)

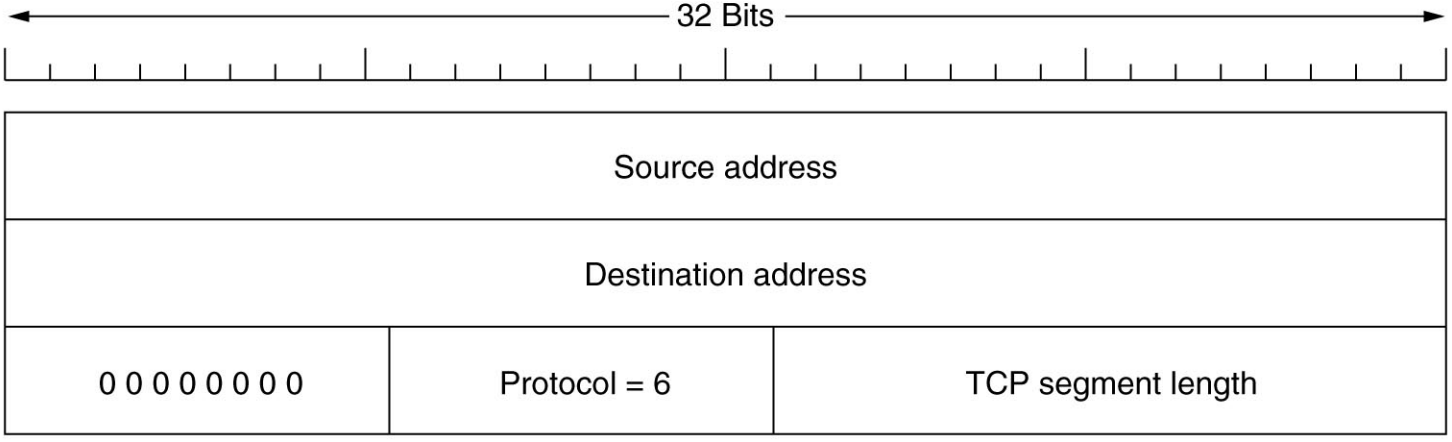
- **URG is set to 1 if urgent pointer is in use, which indicates a byte offset from the current sequence number at which urgent data are to be found.**
- **ACK is set to 1/0**
  - **1 :indicate that the Acknowledgment number is valid**
  - **0: invalid.**

# TCP Header (cont'd)

- **PSH indicates PUSHed data.**
  - **=1:The receiver is requested to deliver the data to the application upon arrival and not buffer it**
- **RST is used to reset a connection that has become confused due to delayed duplicate SYNs or host crashes.**
- **SYN is used with the ACK to distinguish two possibilities**
  - **When SYN=1，ACK=0， CONNECTION REQUEST**
  - **When SYN=1，ACK=1， CONNECTION CONFIRM**
- **FIN is used to release a connection. It specifies that the sender has no more data to transmit, but may continue to receive data indefinitely.**

# TCP Header (cont'd)

- **Flow control in TCP is handled using a variable-sized sliding window.**
- **Window size - tells how many bytes may be sent starting at the byte acknowledged. （up to receiver）**
- **Checksum – provide extra reliability.**
  - **checksums the header, the data, and the conceptual pseudoheader.**



# TCP Header (cont'd)

- The Options field provides a way to add extra facilities not covered by the regular header.
- Example options:
  1. The most important option is the one that allows each host to specify the maximum TCP payload it is willing to accept.
     - Using large segments is more efficient than using small ones
     - During connection setup, each site can announce its maximum TCP payload in the Option field and see its partner's. The smaller of two numbers wins.
     - The default is 536 bytes. All Internet hosts are required to accept TCP segments of 536 + 20 = 556 bytes.

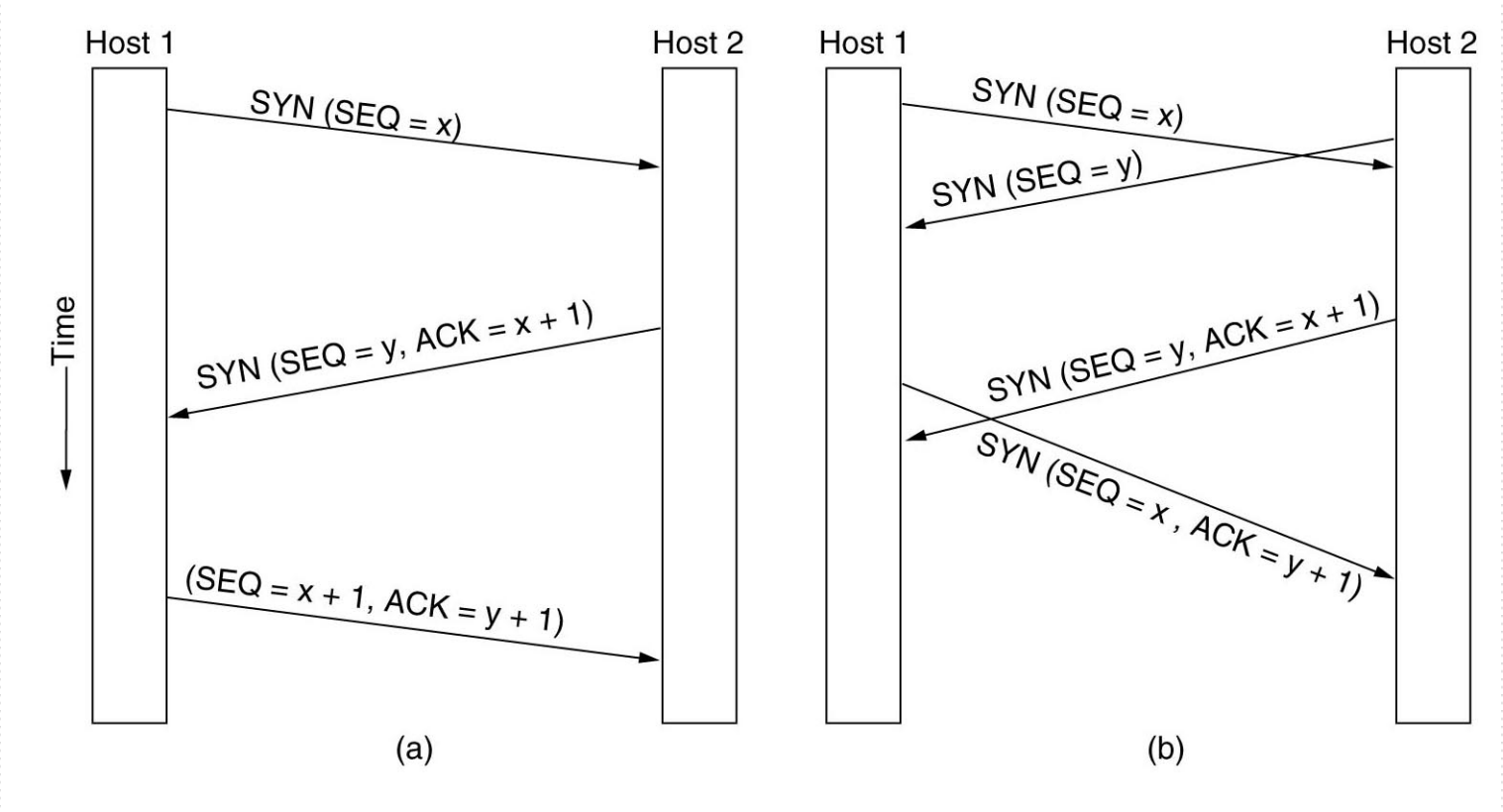
# TCP Header (cont'd)

- **Example options:**
  2. **For lines with high bandwidth and/or high delay, the 64 KB window size is often a problem. A Window scale option allows the sender and receiver to negotiate a window scale factor. This number allows both sides to shift the Window size field up to 14 bits to the left, thus allowing windows of up to $2^{30}$ bytes. Most TCP implementations support this option.**
  3. **Another option proposed by RFC 1106 and now widely implemented is the use of the selective repeat instead of go back n protocol.**

# TCP Connection Establishment

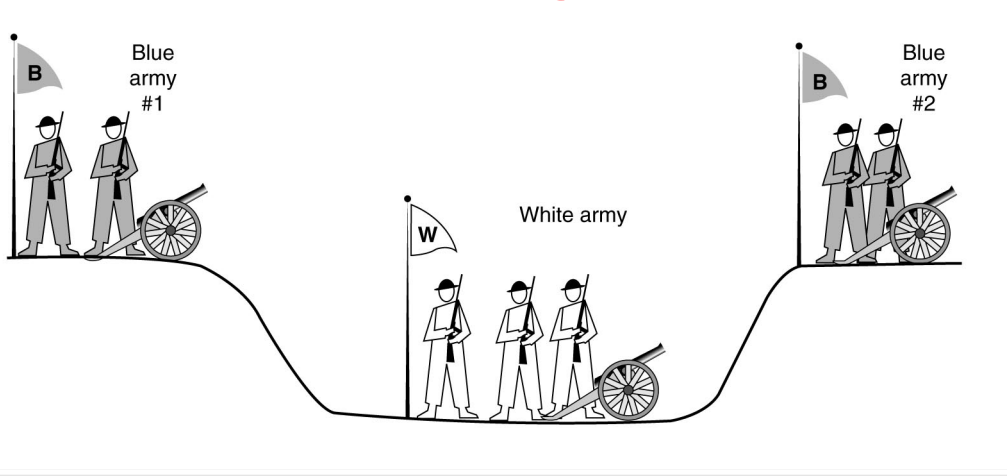
- The three-way handshake is used to establish connections.
  - One side (the server) passively listens for incoming connections
  - The other side (the client) connects to the server with a connection request and sets out the parameters.
  - The server replies with an acknowledgement and the connection is set.

# TCP Connection Establishment (cont'd)



# Two-army problem

- Symmetric release –treats the connection like two unidirectional connections, requiring each end of the connection to be released.
- Unfortunately, determining when the two sides are done is difficult.
- This problem is known as the two-army problem.
  - It can be proven that no protocol exists for the problem.
  - The sender of the final message can never be sure of its arrival



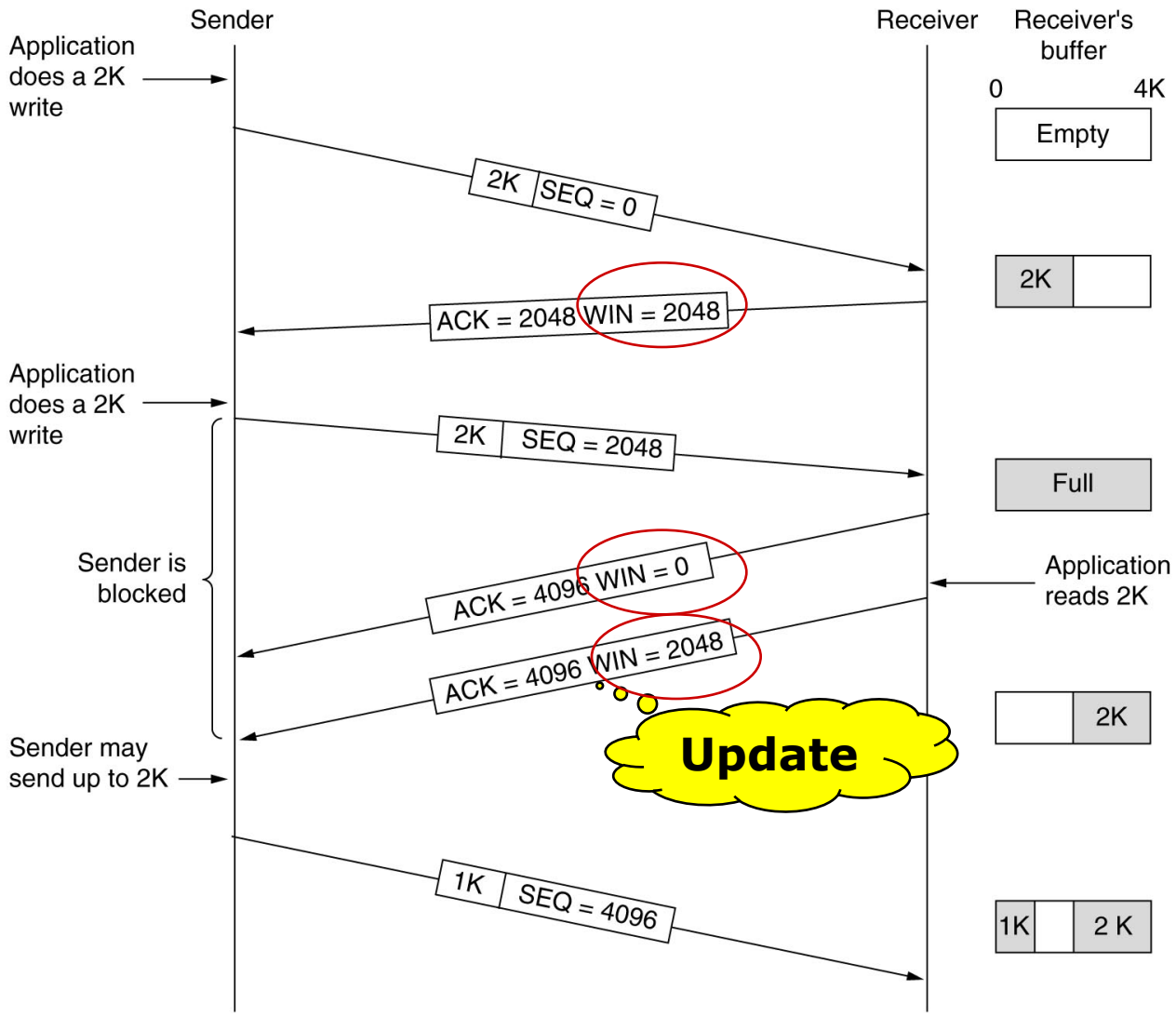
# TCP Connection Release

- To release a connection:
  - Either party can send a TCP segment with the FIN bit set, which means that it has no more data to transmit.
  - When the FIN is acknowledged, that direction is shut down.
  - When both directions have been shut down, the connection is released.
- To avoid the two-army problem, timers are used:
  - If a response to a FIN is not forthcoming within two maximum packet lifetimes, the sender of the FIN releases the connection.
  - The other side will eventually notice that nobody seems to be listening to it any more, and time out as well.

# Connection Release (cont'd)

- In theory, this protocol can fail if the initial DR and retransmissions are all lost.
  - The sender will give up and delete the connection, while the other side knows nothing at all about the attempts to disconnect and is still fully active.
  - This situation results in a half- open connection.
- One way to kill off half-open connections:P505
  - If no TPDUs have arrived for a certain number of seconds, the connection is automatically disconnected.
  - It is necessary for each transport entity to have a timer that is stopped and then restarted whenever a TPDU is sent. If this timer expires, a dummy TPDU is transmitted P506

# TCP Transmission Policy




**Window Size is decided by receiver!**

# TCP Transmission Policy (cont'd)

- When the window is 0, the sender may not normally send segments, with two exceptions:
  - Urgent data may be sent, for example, to allow the user to kill the process running on the remote machine.
  - The sender may send a 1-byte segment to make the receiver re-announce the next byte expected and window size (in order to prevent deadlock if a window announcement ever gets lost).
- Senders are not required to transmit data as soon as they come in from the application.
- Receivers are also not required to send acknowledgments as soon as possible.

# TCP Transmission Policy (cont'd)

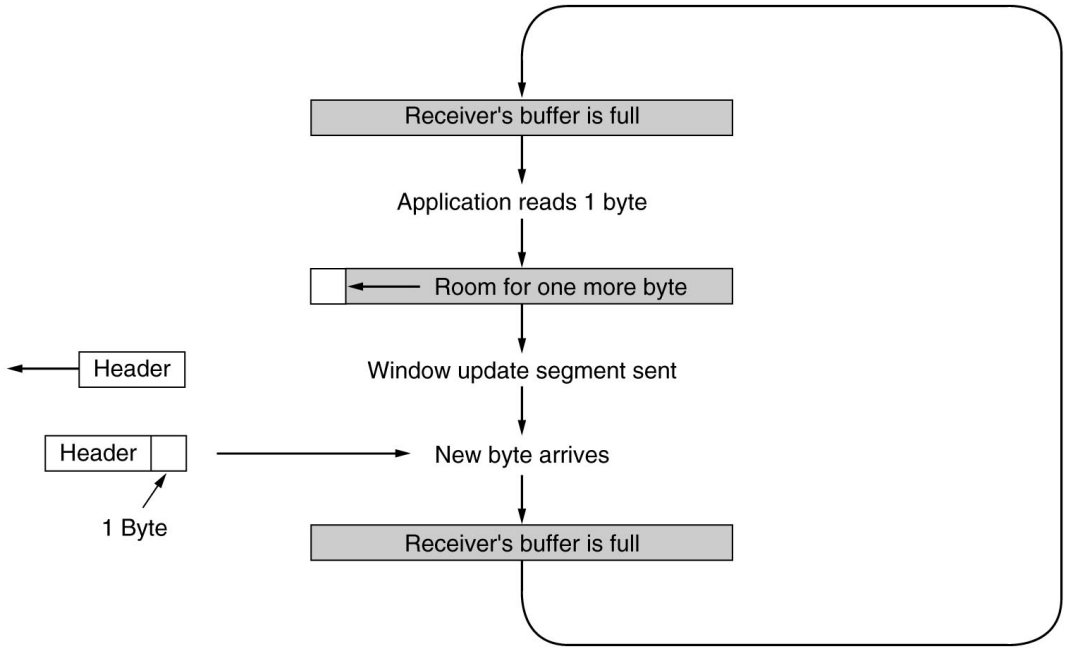
- Consider a TELNET connection to an interactive editor (running on a remote machine but displaying on a local screen) that reacts on every keystroke. In the worst case:
  - When a character is typed, it is given to the sending TCP entity, which creates a 21-byte TCP segment, which the TCP gives to IP to send as a 41-byte IP datagram.
  - At the receiving side (where the editor is running), TCP immediately sends a 40-byte acknowledgment (20 bytes of TCP header and 20 bytes of IP header).
  - Later, when the editor has read the byte, TCP sends a window update, moving the window 1 byte to the right. This packet is also 40 bytes.
  - Finally, when the editor has processed the character, it echoes it as a 41-byte packet (so that the character can be displayed on the local screen).
- In total, at least 162 bytes of bandwidth (without counting the data link layer overhead) are used and four segments are sent for each character typed.

# TCP Transmission Policy

- How to optimize the receiver side ?
  - The receiver can delay acknowledgments and window updates for 500 msec in the hope of acquiring some data on which to hitch a free ride.
- How to optimize the sender side ?
  - Nagle's algorithm (1984):
    - When data come into the sender one byte at a time, just send the first byte and buffer all the rest until the outstanding byte is acknowledged.
    - Then send all the buffered characters in one TCP segment and start buffering again until they are all acknowledged.
    - A scenario in which it is better to disable the Nagle's algorithm: When an X-Windows application is being run over the Internet, mouse movements have to sent to the remote computer. Gathering them up to send in bursts makes the mouse cursor move erratically.

# Silly Window Syndrome

- The **silly window syndrome problem** occurs when data are passed to the sending TCP entity large blocks, but an interactive application on the receiving side reads data 1 byte at a time, as illustrated in the figure below.



# Silly Window Syndrome (cont'd)

- **Clark's solution is to prevent the receiver from sending a window update for 1 byte. Instead it is forced to wait until it has a decent amount of space available.**
- **Furthermore, the sender can also help by not sending tiny segment. Instead, it should try to wait until it has accumulated enough space in the window to send a full segment or at least one containing half of the receiver's buffer size.**
- **The receiver can maintain an internal buffer and block a READ request from the application until it has a large chunk of data to provide.**

# summary

- UDP (segment)
- TCP (segment)

Comparison

- Measures to provide reliable data transmit (transmission policy)
  - Positive acknowledgement and retransmit
  - Window (sliding widow)
  - nagle and clark

Thanks!

Comparisons between UDP & TCP