



# EEC-484/584 Computer Networks

---

## Lecture 16

Wenbing Zhao

wenbing@ieee.org

(Lecture notes are based on materials supplied by  
Dr. Louise Moser at UCSB and Prentice-Hall)



## Outline

---

- Review
  - Services provided by transport layer
  - Connection management
- Today's topics
  - Connection management (cont'd)
  - Flow control and buffering
  - UDP
  - TCP part 1
- Section 6.3 "A simple transport protocol" not required for exam



## Transport Layer - Design Goals

- To provide efficient, reliable, cost-effective service to the transport layer users
- To allow application programs to be written using a standard set of primitives and to have programs work on a variety of networks
- To enhance the quality of service provided by the network layer below it



## Services Provided to the Upper Layers

- **Transport Entity** - The hardware and/or software within the transport layer that does the work
- The transport layer fulfills the key function of isolating the upper layers from the technology, design, and imperfections of the subnet
  - **Transport service provider** - The bottom four layers
  - **Transport service user** - the upper layer(s)

## Berkeley Sockets

- Socket – an endpoint to which connections can be attached from bottom (OS) and to which processes can be established from top
- Socket primitives for TCP

Primitive	Meaning
SOCKET	Create a new communication end point
BIND	Attach a local address to a socket
LISTEN	Announce willingness to accept connections; give queue size
ACCEPT	Block the caller until a connection attempt arrives
CONNECT	Actively attempt to establish a connection
SEND	Send some data over the connection
RECEIVE	Receive some data from the connection
CLOSE	Release the connection

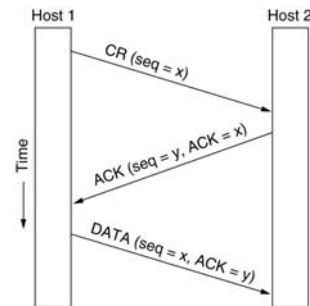
## Connection Establishment

- Problem of Delayed Duplicates
- Solution
  - Sequence number based approach
  - Ensure each packet has finite life using hop count or time stamp

## Connection Establishment

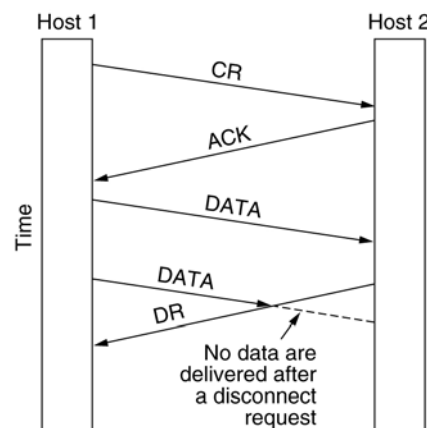
- **Three-way handshake** - to establish a connection, need to agree on initial sequence numbers (instead of using clock). Again, need to prevent delayed duplicates

- Host 1 chooses sequence number  $x$ , sends it to host 2 in a control TPDU
- Host 2 replies with a control TPDU, acks  $x$ , announces its own sequence number  $y$
- Host 1 sends its first data TPDU, using  $x$ , acks  $y$



## Connection Release

- **Asymmetric release** is abrupt and may result in data loss

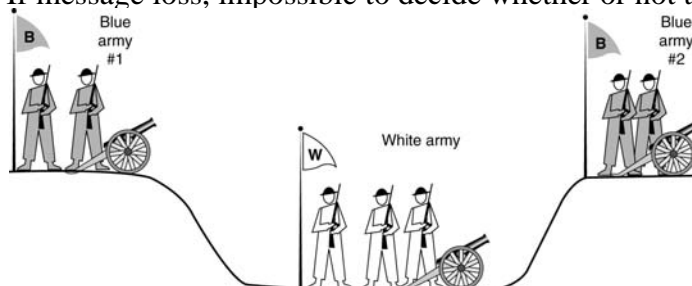


## Connection Release

- **Symmetric release** - to avoid data loss, each side must wait to disconnect until it is sure that other side is prepared to disconnect
  - It works when each process has a fixed amount of data to send and clearly knows when it has sent it
  - Cannot guarantee no message loss otherwise - **two army problem**

## Connection Release

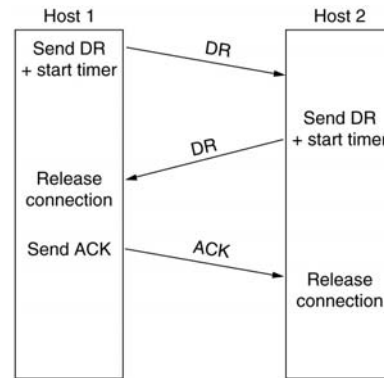
- **Two army problem** - the only way to win is if both blue armies attack. If only one attacks, it will be destroyed
  - Blue armies communicate by messengers, but messengers can be caught, and their messages can be lost
  - If no message loss, wait for message from each army, attack if none objects
  - If message loss, impossible to decide whether or not to attack



## Connection Release

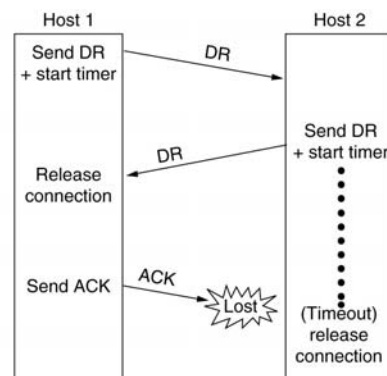
- In practice, three-way handshake works most of the time

- Host 1 sends DR (Disconnect Request), starts timer in case DR lost
- Host 2 sends DC (Disconnect Confirm), starts timer in case DC lost
- When DC arrives, host 1 sends ACK, deletes connection
- When ACK arrives, host 2 deletes connection



## Connection Release

- If host 1's ACK is lost
  - Host 2's timer saves the day



- 
- ```

sequenceDiagram
    participant H1 as Host 1
    participant H2 as Host 2
    H1->>H2: DR
    Note over H2: Send DR & start timer
    H2->>H1: DR
    Note over H2: Send DR & start timer
    Note over H1: (Timeout)
    H1->>H2: DR
    Note over H2: Send DR & start timer
    H2->>H1: DR
    Note over H1: Release connection
    H1->>H2: ACK
    Note over H2: Release connection
  
```
- The diagram illustrates a timeout scenario in a Stop-and-Wait protocol. Host 1 sends a Data Report (DR) to Host 2. Host 2 receives it and starts a timer. Host 2 then sends a DR back to Host 1. Host 1 receives it but, due to a timeout, sends another DR to Host 2. Host 2 receives this second DR and starts its timer again. Host 2 then sends a DR back to Host 1. Host 1 receives it and releases the connection. Finally, Host 1 sends an ACK to Host 2, which also releases the connection.

- [illegible]



## Connection Release

- Three-way handshake can fail if initial DR and  $n$  retransmissions are all lost
  - Host 1 gives up, deletes connection
  - Host 2 knows nothing about attempts to disconnect, remains fully active
  - **Result:** half-open connection
  - To kill off half-open connection, need timer
  - Rule: if no TPDUs arrive in some number of seconds, disconnect connection



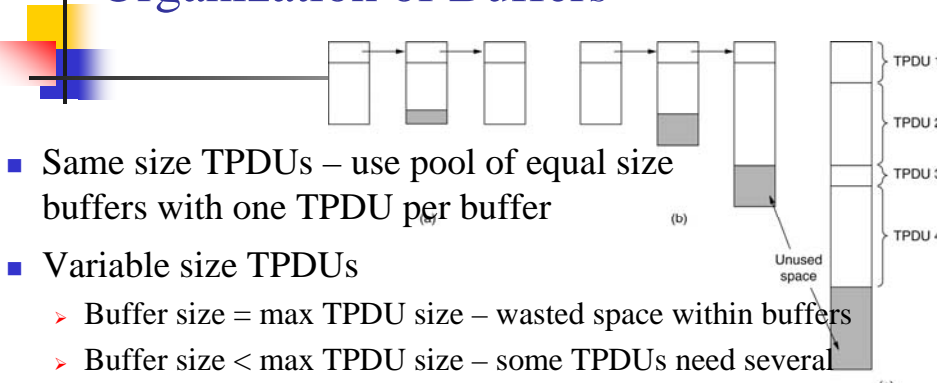
## Flow Control and Buffering

- Like DLL, sliding window to prevent fast transmitter overwhelming slow receiver
- With unreliable network service, sender must buffer all TPDUs sent until they are acknowledged
- With reliable network service
  - If receiver has plenty of buffer space, sender does not need to buffer TPDUs sent
  - If receiver does not have enough buffer space, sender must buffer all TPDUs sent because ACK from the network layer only means the network layer at the other end received them



## Organization of Buffers

17

- 
- Same size TPDUs – use pool of equal size buffers with one TPDU per buffer
  - Variable size TPDUs
    - Buffer size = max TPDU size – wasted space within buffers
    - Buffer size < max TPDU size – some TPDUs need several buffers
    - Variable size buffers – better buffer utilization, but more complex management
    - One large circular buffer per connection – better buffer utilization

6 November 2005

EEC484/584

Wenbing Zhao

## Flow Control and Buffering

18

- Source buffering vs. destination buffering
  - Low bandwidth bursty traffic
    - Acquire buffers dynamically
    - Sender must keep TPDU until its is acked
    - Better to buffer at sender
  - High bandwidth traffic
    - Better to buffer at receiver
    - Receiver should allocate full window of buffers to allow data to be sent at max speed

6 November 2005

EEC484/584

Wenbing Zhao

## Dynamic Buffer Allocation

- Initially, the sender requests certain number of buffers. The receiver then grants as many of these as it can afford
- Every time the sender transmits a TPDU, it must decrement its allocation, stopping altogether when the allocation reaches zero
- The receiver then separately piggybacks both acknowledgements and buffer allocations onto the reverse traffic

6 November 2005

EEC484/584

Wenbing Zhao

## Flow Control and Buffering

- Dynamic buffer allocation. The arrows show the direction of transmission. An ellipsis (...) indicates a lost TPDU

| A      | Message              | B   | Comments                                     |
|--------|----------------------|-----|----------------------------------------------|
| 1 →    | < request 8 buffers> | →   | A wants 8 buffers                            |
| 2 ←    | <ack = 15, buf = 4>  | ←   | B grants messages 0-3 only                   |
| 3 →    | <seq = 0, data = m0> | →   | A has 3 buffers left now                     |
| 4 →    | <seq = 1, data = m1> | →   | A has 2 buffers left now                     |
| 5 →    | <seq = 2, data = m2> | ... | Message lost but A thinks it has 1 left      |
| 6 ←    | <ack = 1, buf = 3>   | ←   | B acknowledges 0 and 1, permits 2-4          |
| 7 →    | <seq = 3, data = m3> | →   | A has 1 buffer left                          |
| 8 →    | <seq = 4, data = m4> | →   | A has 0 buffers left, and must stop          |
| 9 →    | <seq = 2, data = m2> | →   | A times out and retransmits                  |
| 10 ←   | <ack = 4, buf = 0>   | ←   | Everything acknowledged, but A still blocked |
| 11 ←   | <ack = 4, buf = 1>   | ←   | A may now send 5                             |
| 12 ←   | <ack = 4, buf = 2>   | ←   | B found a new buffer somewhere               |
| 13 →   | <seq = 5, data = m5> | →   | A has 1 buffer left                          |
| 14 →   | <seq = 6, data = m6> | →   | A is now blocked again                       |
| 15 ←   | <ack = 6, buf = 0>   | ←   | A is still blocked                           |
| 16 ... | <ack = 6, buf = 4>   | ←   | Potential deadlock                           |

## Crash Recovery

- Router crash / recovery - easy to handle
- How to cope with server host crash and recovery
  - Problem is caused by the non-atomicity of sending ack and delivery of TPDU
  - Problem like this in layer N can only be solved by N+1

## Crash Recovery

- Different combinations of client and server strategy

| Strategy used by sending host | Strategy used by receiving host |     |       |                       |      |       |
|-------------------------------|---------------------------------|-----|-------|-----------------------|------|-------|
|                               | First ACK, then write           |     |       | First write, then ACK |      |       |
|                               | AC(W)                           | AWC | C(AW) | C(WA)                 | W AC | WC(A) |
| Always retransmit             | OK                              | DUP | OK    | OK                    | DUP  | DUP   |
| Never retransmit              | LOST                            | OK  | LOST  | LOST                  | OK   | OK    |
| Retransmit in S0              | OK                              | DUP | LOST  | LOST                  | DUP  | OK    |
| Retransmit in S1              | LOST                            | OK  | OK    | OK                    | OK   | DUP   |

OK = Protocol functions correctly  
 DUP = Protocol generates a duplicate message  
 LOST = Protocol loses a message



## The Internet Transport Protocols: UDP

---

- Introduction to UDP
- Remote Procedure Call
- The Real-Time Transport Protocol

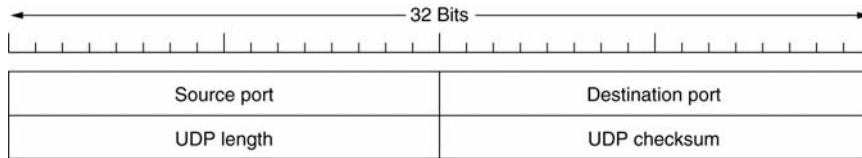


## Introduction to UDP

---

- UDP (User Datagram Protocol) - provides a way for applications to send encapsulated IP datagrams and send them without having to establish a connection
- It does not do flow control, error control, or retransmission upon receipt of a bad segment
- UDP transmits **segments** consisting of an 8-byte header followed by the payload

## UDP Header



- The **source port** and **destination port** serve to identify the end points within the source and destination machines.
  - When a UDP packet arrives, its payload is handed to the process attached to the destination port.
  - This attachment occurs when BIND primitive is used
- 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)

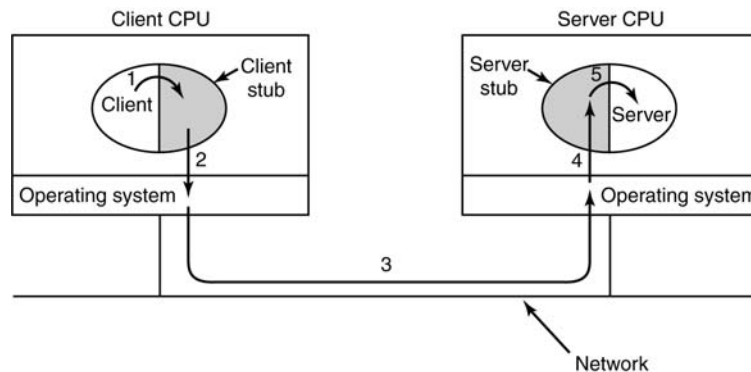
26

## Remote Procedure Call

- RPC - to make a remote procedure call look as much as possible like a local one.
  - In the simplest form, to call a remote procedure, the client program must be bound with a small library procedure, called the **client stub**, that represents the server procedure in the client's address space.
  - Similarly, the server is bound with a procedure called the **server stub**. These procedures hide the fact that the procedure call from the client to the server is not local.
- RPC and UDP are a good fit when the parameters or results are smaller than the maximum UDP packet and when the operation requested is idempotent

## Remote Procedure Call

- Steps in making a remote procedure call
  - The stubs are shaded



6 November 2005

EEC484/584

Wenbing Zhao

## The Real-Time Transport Protocol

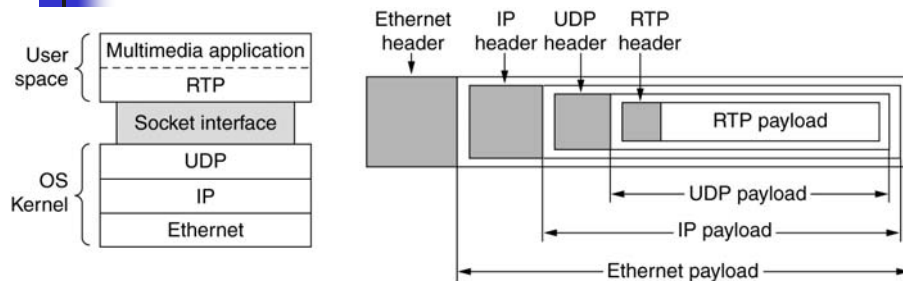
- RTP is used in many multimedia applications
  - E.g., Internet radio, Internet telephony, music-on-demand, videoconferencing, video-on-demand
- RTP runs in user space and normally over UDP
  - It is a transport protocol that is implemented in the application layer
- Basic function of RTP
  - Multiplex several real-time data streams onto a single stream of UDP packets.
  - The UDP stream can be sent to a single destination (unicasting) or to multiple destinations (multicasting)

6 November 2005

EEC484/584

Wenbing Zhao

## The Real-Time Transport Protocol



The position of RTP  
in the protocol stack

Packet nesting

## The Real-Time Transport Protocol

- Each packet sent in an RTP stream is given a number one higher than its predecessor
  - This numbering allows the destination to determine if any packets are missing.
  - If a packet is missing, the best action for the destination to take is to approximate the missing value by interpolation
- Each RTP payload may contain multiple samples, and they may be coded any way the application wants

## The Real-Time Transport Protocol

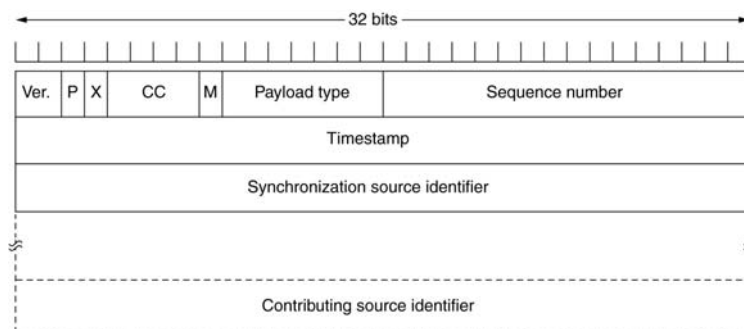
- Timestamping - allow the source to associate a timestamp with the first sample in each packet
  - The timestamps are relative to the start of the stream
  - Allows the destination to do a small amount of buffering and play each sample the right number of milliseconds after the start of the stream (reduce the effects of jitter)
  - Allows multiple streams to be synchronized with each other

6 November 2005

EEC484/584

Wenbing Zhao

## RTP Header



- The *P* bit indicates that the packet has been padded to a multiple of 4 bytes. The last padding byte tells how many bytes were added
- The *X* bit indicates that an extension header is present
  - The format and meaning of the extension header are not defined. This is an escape hatch for any unforeseen requirements
- The *CC* field tells how many contributing sources are present, from 0 to 15



## RTP Header

- The *M* bit is an application-specific marker bit
  - It can be used to mark the start of a video frame, the start of a word in an audio channel, etc.
- The *Payload type* field tells which encoding algorithm used
  - E.g., uncompressed 8-bit audio, MP3, etc.
- The *Sequence number* is a counter that is incremented on each RTP packet sent
- The timestamp is produced by the stream's source to note when the first sample in the packet was made
- The *Synchronization source identifier* tells which stream the packet belongs to
  - It is the method used to multiplex and demultiplex multiple data streams onto a single stream of UDP packets.
- The *Contributing source identifiers*, if any, are used when mixers are present in the studio

34



## Realtime Transport Control Protocol

- RTCP is a sister protocol of RTP
- RTCP handles feedback, synchronization, and the user interface but does not transport any data
  - **Feedback** - increase data rate when network functions well, decrease data rate when network is congested
  - **Synchronization** - handles inter-stream synchronization. Different streams may use different clocks



## The Internet Transport Protocols: TCP

- Introduction to TCP
- The TCP Service Model
- The TCP Protocol
- The TCP Segment Header
- TCP Connection Establishment
- TCP Connection Release
- TCP Connection Management Modeling



## Introduction to TCP

- TCP – Transmission Control Protocol
  - Provide a **reliable end-to-end byte stream** over an unreliable internetwork
  - Each machine supporting TCP has a TCP transport entity 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 and sends each piece as a separate IP datagram
  - When datagrams containing TCP data arrive at a machine, they are given to the TCP entity, which reconstructs the original byte streams
- Best reference book on TCP/IP
  - TCP/IP Illustrated, Volume 1: The Protocols (By W. Richard Stevens)
  - <http://proquest.safaribooksonline.com/0201633469>

## The TCP Service Model

- Requires both sender and receiver to create socket
- Each socket has socket number (address) consisting of IP address of host and 16-bit port number
- Port is TCP name for TSAP
- Connections are identified by the socket identifiers at both ends, i.e., (socket1, socket2)
- Port numbers below 1024 are called **well-known ports** and are reserved for standard services
  - E.g., 21 for FTP, 23 for Telnet
- Full-duplex, point-to-point

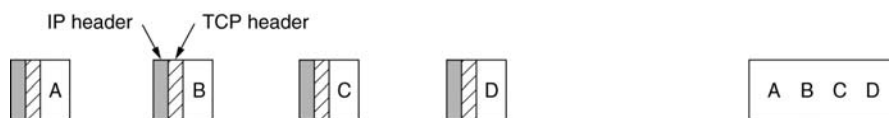
## The TCP Service Model

### Some assigned ports

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

## The TCP Service Model

- TCP connection is byte stream, not message stream, no message boundaries
- TCP may send immediately or buffer before sending
  - PUSH - do not accumulate, send data immediately
  - URGENT - stop accumulating and send immediately



Four 512-byte segments sent  
as separate IP datagrams

The 2048 bytes of data  
delivered to the application  
in a single READ CALL

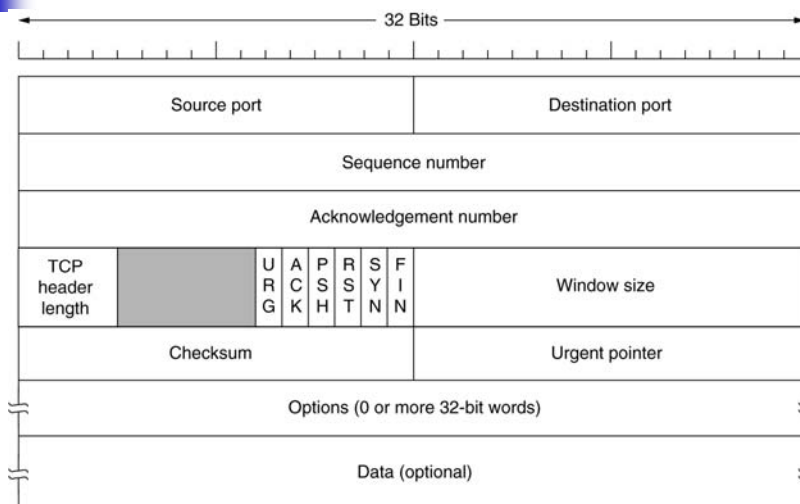
## TCP Protocol

- Sequence numbers used for ACKs and also for window mechanism
- Sender and receiver TCP entities exchange data in the form of segments
- **Segment** - fixed 20 byte header + optional part followed by data
- TCP software decides size of segments, accumulate data from several writes into 1 segment, or split data from one write over multiple segments

## TCP Protocol

- Uses sliding window for flow control
  - Resembles go-back-n protocol
  - No selective ack, or nack, e.g., if 1-1024 and 2049-3072 are received, can only ack 1025
  - RFC 1106 propose a solution, using TCP options

## The TCP Segment Header



## The TCP Segment Header

- Source port and destination port: identify local end points of the connection
  - Source and destination end points together identify the connection
- Sequence number: identify the byte in the stream of data from sending TCP to receiving TCP that the *first byte* of data in this segment represents
- Acknowledgement number: the next sequence number that the sender of the ack expects to receive
  - Last received seq num + 1
- TCP header length - number of 32-bit words in header

## The TCP Segment Header

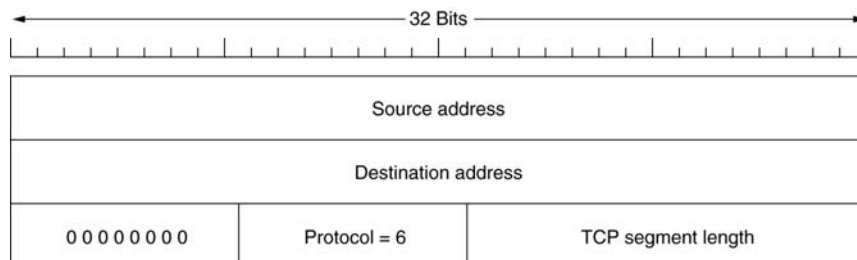
- URG - indicates urgent pointer field is set
- Urgent pointer used to indicate byte offset from current sequence number at which urgent data can be found
- ACK - acknowledgement number is valid
- SYN - used to establish a connection
  - Connection request: ACK = 0, SYN = 1
  - Connection confirm: ACK=1, SYN = 1
- FIN - release a connection, sender has no more data
- RST - reset a connection that is confused (due to delayed duplicate SYNs or host crashes)
- PSH - receiver asked to deliver data upon arrival, not wait until buffer full

## The TCP Segment Header

- Window size - variable-sized sliding window used for flow control, number of bytes that may be sent beyond byte acked
- Checksum - add the header, the data, and the conceptual pseudoheader as 16-bit words, take 1's complement of sum
- Options - provides a way to add extra facilities not covered by the regular header
  - E.g., communicate buffer sizes during set up

## The TCP Segment Header

The pseudoheader included in the TCP checksum





## TCP Connection Establishment

- Uses 3-way handshake, to establish connection
  - Client executes CONNECT, specifying IP address and port to which it wants to connect, max TCP segment size it is willing to accept, and optional user data (e.g., password)
  - The CONNECT primitive sends a TCP segment with *SYN* bit on, *ACK* bit off and the client's initial sequence number (ISN) and waits for a response
  - TCP entity at server check if process is listening on port given in TCP segment header field (i.e., if there is a process that has done a LISTEN on the port)

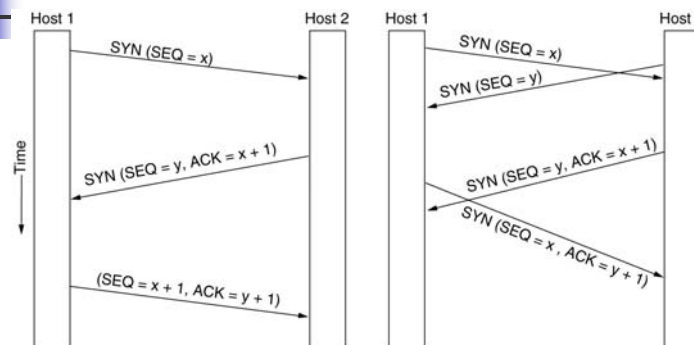


## TCP Connection Establishment

- If not, server sends reply with RST bit set to reject the connection
- Else server responds with its own SYN segment containing the server's initial sequence number, an ack (client's ISN plus one) for the client's SYN. A SYN consumes one sequence number.
- The client must acknowledge this SYN from the server by ACKing the server's ISN plus one



## TCP Connection Establishment



TCP connection establishment  
in the normal case

Call collision. The result of these  
events is that just one connection  
is established, not two because  
connections are identified by  
their end points

6 November 2005

EEC484/584

Wenbing Zhao

## TCP Connection Release

- Either side can send TCP segment with FIN set, i.e., no more data to transmit
- When FIN is ACKed, that direction is shut down
- When both directions are shut down, connection is released
- Normally, 4 TCP segments to release connection:  
1 FIN, 1 ACK in each direction
- Timers are used to avoid two-army problem
  - If response to FIN does not arrive in two max packet lifetimes, sender of FIN releases connection, other side also times out

6 November 2005

EEC484/584

Wenbing Zhao

## TCP Connection Management Modeling

- The states used in the TCP connection management finite state machine

| State       | Description                                      |
|-------------|--------------------------------------------------|
| CLOSED      | No connection is active or pending               |
| LISTEN      | The server is waiting for an incoming call       |
| SYN RCVD    | A connection request has arrived; wait for ACK   |
| SYN SENT    | The application has started to open a connection |
| ESTABLISHED | The normal data transfer state                   |
| FIN WAIT 1  | The application has said it is finished          |
| FIN WAIT 2  | The other side has agreed to release             |
| TIMED WAIT  | Wait for all packets to die off                  |
| CLOSING     | Both sides have tried to close simultaneously    |
| CLOSE WAIT  | The other side has initiated a release           |
| LAST ACK    | Wait for all packets to die off                  |

6 November 2005

EEC484/584

Wenbing Zhao

## TCP Connection Management Modeling

TCP connection management finite state machine

Heavy solid line - normal path for a client

Heavy dashed line - normal path for a server.

Light lines - unusual events

Each transition is labeled by the event causing it and the action resulting from it, separated by a slash.

