

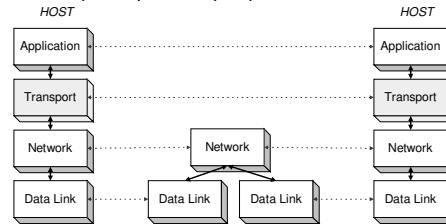
## Transport Layer

Flow control  
Connection management  
TCP, UDP

1

## Introduction

Transport layer protocols are end-to-end protocols  
Transport layer is only implemented at the hosts



Copyright Jorg Liebeherr 98, Modified with permission, Abdelzaher

2

## Why the Transport Layer?

- The IP layer provides a mechanism for transmitting datagrams from point A to point B.
- What are the missing pieces?
  - ?
  - ?
  - ?
- What are the functions of the transport layer?

Copyright Jorg Liebeherr 98, Modified with permission, Abdelzaher

3

## Functions of the Transport Layer

- Addressing
- Flow Control
- Error Control
- Connection Management

Note: The mechanisms needed to implement a transport service are largely dependent on the existing network layer service

Copyright Jorg Liebeherr 98, Modified with permission, Abdelzaher

4

## Addressing

- An address at the transport layer is typically a tuple (**Station, Port**) where
  - Station is the network address of the host, and
  - Port identifies the application
- The <IP address, port number> tuples used in Unix sockets are in fact transport layer addresses
- Problem with Addressing: How to find the address of a service?

Copyright Jorg Liebeherr 98, Modified with permission, Abdelzaher

5

## Homework 6

- What is the well known port number for the following services:
  - Web servers
  - SMTP (Simple Mail Transfer Protocol)
  - Quake
  - ICQ
  - Mobile IP (Home Agent)
  - FTP (both data and control ports)
  - Telnet
  - DNS (Domain Name Service)
  - SSH (Secure Shell – SecureCRT)
  - Kerberos Authentication

(Hint: Search Google for “well-known ports”)

Copyright Jorg Liebeherr 98, Modified with permission, Abdelzaher

6

## Flow Control

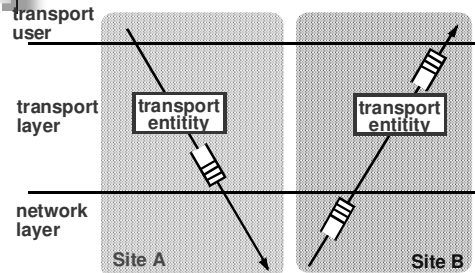
- Why do we need flow control at the transport layer?
  - User of receiving transport entity cannot keep up with the data flow.
  - Receiving transport entity itself cannot keep up with flow of incoming packet.

**Result:** Buffer overflows in the receiving transport entity.

Copyright Jorg Liebeherr 98, Modified with permission, Abdelzaher

7

## Need for Flow Control



Copyright Jorg Liebeherr 98, Modified with permission, Abdelzaher

8

## Flow Control at the Transport Layer

- Flow Control at the transport layer is more complex than flow control at the data link layer:
  - Delays are variable and are longer
  - Flow control involves the transport users, the transport entities, and the network service
- What's the problem with fixed-size sliding window flow control?

Copyright Jorg Liebeherr 98, Modified with permission, Abdelzaher

9

## Credit Allocation Flow Control

- Credit Allocation Flow Control is an extension of the sliding window flow control.
- Main Idea:**
  - Enhance the sliding window protocol by a mechanism that decouples acknowledgments from flow control.
- Then:**
  - Packets can be acknowledged without granting permission for new transmissions
  - Used in many existing transport protocols, including TCP

Copyright Jorg Liebeherr 98, Modified with permission, Abdelzaher

10

## Credit Allocation Flow Control

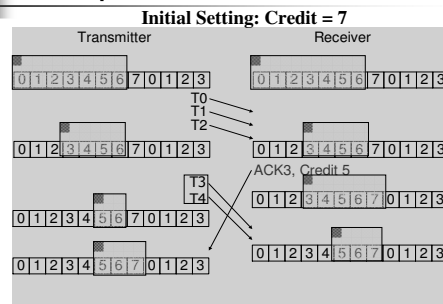
- Initialization during connection setup:
  - Set initial window size of receiver
  - Receiver both acknowledges TPDU's and grants credit by sending a message:
 

**(ACK N, CREDIT M)**
  - ACK N:** Acknowledges all sequence numbers through N-1
  - CREDIT M:** Sets the number of credits to M  
 Credit is the maximum window size (=buffer space at the receiver)

Copyright Jorg Liebeherr 98, Modified with permission, Abdelzaher

11

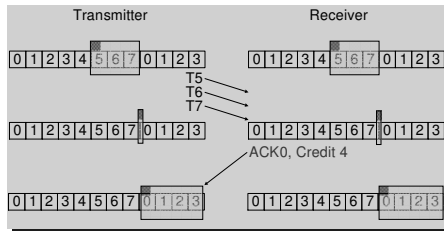
## Example



Copyright Jorg Liebeherr 98, Modified with permission, Abdelzaher

12

## Example



Copyright Jorg Liebeherr 98, Modified with permission, Abdelzaher

13

## Error Control at the Transport Layer

- Basic techniques for error recovery are the same as at the Data Link Layer:
  - Lost or damaged TPDU's are recovered with one of the ARQ retransmission schemes
  - Mostly: Go-back-N, Selective Repeat.
- Problem:** The end-to-end delay of TPDU's is variable. This makes it difficult to set the timeout values.
  - Small timeout value: unnecessary retransmissions.
  - Large timeout value: low throughput.
- Most transport protocols have adaptive timers

Copyright Jorg Liebeherr 98, Modified with permission, Abdelzaher

14

## Connection Management

- A connection is an abstraction of a communication channel between a sender and a receiver.
- A reliable connection guarantees that:
  - All transmitted packets are received correctly
  - All transmitted packets are received in the order they were sent
- Order is enforced via sequence numbers
- A connection may be bi-directional (e.g., sockets)
- Issues in connection management
  - Connection establishment
  - Connection termination

Copyright Jorg Liebeherr 98, Modified with permission, Abdelzaher

15

## Connection Establishment

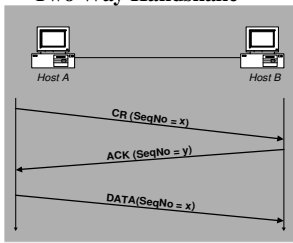
- Connection establishment is asymmetric:
  - one side puts itself in a LISTEN state (**server**)
  - one side issues a request for connection or RFC (**client**)

Copyright Jorg Liebeherr 98, Modified with permission, Abdelzaher

16

## Simple Solution (which has problems)

### Two Way Handshake

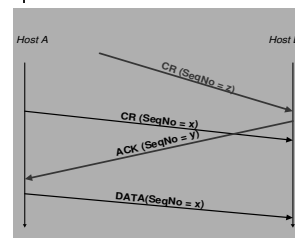


- CR (SeqNo = x)**  
Connection Request, A wants to start with SeqNo = x
- ACK (SeqNo = y)**  
Acknowledge request, B will want to start with SeqNo = y
- DATA (SeqNo = x)**  
Data transmission with SeqNo x

Copyright Jorg Liebeherr 98, Modified with permission, Abdelzaher

17

## Problems with Two-Way Handshake



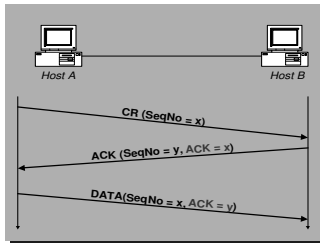
- B responds to CR(SeqNo = z), an old duplicate connection requests from A
- In the shown scenario, A believes that the ACK is for the connection request CR(SeqNo = y)

Result: A starts to send data with Sequence x. B will throw the data away since it expects SeqNo = z

Copyright Jorg Liebeherr 98, Modified with permission, Abdelzaher

18

## Three-Way Handshake



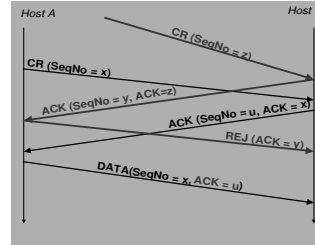
- Note: A and B acknowledge the sequence number from the other side
- This solution provides protection from old duplicate connection requests

Copyright Jörg Liebeherr 98, Modified with permission, Abdelzaher

19

## Scenario 1

Duplicate connection request (CR) appears



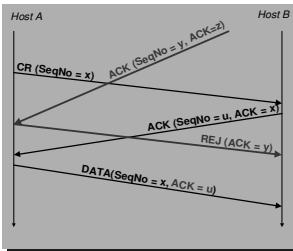
- Host A rejects the invalid connection request in the **REJ(ACK=y)** packet
- Note: The connection request CR(SeqNo=x) is completed successfully

Copyright Jörg Liebeherr 98, Modified with permission, Abdelzaher

20

## Scenario 2

A duplicate acknowledgement (ACK) appears



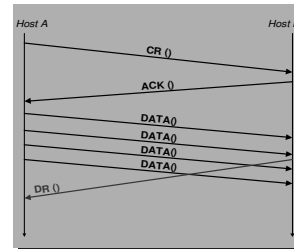
- Host A rejects the invalid ACK by sending **REJ(ACK=y)**
- Note: The connection request CR(SeqNo=x) is completed successfully

Copyright Jörg Liebeherr 98, Modified with permission, Abdelzaher

21

## Connection Termination

- A connection release should involve both sides of the connection (otherwise data is lost)



Here: B should wait after Disconnection Request (DR) is sent until all data has arrived

Copyright Jörg Liebeherr 98, Modified with permission, Abdelzaher

22

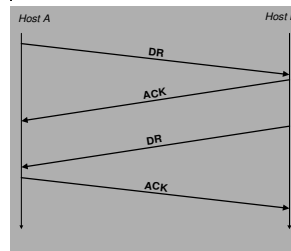
## Connection Termination in 4 steps

- An elegant way to terminate connections is to have each end shut down independently ("**half-close**")
- If one end wants to shut down, it sends a DR message
- Four steps involved:
  - (1) A sends a DR to B (**active close**)
  - (2) B ACKs the DR, (at this time: B can still send data to A)
  - (3) and B sends a DR to A (**passive close**)
  - (4) A ACKs the DR

Copyright Jörg Liebeherr 98, Modified with permission, Abdelzaher

23

## Connection Termination in 4 steps



To account for packet losses, a timer is needed to limit the waiting time of a side

Copyright Jörg Liebeherr 98, Modified with permission, Abdelzaher

24

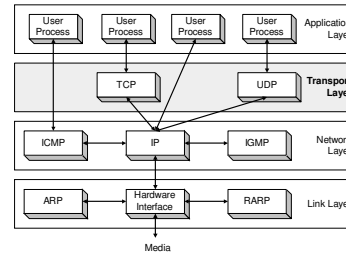
## Transport Protocols in the Internet

- The Internet uses two transport protocols
  - Transmission Control Protocol (TCP)
  - User Datagram protocol (UDP)

Copyright Jorg Liebeherr 98, Modified with permission, Abdelzaher

25

## Transport Protocols in the Internet



Copyright Jorg Liebeherr 98, Modified with permission, Abdelzaher

26

## Transport Protocols in the Internet

### UDP - User Datagram Protocol

- datagram oriented
- unreliable, connectionless
- simple
- unicast and multicast
- useful only for few applications, e.g., multimedia applications
- used a lot for services
  - network management (SNMP), routing (RIP), naming (DNS), etc.

### TCP - Transmission Control Protocol

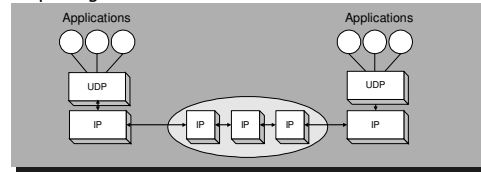
- stream oriented
- reliable, connection-oriented
- complex
- only unicast
- used for most Internet applications:
  - web (http), email (smtp), file transfer (ftp), terminal (telnet), etc.

Copyright Jorg Liebeherr 98, Modified with permission, Abdelzaher

27

## UDP - User Datagram Protocol

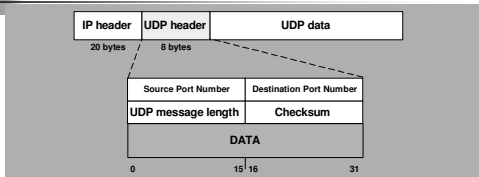
- UDP supports unreliable transmissions of datagrams
- UDP merely extends the host-to-host delivery service of IP datagrams to an application-to-application service
- The only thing that UDP adds is multiplexing and demultiplexing



Copyright Jorg Liebeherr 98, Modified with permission, Abdelzaher

28

## UDP Format



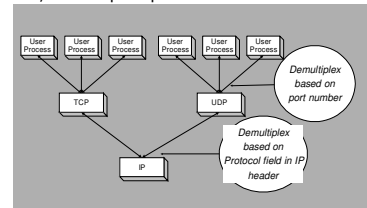
- Port numbers** identify sending and receiving applications (processes). Maximum port number is  $2^{16}-1 = 65,535$
- Message Length** is at least 8 bytes (i.e., Data field can be empty) and at most 65,535
- Checksum** is for header (of UDP and some of the IP header fields)

Copyright Jorg Liebeherr 98, Modified with permission, Abdelzaher

29

## Port Numbers

- UDP (and TCP) use port numbers to identify applications
- A globally unique address at the transport layer (for both UDP and TCP) is a tuple **<IP address, port number>**
- There are 65,535 UDP ports per host

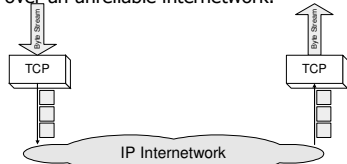


30

# TCP

## TCP = Transmission Control Protocol

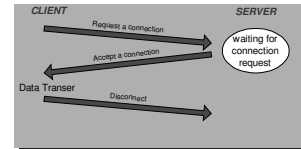
- Connection-oriented protocol
- Provides a reliable unicast end-to-end byte stream over an unreliable internetwork.



Copyright Jorg Liebeherr 98, Modified with permission, Abdelzaher

# TCP is Connection-Oriented

- Before any data transfer, TCP establishes a connection:
  - One TCP entity is waiting for a connection ("server")
  - The other TCP entity ("client") contacts the server
- The actual procedure for setting up connections is the three way handshake
- Each connection is full duplex



Copyright Jorg Liebeherr 98, Modified with permission, Abdelzaher

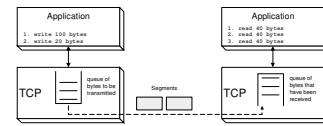
# Reliable Data Transfer

- Byte stream is broken up into chunks which are called **segments**
  - Receiver sends acknowledgements (ACKs) for segments
  - TCP maintains a timer. If an ACK is not received in time, the segment is retransmitted
- Detecting errors:
  - TCP has checksums for header and data. Segments with invalid checksums are discarded
  - Each byte that is transmitted has a sequence number

Copyright Jorg Liebeherr 98, Modified with permission, Abdelzaher

# TCP gives a Byte Stream Service

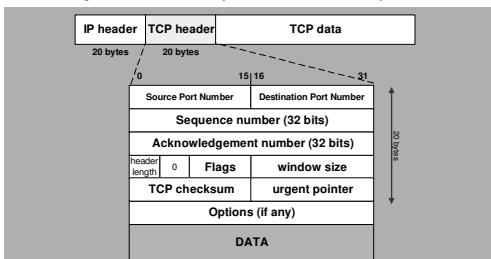
- To the lower layers, TCP handles data in blocks, the segments.
- To the higher layers TCP handles data as a sequence of bytes and does not identify boundaries between bytes
- So: Higher layers do not know about the beginning and end of segments !



Copyright Jorg Liebeherr 98, Modified with permission, Abdelzaher

# TCP Segment Format

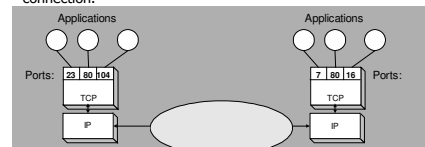
TCP segments have a 20 byte header with >= 0 bytes of data.



Copyright Jorg Liebeherr 98, Modified with permission, Abdelzaher

# TCP header fields

- Port Number:**
  - A port number identifies the endpoint of a connection.
  - A pair <IP address, port number> identifies one endpoint of a connection.
  - Two pairs <client IP address, server port number> and <server IP address, server port number> identify a TCP connection.



Copyright Jorg Liebeherr 98, Modified with permission, Abdelzaher

## TCP header fields

- Sequence Number (**SeqNo**):
  - Sequence number is 32 bits long.
  - So the range of **SeqNo** is
    - $0 \leq \text{SeqNo} \leq 2^{32} - 1 \approx 4.3 \text{ Gbyte}$
  - Each sequence number identifies a byte in the byte stream
  - Initial Sequence Number (ISN) of a connection is set during connection establishment

Copyright Jorg Liebeherr 98, Modified with permission, Abdelzaher

37

## TCP header fields

- Acknowledgement Number (**AckNo**):
  - Acknowledgements are piggybacked
  - A hosts uses the AckNo field to send acknowledgements. (If a host sends an AckNo in a segment it sets the "ACK flag")
  - The AckNo contains the next SeqNo that a hosts wants to receive  
Example: The acknowledgement for a segment with sequence numbers 0-1500 is AckNo=1501

Copyright Jorg Liebeherr 98, Modified with permission, Abdelzaher

38

## TCP header fields

- Flag bits:
  - URG**: Urgent pointer is valid
    - If the bit is set, the following bytes contain an urgent message in the sequence number range "SeqNo <= urgent message <= SeqNo+urgent pointer"
  - ACK**: Segment carries a valid acknowledgement
  - PSH**: PUSH Flag
    - Notification from sender to the receiver that the receiver should pass all data that it has to the application.
    - Normally set by sender when the sender's buffer is empty

Copyright Jorg Liebeherr 98, Modified with permission, Abdelzaher

39

## TCP header fields

- Flag bits:
  - RST**: Reset the connection
    - The flag causes the receiver to reset the connection
    - Receiver of a RST terminates the connection and indicates higher layer application about the reset
  - SYN**: Synchronize sequence numbers
    - Sent in the first packet when initiating a connection
  - FIN**: Sender is finished with sending
    - Used for closing a connection
    - Both sides of a connection must send a FIN

Copyright Jorg Liebeherr 98, Modified with permission, Abdelzaher

40

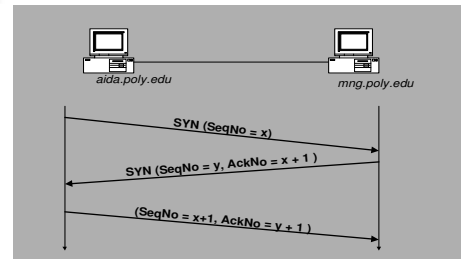
## TCP Connection Establishment

- TCP uses a three-way handshake
  - ACTIVE OPEN**: Client sends a segment with
    - SYN bit set \*
    - port number of client
    - initial sequence number (ISN) of client
  - PASSIVE OPEN**: Server responds with a segment
    - SYN bit set \*
    - initial sequence number of server
    - ACK for ISN of client
  - Client acknowledges** by sending a segment with
    - ACK ISN of server (\* counts as one byte)

Copyright Jorg Liebeherr 98, Modified with permission, Abdelzaher

41

## Three-Way Handshake



Copyright Jorg Liebeherr 98, Modified with permission, Abdelzaher

42



## Flow Control

- Sender does not have to send all data immediately
- Receiver does not have to deliver immediately
- What if
  - Sender application generates one character at a time?
  - Receiver application consumes one character at a time?

Copyright Jorg Liebeherr 98, Modified with permission, Abdelzaher

49

## Nagle's Algorithm

- Deals with slow senders
  - Send first byte
  - Hold back until ACK is received, then send what's in the buffer
  - Send if accumulated MSS worth of data or if accumulated half the window size worth of data

Copyright Jorg Liebeherr 98, Modified with permission, Abdelzaher

50

## The Silly Window Syndrome

- If receiver dequeues its buffer one character at a time it gives "silly" window information to the sender
- Solution: Clark's Algorithm
  - Do not send window updates until you can handle one MSS or half your buffer is empty.

Copyright Jorg Liebeherr 98, Modified with permission, Abdelzaher

51

## Slow Start/Congestion Avoidance

- TCP has a mechanism for congestion control. The mechanism is implemented at the sender
- The sender has two additional parameters:
  - **Congestion Window (cwnd)**; Initial value is 1 MSS counted as bytes)
  - **Threshold Value (ssthresh)**; Initial value is 65536 bytes)
- The window size at the sender is set as follows:
 

**Allowed Window = MIN (flow control window, congestion window)**

Copyright Jorg Liebeherr 98, Modified with permission, Abdelzaher

52

## Slow Start

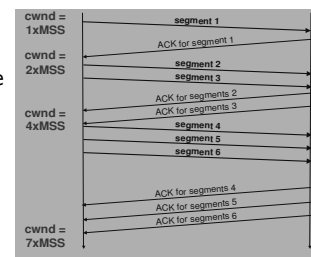
- Whenever starting traffic on a new connection, or whenever increasing traffic after congestion was experienced:
  - Set *cwnd* = MSS bytes (=1 segment)
  - Each time an ACK is received, the congestion window is increased by 1 segment (= MSS bytes).
  - If an ACK acknowledges two segments, *cwnd* is still increased by only 1 segment.
  - Even if ACK acknowledges a segment that is smaller than MSS bytes long, *cwnd* is increased by MSS bytes.
- Does Slow Start increment slowly? Not really. In fact, the increase of *cwnd* can be exponential

Copyright Jorg Liebeherr 98, Modified with permission, Abdelzaher

53

## Slow Start Example

- The congestion window size grows very rapidly
  - For every ACK, we increase *cwnd* by 1 irrespective of the number of segments ACK'ed
- TCP slows down the increase of *cwnd* when ***cwnd* > *ssthresh***



Copyright Jorg Liebeherr 98, Modified with permission, Abdelzaher

54

## Slow Start

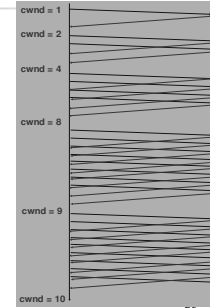
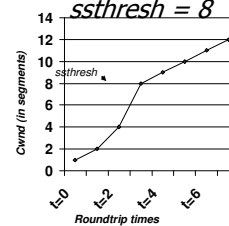
- n "Slow Start" slows down if the congestion window is larger than a threshold value
- n If  $cwnd > ssthresh$  then each time an ACK is received, increment  $cwnd$  as follows:
  - o  $cwnd = cwnd + MSS * MSS / cwnd$
- n So  $cwnd$  is increased by one only if all segments have been acknowledged.
- n  $ssthresh$  is modified if there is congestion in the network

Copyright Jörg Liebeherr 98, Modified with permission, Abdelzaher

55

## Slow Start Example

Assume that  $ssthresh = 8$



Copyright Jörg Liebeherr 98, Modified with permission, Abdelzaher

56

## Congestion Avoidance

- n Most often, a packet loss in a network is due to an overflow at a congested router (rather than due to a transmission error)
- n A sender can detect lost packets through a:
  - o Timeout of a retransmission timer
  - o Receipt of a duplicate ACK
- n TCP assumes that a packet loss is caused by congestion and reduces the size of the sending window
- n The algorithm that reduces and then reopens the sending window is called **Congestion Avoidance**

Copyright Jörg Liebeherr 98, Modified with permission, Abdelzaher

57

## Slow Start / Congestion Avoidance

- n Implemented with two variables:
  - o Congestion window ( $cwnd$ )
  - o Slow start threshold ( $ssth$ )
- n Initialization:
  - o  $cwnd = MSS$  bytes
  - o  $ssth = 64535$  bytes
- n The window size at the sender is set as follows:
  - o Allowed Window = MIN (advertised window,  $cwnd$ )

Copyright Jörg Liebeherr 98, Modified with permission, Abdelzaher

58

## Slow Start / Congestion Avoidance

- n Here we give a more accurate version than in our earlier discussion of Slow Start:

```

If  $cwnd \leq ssthresh$  then
    Each time an Ack is received:
     $cwnd = cwnd + MSS$ 
else /*  $cwnd > ssthresh$  */
    Each time an Ack is received :
     $cwnd = cwnd + MSS * MSS / cwnd + segsize / 8$ 
endif
    
```

Copyright Jörg Liebeherr 98, Modified with permission, Abdelzaher

59

## Slow Start / Congestion Avoidance

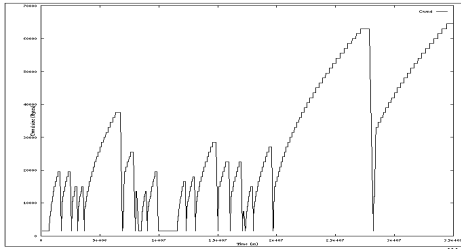
- n Each time when congestion occurs (timeout or receipt of duplicate ACK),
  - n  $cwnd$  is reset to one:
    - o  $cwnd = 1$
  - n  $ssthresh$  is set to half the current size of the congestion window:
    - o  $ssthresh = cwnd / 2$

Copyright Jörg Liebeherr 98, Modified with permission, Abdelzaher

60

## Slow Start / Congestion Avoidance

- A typical plot of cwnd for a TCP connection (MSS = 1500 bytes) :



Copyright Jorg Liebeherr 98, Modified with permission, Abdelzaher

## Error Control in TCP

- TCP implements a variation of the **Go-back-N** retransmission scheme
- TCP couples error control and congestion control (I.e., it assumes that errors are caused by congestion)

Copyright Jorg Liebeherr 98, Modified with permission, Abdelzaher

62

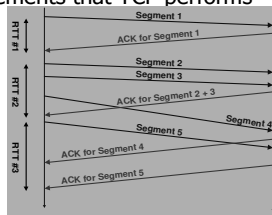
## Round-Trip Time Measurements

- The retransmission mechanism of TCP is adaptive
- The retransmission timers are set based on round-trip time (RTT) measurements that TCP performs

The RTT is based on time difference between segment transmission and receipt of ACK

**But:**

- TCP does not ACK each segment
- Each connection has only one timer



Copyright Jorg Liebeherr 98, Modified with permission, Abdelzaher

## Round-Trip Time Measurements

- Retransmission timer is set to a **Retransmission Timeout (RTO)** value
- RTO is calculated based on the *RTT* measurements
- The RTT measurements are smoothed by the following estimators *srtt* and *rttvar*:

$$srtt_{n+1} = \alpha RTT + (1 - \alpha) srtt_n$$

$$rttvar_{n+1} = \beta ( | RTT - srtt_{n+1} | ) + (1 - \beta) rttvar_n$$

$$RTO_{n+1} = srtt_{n+1} + 4 rttvar_{n+1}$$

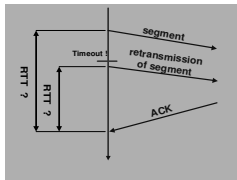
- The gains are set to  $\alpha = 1/4$  and  $\beta = 1/8$
- $srtt_0 = 0 \text{ sec}$ ,  $rttvar_0 = 3 \text{ sec}$ , Also:  $RTO_0 = srtt_0 + 2 rttvar_0$

Copyright Jorg Liebeherr 98, Modified with permission, Abdelzaher

64

## Karn's Algorithm

- If an ACK for a retransmitted segment is received, the sender cannot tell if the ACK belongs to the original or the retransmission.



**Karn's Algorithm:**

Don't update *srtt* on any segments that have been retransmitted.

Each time when TCP retransmits, it sets:

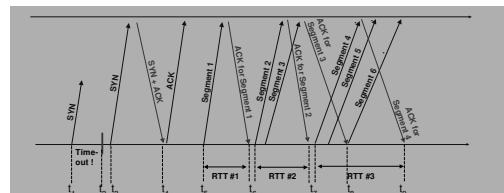
$$RTO_{n+1} = \max ( 2 RTO_n, 64 ) \quad (\text{exponential backoff})$$

Copyright Jorg Liebeherr 98, Modified with permission, Abdelzaher

65

## RTO Calculation: Example

- At  $t_1$ :  $RTO = srtt + 2 rttvar = 6 \text{ sec}$
- At  $t_2$ :  $RTO = 2 * (srtt + 4 rttvar) = 24 \text{ sec}$  (exponential backoff)
- At  $t_3$ : RTO is not updated (Due to Karn's algorithm)



Copyright Jorg Liebeherr 98, Modified with permission, Abdelzaher

66