Telematics
Chapter 7: Transport Layer

Prof. Dr. Mesut Güneş
Computer Systems and Telematics (CST)
Distributed, embedded Systems
Institute of Computer Science
Freie Universität Berlin
http://cst.mi.fu-berlin.de
Contents

- Design Issues
- User Datagram Protocol (UDP)
- Transmission Control Protocol (TCP)
- An example of socket programming
Design Issues
Design Issues

- Characteristics of the layers **below** the Transport Layer
  - Available on the hosts and on the routers
  - Operate in a hop-to-hop fashion
  - Typically unreliable

- Characteristics of the Transport Layer
  - Available only on the hosts
  - Operate in a end-to-end fashion
  - It has to operate like a pipe

- Services provided to the upper layers
  - Connection-oriented service
  - Connectionless service
  - Convenient interface for application programmers
    - Berkeley sockets

### OSI Reference Model

<table>
<thead>
<tr>
<th>Layer</th>
</tr>
</thead>
<tbody>
<tr>
<td>Application Layer</td>
</tr>
<tr>
<td>Presentation Layer</td>
</tr>
<tr>
<td>Session Layer</td>
</tr>
<tr>
<td><strong>Transport Layer</strong></td>
</tr>
<tr>
<td>Network Layer</td>
</tr>
<tr>
<td>Data Link Layer</td>
</tr>
<tr>
<td>Physical Layer</td>
</tr>
</tbody>
</table>
Services Provided to the Upper Layers

- Services provided to the upper layers
  - Goal is to provide an efficient, reliable, and cost-effective service
  - **Transport entity** is responsible to provide that service
    - Maybe located in the operating system kernel, user process, library package, or network interface card

![Diagram showing the transport layer]
Transport Service Primitives

● Some terminology
  ● Messages from transport entities: Transport Protocol Data Unit (TPDU)
  ● TPDUs are contained in network layer packets
  ● Packets are contained in data link layer frames

![Diagram of frame, packet, and TPDU structures]
Transport Service Primitives

- Processes on the application layer expect 100% reliable connections
  - They are not interested in acknowledgements, lost packets, congestions, ...
- Transport layer provides
  - Unreliable datagram service (Connectionless)
  - Reliable connection-oriented service
    - Three phases: establishment, communication, termination

- The primitives for a simple connection-oriented transport service

<table>
<thead>
<tr>
<th>Primitive</th>
<th>Packet sent</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>LISTEN</td>
<td>(none)</td>
<td>Block until some process tries to connect</td>
</tr>
<tr>
<td>CONNECT</td>
<td>CONNECTION REQ.</td>
<td>Actively attempt to establish a connection</td>
</tr>
<tr>
<td>SEND</td>
<td>DATA</td>
<td>Send information</td>
</tr>
<tr>
<td>RECEIVE</td>
<td>(none)</td>
<td>Block until a DATA packet arrives</td>
</tr>
<tr>
<td>DISCONNECT</td>
<td>DISCONNECTION REQ.</td>
<td>This side wants to release the connection</td>
</tr>
</tbody>
</table>
Transport Service Primitives

Client

Connect
Send
Receive
Disconnect
Receive

Server

Listen
Receive
Send
Receive
Disconnect

Connection-Req.
Connection-Accepted
Data
Disconnection-Req.
Disconnection-Accepted

Prof. Dr. Mesut Güneş • cst.mi.fu-berlin.de • Telematics • Chapter 7: Transport Layer
Transport Service Primitives

● A state diagram for a simple connection management scheme
  ● Transitions labeled in italics are caused by packet arrivals
  ● The solid lines show the client's state sequence
  ● The dashed lines show the server's state sequence

![State Diagram]

- IDLE
- PASSIVE
- ESTABLISHMENT
- PENDING
- ACTIVE
- ESTABLISHMENT
- PENDING
- ESTABLISHED
- PASSIVE
- DISCONNECT
- PENDING
- IDLE

Connection request
TPDU received

Connect primitive
executed

Disconnection
request TPDU
received

Disconnect primitive
executed

Connection accepted
TPDU received

Disconnection request
TPDU received

Connect primitive
executed

Prof. Dr. Mesut Güneş • cst.mi.fu-berlin.de • Telematics • Chapter 7: Transport Layer
Transport Protocol

Transport protocol
- Transport service is implemented between transport entities by a transport protocol
- Similar to the protocols studied in chapter “Data Link Layer”
  - Have to deal with: error control, sequencing, and flow control

- Environment of the data link layer
  - On DLL two router communicate directly via a physical channel
    - No explicit addressing is required
    - Channel always there

- Environment of the transport layer
  - On TL channel is given by the subnet
    - Explicit addressing of the destination is required
    - Channel is not always there
    - Connection establishment is complicated
Transport Protocol: Addressing

- Addressing on the transport layer
  - To which process connect to?
  - Transport Service Access Point (TSAP)
  - Network Service Access Point (NSAP)

- Questions
  - How does the process on Host 1 know the TSAP of Server 1 on Host 2?

- Solution
  - TSAPs are stored in a specific file:
    - Unix: /etc/services
    - Windows: \system32\drivers\etc\services
Transport Protocol: Addressing

- Disadvantage of previous solution
  - For small number of servers the solution with specific files works fine.

- Problem
  - When there are many servers, which are rarely used.

- Solution
  - Special process: Process Server
    - Listens to a set of TSAPs
  - When desired server is not active, connection is made with the Process Server
  - Process Servers starts the server for the desired service and passes the connection to it
Transport Protocol: Addressing

- Disadvantage of previous solution
  - What happens if server cannot be started, i.e., service depends on the process?
- Solution
  - Special process: Name Server / Directory Server
  - Client first connects to the Name Server and requests the TSAP of the service
  - Subsequently, connection is established with the desired server
- Requirement
  - Servers have to register with the Name Server
    - Records of (Name, TSAP)
Transport Protocol: Connection Establishment

- Problems with connection establishment
  - When network can lose, store, and duplicate packets
  - Delayed duplicates

- Solution approaches
  - Throw-away TSAPs: For each connection a new TSAP is used
    - Management of used TSAPs
  - Restrict packet life time, e.g., by a TTL field, timestamp, ...

- Solution of Tomlinson, Sunshine, and Dalal (Three-way Handshake)
  - Each computer has a clock (time of day)
  - Clocks do not need to be synchronized
  - Clock has to run even when the computer crashes or is switched off
  - Idea: Put sequence numbers into TPDUs and two TPDUs with the same sequence number may not outstand at the same time
    - Each connection starts with a different initial sequence number
Transport Protocol: Connection Establishment

Assumption: After $T$ time units TPDU and Ack are dead and have no effect.

TPDUs may not enter the forbidden region

The resynchronization problem
Transport Protocol: Connection Establishment

- Connection Establishment with the **Three-way Handshake**
  - Three protocol scenarios for establishing a connection using a three-way handshake. CR denotes CONNECTION REQUEST
    - a) Normal operation
    - b) Old CONNECTION REQUEST appearing out of nowhere
    - c) Duplicate CONNECTION REQUEST and duplicate ACK

Prof. Dr. Mesut Güneş • cst.mi.fu-berlin.de • Telematics • Chapter 7: Transport Layer
Transport Protocol: Connection Release

- Terminating a connection
  - Asymmetric release
    - Telephone system model
    - Either one peer can terminate the connection
    - Danger of data loss
  - Symmetric release
    - Model of two independent unicast connections
    - Each peer has to terminate the connection explicitly
    - Data can be received by in the non-terminated direction
- Problem: Data loss can happen on both cases
  - Question: Is there an optimal solution?

Diagram:
- Host 1 to Host 2
  - CR: Disconnect Request
  - ACK: Acknowledgment
  - DATA: Data packet
  - DR: Disconnect Request

Abrupt disconnection with loss of data.
DR denotes Disconnect Request.

Prof. Dr. Mesut Güneş • cst.mi.fu-berlin.de • Telematics • Chapter 7: Transport Layer
Famous example to illustrate the problem of controlled (reliable) connection termination: **The two-army problem**
- The Blue armies can only communicate with messengers, i.e., soldiers running through the valley
- Messengers are subject to loss
Transport Protocol: Connection Release

Four protocol scenarios for releasing a connection

(a) Normal case of a three-way handshake
(b) final ACK lost
Transport Protocol: Connection Release

- Four protocol scenarios for releasing a connection
  
  c) Response lost
  
  d) Response lost and subsequent DRs lost
Transport Protocol: Multiplexing

- Multiplexing of several conversations onto connections
  
  (a) Upward multiplexing: Many transport connections use the same network address
  
  (b) Downward multiplexing: Distribute the traffic of one connection over many network connections
Transport Protocol: Crash Recovery

- If hosts and routers are subject to crashes, recovery becomes an issue.

Scenario
- A client sends a large file to a server.
- Each chunk of the transmitted file is acked by the server.
- After a crash server does not know the status.

Possible client states
- S0: No outstanding ack.
- S1: One outstanding ack.

Client strategies
- Always retransmit last TPDU.
- Never retransmit last TPDU.
- Retransmit last TPDU in S0.
- Retransmit last TPDU in S1.
Transport Protocol: Crash Recovery

- Processing strategies of server
  - Strategy 1: First send ack, then write to application
  - Strategy 2: First write to application, then send ack

- Different combinations of client and server strategy
  - Server events are \{A=\text{Ack}, W=\text{Write}, C=\text{Crash}\}

<table>
<thead>
<tr>
<th>Strategy used by sending host</th>
<th>First ACK, then write</th>
<th>First write, then ACK</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>AC(W)</td>
<td>AWC</td>
</tr>
<tr>
<td>Always retransmit</td>
<td>OK</td>
<td>DUP</td>
</tr>
<tr>
<td>Never retransmit</td>
<td>LOST</td>
<td>OK</td>
</tr>
<tr>
<td>Retransmit in S0</td>
<td>OK</td>
<td>DUP</td>
</tr>
<tr>
<td>Retransmit in S1</td>
<td>LOST</td>
<td>OK</td>
</tr>
</tbody>
</table>

S0: No outstanding ack
S1: One outstanding ack

OK: Protocol functions correctly
DUP: Protocol duplicates message
LOST: Protocol loses a message
Transport Protocols in the TCP/IP Reference Model
Transport Protocols in the TCP/IP Reference Model

Connection-oriented
- HTTP
- FTP
- Telnet
- SMTP
- DNS
- SNMP
- TFTP

Connectionless
- TCP
- UDP

Application Layer

Transport Layer

Internet Layer

Host-to-Network Layer

TCP (Transmission Control Protocol): Reliable, connection-oriented

UDP (User Datagram Protocol): Datagram principle, connectionless
The Transport Layer: TCP and UDP

- Transport protocols are used by the application layer as communication services
  - They allow the communication between application processes
- TCP is a connection-oriented protocol
- UDP is a connectionless protocol
User Datagram Protocol (UDP)
The User Datagram Protocol (UDP)

- Principle: “Keep it simple!”
  - 8 byte header
  - Like IP: connectionless and unreliable
  - Small reliability, but fast exchange of information
  - No acknowledgement between communication peers with UDP
    - Incorrect packets are simply discarded
    - Duplication, sequence order permutation, and packet loss are possible
  - The checksum offers the only possibility of testing the packets on transfer errors
  - Possible: ACKs and retransmissions are controlled by the application
  - Use in multicast (not possible with TCP)

- Why at all UDP?
  - Only the addition of a **port** to a **network address** marks communication unique

(IP Address₁, Port₁, IP Address₂, Port₂)
## UDP Header

- **Source Port, Destination Port:** Addressing of the applications by port numbers
- **Length:** The total length of the datagram (header + data) in 32-bit words
- **Checksum** (optional): IP does not have a checksum for the data part, therefore it can be a meaningful addition here
  - The same procedure as in TCP
- **Data:** The payload, it is filled up if necessary to an even byte number, since message length counts in 32-bit words

<table>
<thead>
<tr>
<th>0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5</th>
<th>6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>Source Port</td>
<td>Destination Port</td>
</tr>
<tr>
<td>Length</td>
<td>Checksum</td>
</tr>
<tr>
<td>Data</td>
<td></td>
</tr>
</tbody>
</table>
UDP-based Applications

- Port number is 16-bit address, currently three different ranges
  - Ports in the range 0-1023 are **Well-Known** (a.k.a. “system”)
  - Ports in the range 1024-49151 are **Registered** (a.k.a. “user”)
  - Ports in the range 49152-65535 are **Dynamic/Private**
  - See for more information: [http://www.iana.org/assignments/port-numbers](http://www.iana.org/assignments/port-numbers)
User Datagram Protocol (UDP)
Socket Programming with UDP
Socket Programming with UDP

Server (on host `hostid`)

- create socket, port=x,
- for incoming request:
  - `serverSocket = DatagramSocket()`

Client

- create socket,
  - `clientSocket = DatagramSocket()`
- Create, address `(hostid, port=x)`, send datagram request using `clientSocket`
- read reply from `clientSocket`
- write reply to `serverSocket` specifying client host address, port number
- close `clientSocket`

For incoming request:

- `serverSocket = DatagramSocket()`
- read request from `serverSocket`
- write reply to `serverSocket`
import java.io.*;
import java.net.*;

class UDPClient {
    public static void main(String arg []) throws Exception {
        BufferedReader inFromUser = new BufferedReader(new InputStreamReader(System.in));
        DatagramSocket clientSocket = new DatagramSocket();
        InetAddress IPAddress = InetAddress.getByName("hostname");
        byte[] sendData = new byte[1024];
        byte[] receiveData = new byte[1024];
        String sentence = inFromUser.readLine();
        sendData = sentence.getBytes();
        DatagramPacket send_pack = new DatagramPacket(sendData, sendData.length,
                                                   IPAddress, 9876);
        clientSocket.send(send_pack);
        DatagramPacket receivePacket = new DatagramPacket(receiveData, receiveData.length);
        clientSocket.receive(receivePacket);
        String modifiedSentence = new String(receivePacket.getData());
        System.out.println("FROM SERVER:" + modifiedSentence);
        clientSocket.close();
    }
}
import java.io.*;
import java.net.*;

class UDPServer {
    public static void main(String args[]) throws Exception {
        DatagramSocket serverSocket = new DatagramSocket(9876);

        byte[] receiveData = new byte[1024];
        byte[] sendData = new byte[1024];

        while (true) {
            DatagramPacket receivePacket = new DatagramPacket(receiveData, receiveData.length);
            serverSocket.receive(receivePacket);
            String sentence = new String(receivePacket.getData());
            InetAddress IPAddress = receivePacket.getAddress();
            int port = receivePacket.getPort();
            String capitalizedSentence = sentence.toUpperCase();
            sendData = capitalizedSentence.getBytes();
            DatagramPacket sendPacket = new DatagramPacket(sendData, sendData.length,
                                                       IPAddress, port);
            serverSocket.send(sendPacket);
        }
    }
}
Transmission Control Protocol (TCP)
Characteristics of TCP

- Connection-oriented and reliable (error-free, keeps packet order, without duplicates)
- Error handling, acknowledgements, flow control (Sliding Window procedure)
- **Byte stream**, not message stream, i.e., message boundaries are not preserved
- Segmentation (max. segment size of 64 KByte)
- “Urgent”-messages outside of flow control
- Limited QoS
- Addressing of the application by port numbers
  - Port numbers below 1024 are called **well-known ports**, these are reserved for standard services

![Layer Diagram](image)

Prof. Dr. Mesut Güneş • cst.mi.fu-berlin.de • Telematics • Chapter 7: Transport Layer
TCP as a Reliable Connection

If the server port is unknown, the use of a process server (Initial Connection Protocol) is possible:

Alternatively: Name server (comparable to a phone book) returns the destination port

Prof. Dr. Mesut Güneş • cst.mi.fu-berlin.de • Telematics • Chapter 7: Transport Layer
TCP as a Reliable Connection

- Establishes logical connections between two Sockets
  - IP address + 16 bit port number (48 bit address information)
    \[(\text{IP Address}_1, \text{Port}_1, \text{IP Address}_2, \text{Port}_2)\]
  - For an application, sockets are the access points to the network
  - A socket can be used for several connections at the same time
- TCP connections are always **full-duplex** and **point-to-point** connections
- TPDUs exchanged between the two communicating stations are called **segments**
- Segments are being exchanged for realizing
  - Connection establishment
  - Agreement on a window size
  - Data transmission
  - Sending of confirmations
  - Connection termination
TCP: Overview
RFCs: 793, 1122, 1323, 2018, 2581

● Point-to-point
  ● One sender, one receiver
● Reliable, in-order byte steam:
  ● No “message boundaries”
● Pipelined
  ● TCP congestion and flow control set window size
● Send & receive buffers

● Full duplex data
  ● Bi-directional data flow in same connection
  ● MSS: maximum segment size
● Connection-oriented
  ● Handshaking (exchange of control msgs) init’s sender, receiver state before data exchange
● Flow controlled
  ● Sender will not overwhelm receiver
Transmission Control Protocol (TCP)

Socket Programming in TCP
Socket Programming in TCP

**Server side**
- The receiving application process (server) has to run at first
- The server provides a socket over which connection requests are received (i.e. a port is made available)
- In order to be able to receive requests of several clients, the server provides a new socket for a connection request of each client

**Client side**
- The client generates a socket
- The client creates a request with IP address and port of the server
- When the client creates its socket, a connection establishment to the server is made
Socket Primitives in TCP

- For communication via TCP, a set of primitives exists which an application programmer can use for initializing and carrying out a communication.

- The essential primitives are:

<table>
<thead>
<tr>
<th>Primitive</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>SOCKET</td>
<td>Creation of a new network access point</td>
</tr>
<tr>
<td>BIND</td>
<td>Assign a local address with the socket</td>
</tr>
<tr>
<td>LISTEN</td>
<td>Wait for arriving connecting requests</td>
</tr>
<tr>
<td>ACCEPT</td>
<td>Accept a connecting request</td>
</tr>
<tr>
<td>CONNECT</td>
<td>Attempt of a connection establishment</td>
</tr>
<tr>
<td>SEND</td>
<td>Send data over the connection</td>
</tr>
<tr>
<td>RECEIVE</td>
<td>Receive data on the connection</td>
</tr>
<tr>
<td>CLOSE</td>
<td>Release of the connection</td>
</tr>
</tbody>
</table>
Socket programming in TCP

Server (on host \textbf{hostid})

- create socket, \texttt{port=x}, for incoming request:

  \begin{verbatim}
  welcomeSocket = ServerSocket()
  \end{verbatim}

- wait for incoming connection request

  \begin{verbatim}
  connectionSocket = welcomeSocket.accept()
  \end{verbatim}

- read request from \texttt{connectionSocket}

- write reply to \texttt{connectionSocket}

- close \texttt{connectionSocket}

Client

- create socket,

  \begin{verbatim}
  clientSocket = Socket()
  \end{verbatim}

- connect to \texttt{hostid, port=x}

- send request using \texttt{clientSocket}

- read reply from \texttt{clientSocket}

- close \texttt{clientSocket}
import java.io.*;
import java.net.*/;

class TCPClient {
  public static void main(string argv[]) throws Exception {
    String sentence;
    String modifiedSentence;
    BufferedReader inFromUser = new BufferedReader(new InputStreamReader(System.in));
    Socket clientSocket = new Socket("hostname", 6789);
    DataOutputStream outToServer = new DataOutputStream(clientSocket.getOutputStream());
    BufferedReader inFromServer = new BufferedReader(new InputStreamReader(clientSocket.getInputStream()));
    sentence = inFromUser.readLine();
    outToServer.writeBytes(sentence + "\n");
    modifiedSentence = inFromServer.readLine();
    System.out.println("FROM SERVER: " + modifiedSentence);
    clientSocket.close();
  }
}
import java.io.*;
import java.net.*;

class TCPServer {
    public static void main(String arg []) throws Exception {
        String clientSentence;
        String capitalizedSentence;
        ServerSocket welcomeSocket = new ServerSocket(6789);

        while (true) {
            Socket connectionSocket = welcomeSocket.accept();
            BufferedReader inFromClient =
                new BufferedReader(new InputStreamReader(connectionSocket.getInputStream()));
            DataOutputStream outToClient = new DataOutputStream(connectionSocket.getOutputStream());
            
            clientSentence = inFromClient.readLine();
            capitalizedSentence = clientSentence.toUpperCase() + "\n";
            outToClient.writeBytes(capitalizedSentence);
        }
        welcomeSocket.close();
    }
}
Transmission Control Protocol (TCP)

The TCP Header
TCP-based Applications

Well known TCP ports:
- Port number is 16-bit address, currently three different ranges
- Ports in the range 0-1023 are **Well-Known** (a.k.a. “system”)
- Ports in the range 1024-49151 are **Registered** (a.k.a. “user”)
- Ports in the range 49152-65535 are **Dynamic/Private**
- See for more information: [http://www.iana.org/assignments/port-numbers](http://www.iana.org/assignments/port-numbers)
The TCP Header

- 20 byte header
- Plus options
- Up to 65495 data bytes

Counting by bytes of data (not segments!)

Number of bytes receiver willing to accept
The TCP Header

- Source and Destination Port: port number of sender resp. receiver
- Sequence Number/Acknowledgment Number: Segments have a 32 bit sequence and acknowledgement number for the window mechanism in flow control (Sliding Window).
  - Sequence and acknowledgement number count single bytes!
  - The **acknowledgement number** indicates the next expected byte!
  - **Sequence numbers** begin not necessarily with 0! A *random value is chosen* to avoid a possible mix-up of old (late) segments.
  - Piggybacking, i.e., an acknowledgement can be sent in a data segment.
- Header Length: As in case of IP, also the TCP header has an indication of its length. The length is **counted in 32-bit words**.
- Window Size: Size of the receiver’s buffer for the connection.
  - Used in flow control: the window of a flow indicates how many bytes at the same time can be sent.
  - The size of the buffer indicates, the number of bytes the receiver can accept.
  - The window of flow control is adapted to this value.

![TCP Header Diagram]

Prof. Dr. Mesut Güneş • cst.mi.fu-berlin.de • Telematics • Chapter 7: Transport Layer
The TCP Header

- **Flags:**
  - **URG:** Signaling of special important data, e.g., abort, Ctrl-C
  - **ACK:** This bit is set, if an acknowledgement is sent
  - **PSH:** Immediate transmission of data, no more waiting for further data
  - **RST:** Reset a connection, e.g., during a host crash or a connecting rejection
    - Generally problems arise when a segment with set RST bit is received
  - **SYN:** set to 1 for connection establishment
  - **FIN:** set to 1 for connection termination
- **Urgent pointer:** indicates, at which position in the data field the urgent data ends (byte offset of the current sequence number).

- **Option:**
  - **Negotiation of a window scale:** Window size field can be shifted up to 14 bits
    - allowing windows of up to $2^{30}$ bytes
  - **Use of Selective Repeat instead of Go-Back**
  - **Indication of the Maximum Segment Size (MSS) to determine the size of the data field**
TCP Pseudo Header

- Checksum: serves among other things for the verification that the packet was delivered to the correct device.
  - The checksum is computed using a **pseudo header**. The pseudo header is placed in front of the TCP header, the checksum is computed based on both headers (the checksum field is here 0).
    - The checksum is computed as the 1-complement of the sum of all 16-bit words of the segment including the pseudo header.
  - The receiver also places the pseudo header in front of the received TCP header and executes the same algorithm (the result must be 0).

<table>
<thead>
<tr>
<th>Source address (IP)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Destination address (IP)</td>
</tr>
<tr>
<td>000000000</td>
</tr>
<tr>
<td>Protocol = 6</td>
</tr>
<tr>
<td>Length of the TCP segment</td>
</tr>
</tbody>
</table>

Prof. Dr. Mesut Güneş • cst.mi.fu-berlin.de • Telematics • Chapter 7: Transport Layer
Transmission Control Protocol (TCP)
Connection Management
TCP Connection Management:
1. Connection Establishment

The server waits for connection requests using LISTEN and ACCEPT.

The client uses the CONNECT operation by indicating IP address, port number, and the acceptable maximum segment size (MSS).

CONNECT sends a SYN.

If the destination port of the CONNECT is identical to the port number on which the server waits, the connection is accepted, otherwise it is rejected with RST.

The server also sends a SYN to the client and acknowledges at the same time the receipt of the client’s SYN segment.

The client sends an acknowledgement for the SYN segment of the server. The connection is established.

Three Way Handshake
TCP Connection Management: Irregular Connection Establishment

- Two computers at the same time try to establish a connection between the same sockets.
- Connections are characterized by their endpoints; only one connection is established between a pair of endpoints.
- The endpoints are uniquely characterized by:

\[(\text{IP Address}_1, \text{Port}_1, \text{IP Address}_2, \text{Port}_2)\]
TCP Connection Management: 2. Data Transmission

- Full-duplex connection
- Segmentation of a byte stream into segments.
  - Usual sizes are 1500, 536, or 512 byte; thus IP fragmentation is avoided.
  - All hosts have to accept TCP segments of 536 byte + 20 byte = 556 byte

- Usual acknowledgement mechanism:
  - All segments up to ACK-1 are confirmed. If the sender has a timeout before an ACK, he repeats the sending.

- Usual procedure for repeating:
  - Go-Back-N or Selective Repeat
3. Connection Termination

- Termination as two simplex connections
- Send a FIN segment
- If the FIN segment is confirmed, the direction is “switched off”. The opposite direction remains however still open, data can be still further sent.
  - Half-open connections!
- Use of timers to protect against packet loss.
TCP Connection Management

TCP client lifecycle

TCP server lifecycle

CLOSED

SYN_SENT

FIN_WAIT_1

FIN_WAIT_2

TIME_WAIT

wait 30 seconds

client application initiates a TCP connection

send SYN

receive SYN & ACK send ACK

receive FIN send ACK

client application initiates close connection

send FIN

receive ACK send nothing

server application creates a listen socket

CLOSED

LISTEN

SYN_RCVD

CLOSE_WAIT

LAST_ACK

ESTABLISHED

Prof. Dr. Mesut Güneş • cst.mi.fu-berlin.de • Telematics • Chapter 7: Transport Layer
The Entire TCP Connection

Normal path of server

Unusual events

Event/action pair
- Event: System call by user, arrival of segment, timeout
- Action: Send a control segment

Prof. Dr. Mesut Güneş • cst.mi.fu-berlin.de • Telematics • Chapter 7: Transport Layer
## States during a TCP Session

<table>
<thead>
<tr>
<th>State</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CLOSED</td>
<td>No active communications</td>
</tr>
<tr>
<td>LISTEN</td>
<td>The server waits for a connection request</td>
</tr>
<tr>
<td>SYN RCVD</td>
<td>A connection request was received and processed, wait for the last ACK of the connection establishment</td>
</tr>
<tr>
<td>SYN SENT</td>
<td>Application began to open a connection</td>
</tr>
<tr>
<td>ESTABLISHED</td>
<td>Connection established, transmit data</td>
</tr>
<tr>
<td>FIN WAIT 1</td>
<td>Application starts a connection termination</td>
</tr>
<tr>
<td>FIN WAIT 2</td>
<td>The other side confirms the connection termination</td>
</tr>
<tr>
<td>TIME WAIT</td>
<td>Wait for late packets</td>
</tr>
<tr>
<td>CLOSING</td>
<td>Connection termination</td>
</tr>
<tr>
<td>CLOSE WAIT</td>
<td>The other side initiates a connection termination</td>
</tr>
<tr>
<td>LAST ACK</td>
<td>Wait for late packets</td>
</tr>
</tbody>
</table>
Transmission Control Protocol (TCP)
Timer Management
Timer Management with TCP

- TCP uses several timers
- **Retransmission timer** (for repeating a transmission)
  - But: how to select the timer value?
  - Probability density of the time till an acknowledgement arrives:

![Graph showing probability density of round trip time](image)

**Problem on the transport layer:**
- $T_1$ is too small: too many retransmissions
- $T_2$ is too large: inefficient for actual packet loss
Timer Management with TCP: Example RTT estimation

RTT: gaia.cs.umass.edu to fantasia.eurecom.fr

SampleRTT
Estimated RTT

Rounded Average RTTs:
350
300
250
200
150
100

time (seconds)

1 8 15 22 29 36 43 50 57 64 71 78 85 92 99 106

Estimated RTT
Timer Management with TCP: Example RTT estimation

Prof. Dr. Mesut Güneş • cst.mi.fu-berlin.de • Telematics • Chapter 7: Transport Layer
Timer Management with TCP

● How to set TCP timeout value?
  ● Longer than RTT
    ● But RTT varies
  ● Too short: premature timeout
    ● Unnecessary retransmissions
  ● Too long: slow reaction to segment loss

● How to estimate RTT?
  ● SampleRTT: measured time from segment transmission until ACK receipt
    ● Ignore retransmissions
  ● SampleRTT will vary, want estimated RTT “smoother”
    ● Average several recent measurements, not just current SampleRTT
Timer Management with TCP: Retransmission Timer

- Solution: dynamic algorithm, which adapts the timer by current measurements of the network performance.
- **Algorithm of Jacobson (1988)**
  - TCP manages a variable \( RTT \) (Round Trip Time) for each connection
  - \( RTT \) is momentarily the best *estimation* of the round trip time
  - When sending a segment, a timer is started which measures the time the acknowledgement needs and initiates a retransmission if necessary.
  - If the acknowledgement arrives before expiration of the timer (after a time unit \( \text{sampleRTT} \)), \( RTT \) is updated:
    \[
    RTT = \alpha \times RTT + (1 - \alpha) \times \text{sampleRTT}
    \]
    \( \alpha \) is a smoothing factor, typically 0.875
  - The choice of the timeout is based on \( RTT \)
    \[
    \text{Timeout} = \beta \times RTT
    \]
    At the beginning, \( \beta \) was chosen as 2, but this was too inflexible
Timer Management with TCP

Prof. Dr. Mesut Güneş • cst.mi.fu-berlin.de • Telematics • Chapter 7: Transport Layer
Timer Management with TCP: Retransmission Timer

- **Improvement (Jacobson):** set timer proportionally to the standard deviation of the arrival time of acknowledgements
  - Computation of standard deviation:
    
    \[
    devRTT = \gamma \times devRTT + (1 - \gamma) \times |RTT - sampleRTT|
    \]
  
  - Typically \( \gamma = 0.75 \)

- Standard timeout interval:
  
  \[
  Timeout = RTT + 4 \times devRTT
  \]

  - The factor 4 was determined on the one hand by trying out, on the other hand because it is fast and simple to use in computations.
Timer Management with TCP
Timer Management with TCP: Retransmission Timer

- Karn’s Algorithm
  - Very simple proposal, which is used in most TCP implementations (optional)
  - Do not update $RTT$ on any segments that have been **retransmitted**.
    - The timeout is doubled on each failure until the segments get through.
### Persistence timer
- Prevents a deadlock with a loss of the buffer release message of a receiver.
- With expiration of the timer, the sender transfers a test segment. The response to this transmission contains the current buffer size of the receiver. If it is still zero, the timer is started again.

### Keep-alive timer
- If a connection is inactive for a longer time, at expiration of the timer it is examined whether the other side is still living.
- If no response is given, the connection is terminated.
- Disputed function, not often implemented.

### Time Wait timer
- During the termination of a connection, the timer runs for the double packet life time to be sure that no more late packets arrive.
Transmission Control Protocol (TCP)
Reliable Data Transfer
TCP Reliable Data Transfer

- Reliable transfer with TCP
  - TCP creates reliable data transfer service on top of IP’s unreliable service
  - Pipelined segments
  - Cumulative acks
  - TCP uses single retransmission timer

- Retransmissions are triggered by:
  - Timeout events
  - Duplicate acks

- Initially consider simplified TCP sender:
  - Ignore duplicate acks
  - Ignore flow control, congestion control
TCP Sender Events

● Data received from app:
  ● Create segment with sequence number
  ● Sequence number is byte-stream number of first data byte in segment
  ● Start timer if not already running (think of timer as for oldest unacked segment)
  ● Expiration interval: TimeOutInterval

● Timeout:
  ● Retransmit segment that caused timeout
  ● Restart timer

● Ack received:
  ● If acknowledges previously unacked segments
    ● Update what is known to be acked
    ● Start timer if there are outstanding segments
TCP Sender (simplified)

NextSeqNum = InitialSeqNum
SendBase = InitialSeqNum

loop (forever) {
  switch(event)
  
  event: data received from application above
  create TCP segment with sequence number NextSeqNum
  if (timer currently not running)
    start timer
  pass segment to IP
  NextSeqNum = NextSeqNum + length(data)

  event: timer timeout
  retransmit not-yet-acknowledged segment with smallest sequence number
  start timer

  event: ACK received, with ACK field value of y
  if (y > SendBase) {
    SendBase = y
    if (there are currently not-yet-acknowledged segments)
      start timer
  }
} /* end of loop forever */

Comment:
• SendBase-1: last cumulatively ack’ed byte
Example:
• SendBase-1 = 71; y= 73, so the rcvr wants 73+; y > SendBase, so that new data is acked
TCP Retransmission Scenarios

Lost ACK scenario

Premature timeout

Host A

Host B

SendBase = 100

Seq=92, 8 bytes data

ACK=100

Time

Timeout

X

loss

SendBase = 100

Seq=92, 8 bytes data

ACK=100

SendBase = 120

Seq=92, 8 bytes data

ACK=100

Seq=100, 20 bytes data

ACK=120

SendBase = 120

Seq=92, 8 bytes data

ACK=120

Seq=92 timeout

Time

Prof. Dr. Mesut Güneş • cst.mi.fu-berlin.de • Telematics • Chapter 7: Transport Layer
TCP Retransmission Scenarios

Host A

SendBase = 120

Host B

Seq=92, 8 bytes data
Seq=100, 20 bytes data

ACK=100

Timeout

X

loss

ACK=120

Time

Cumulative ACK scenario

SendBase = 120

Seq=92, 8 bytes data
Seq=100, 20 bytes data

ACK=100

Timeout

X

loss

ACK=120

Time

Cumulative ACK scenario
### TCP ACK generation [RFC 1122, RFC 2581]

<table>
<thead>
<tr>
<th>Event at Receiver</th>
<th>TCP Receiver Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arrival of in-order segment with expected seq #. All data up to expected seq # already ACKed</td>
<td>Delayed ACK. Wait up to 500ms for next segment. If no next segment, send ACK</td>
</tr>
<tr>
<td>Arrival of in-order segment with expected seq #. One other segment has ACK pending</td>
<td>Immediately send single cumulative ACK, ACKing both in-order segments</td>
</tr>
<tr>
<td>Arrival of out-of-order segment higher-than-expect seq. #. Gap detected</td>
<td>Immediately send duplicate ACK, indicating seq. # of next expected byte</td>
</tr>
<tr>
<td>Arrival of segment that partially or completely fills gap</td>
<td>Immediately send ACK, provided that segment starts at lower end of gap</td>
</tr>
</tbody>
</table>
Transmission Control Protocol (TCP)

Flow Control
TCP Flow Control: Sliding Window

- To provide reliable data transfer, as on Layer 2, a sliding window mechanism is used.
- Differences:
  - Sender sends **bytes** according to the window size
  - Window is shifted by **n** byte as soon as an ACK for **n** bytes arrives
  - Exception: Urgent data (URGENT flag is set) are sent immediately
  - Characteristic: the window size can be changed during the transmission phase

```
Initial window
```

```
Segment 1, 2, and 3 acknowledged
```

```
Window slides
```

TCP Flow Control: Sliding Window, Example

- **Application writes 2 KB**
  - Sender: SEQ = 0
  - Receiver: ACK = 2048, WIN = 2048

- **Application writes 2 KB**
  - Sender: SEQ = 2048
  - Receiver: ACK = 4096, WIN = 0

- **Sender is blocked**
  - Sender: ACK = 4096, WIN = 2048
  - Receiver: SEQ = 4096

- **Sender can transfer up to 2 KB**
  - Sender: SEQ = 4096
  - Receiver: ACK = 4096, WIN = 2048

**Receiver buffer**
- Empty
- Full
- Application reads 2 KB
  - 2K
  - 1K 2K

Prof. Dr. Mesut Güneş • cst.mi.fu-berlin.de • Telematics • Chapter 7: Transport Layer
“Silly Window” Syndrome

Solution of Clark:
The receiver must wait with the next window actualization until the receiver buffer again is reasonably empty.
The whole TCP Session

Client

SYN, ACK/SEQ=2000/ACK=3000/Window=4200/MSS=1400

DATA 1400 byte/SEQ=3000/ACK=2001/Window=2500

DATA 1400 byte/SEQ=4400/ACK=2001/Window=2500

DATA 1400 byte/SEQ=5800/ACK=2001/Window=2500

ACK/SEQ=2001/ACK=4400/Window=4200

ACK/SEQ=2001/ACK=5800/Window=4200

Server

ACK/SEQ=3000/ACK=2001/Window=2500

ACK/SEQ=2001/ACK=7200/Window=4200

DATA 1400 byte/SEQ=7200/ACK=2001/Window=2500

DATA 1400 byte/SEQ=8600/ACK=2001/Window=2500

ACK/SEQ=2001/ACK=8600/Window=4200

ACK/SEQ=2001/ACK=10000/Window=4200

FIN, ACK/SEQ=10001

FIN, ACK/SEQ=2001/ACK=10001

ACK/SEQ=10001/ACK=2002

Prof. Dr. Mesut Güneş • cst.mi.fu-berlin.de • Telematics • Chapter 7: Transport Layer
Flow Control: Network Bottlenecks

Assumption:
Packet loss is rarely because of transmission errors, rather because of overload situations.

Necessary:
Congestion Control

Capacity of the receiver:
Flow Control Window

Capacity of the network:
Congestion Window

Prof. Dr. Mesut Güneş • cst.mi.fu-berlin.de • Telematics • Chapter 7: Transport Layer
Principles of Congestion Control
Principles of Congestion Control

• Congestion:
  • Informally: “too many sources sending too much data too fast for network to handle”
  • Different from flow control!
  • Manifestations:
    • Lost packets (buffer overflow at routers)
    • Long delays (queueing in router buffers)
  • A top-10 problem!
Causes/costs of congestion: Scenario 1

- Two senders, two receivers
- One router, infinite buffers
- No retransmission

Host A
\[ \lambda_{in} : \text{original data} \]

Host B

- Large delays when congested
- Maximum achievable throughput

\[ \lambda_{out} \] vs. \[ \lambda_{in} \] with congestion

\[ \text{delay} \] vs. \[ \lambda_{in} \] with congestion
Causes/costs of congestion: Scenario 2

- One router, finite buffers
- Sender retransmission of lost packet
- Offered load $\lambda'_\text{in}$: Original data + retransmitted data

![Diagram showing network traffic and buffer capacities](image)

$\lambda_\text{in}$: original data
$\lambda'_\text{in}$: original data, plus retransmitted data
$\lambda_\text{out}$

Capacity of link R
Finite shared output link buffers

Prof. Dr. Mesut Güneş • cst.mi.fu-berlin.de • Telematics • Chapter 7: Transport Layer
Causes/costs of congestion: Scenario 2

- Always: $\lambda_{\text{in}} = \lambda_{\text{out}}$ (goodput)
- "perfect" retransmission only when loss: $\lambda_{'\text{in}} = \lambda_{\text{out}}$
- Retransmission of delayed (not lost) packet makes $\lambda_{'\text{in}}$ larger (than perfect case) for same $\lambda_{\text{out}}$

- "Costs" of congestion:
  - More work (retrans) for given "goodput"
  - Unneeded retransmissions: link carries multiple copies of packet
Causes/costs of congestion: Scenario 3

- Four senders
- Multi-hop paths
- Timeout/retransmit

What happens as $\lambda_\text{in}$ and $\lambda'_\text{in}$ increase?

Diagram:
- Host A
- Host B
- $\lambda_\text{in}$: original data
- $\lambda'_\text{in}$: original data, plus retransmitted data
- Finite shared output link buffers
Causes/costs of congestion: Scenario 3

- Another “cost” of congestion:
  - When packet dropped, any “upstream transmission capacity used for that packet was wasted!
Two broad approaches towards congestion control:

- **End-end congestion control:**
  - No explicit feedback from network
  - Congestion inferred from end-system observed loss, delay
  - Approach taken by TCP

- **Network-assisted congestion control:**
  - Routers provide feedback to end systems
    - Single bit indicating congestion (SNA, DECbit, TCP/IP ECN, ATM)
  - Explicit rate sender should send at
Case study: ATM ABR congestion control

- ABR: available bit rate:
  - “elastic service”
  - If sender’s path “underloaded”:
    - Sender should use available bandwidth
  - If sender’s path congested:
    - Sender throttled to minimum guaranteed rate

- RM (resource management) cells:
  - Sent by sender, interspersed with data cells
  - Bits in RM cell set by switches (“network-assisted”)
    - NI bit: no increase in rate (mild congestion)
    - CI bit: congestion indication
  - RM cells returned to sender by receiver, with bits intact
Case study: ATM ABR congestion control

- Two-byte ER (explicit rate) field in RM cell
  - Congested switch may lower ER value in cell
  - Sender’s send rate thus maximum supportable rate on path
- EFCI bit in data cells: set to 1 in congested switch
  - If data cell preceding RM cell has EFCI set, sender sets CI bit in returned RM cell
Transmission Control Protocol (TCP)

Congestion control
TCP congestion control: additive increase, multiplicative decrease

- Approach: increase transmission rate (window size), probing for usable bandwidth, until loss occurs
  - Additive Increase: increase $\text{CongWin}$ by 1 MSS every RTT until loss detected
  - Multiplicative Decrease: cut $\text{CongWin}$ in half after loss
  - Additive Increase Multiplicative Decrease (AIMD)

Saw tooth behavior: Probing for bandwidth

![Graph showing saw tooth behavior of congestion window size over time]
TCP Congestion Control: Details

- Sender limits transmission:
  LastByteSent - LastByteAcknowledged ≤ CongWin

- Roughly:
  \[
  \text{rate} = \frac{\text{CongWin}}{\text{RTT}} \text{ Bytes/sec}
  \]

- CongWin is dynamic, function of perceived network congestion

- How does the sender perceive congestion?
  Loss event = Timeout or 3 duplicate acks

- TCP sender reduces rate (CongWin) after loss event

- Three mechanisms:
  - AIMD
  - Slow start
  - Conservative after timeout events
TCP Slow Start

- When connection begins, CongWin = 1 MSS
  - Example: MSS = 500 bytes & RTT = 200 msec
  - Initial rate = 20 kbps

- Available bandwidth may be >> MSS/RTT
  - Desirable to quickly ramp up to respectable rate

- When connection begins, increase rate exponentially fast until first loss event
TCP Slow Start

- Each sender maintains two windows for the number of bytes which can be sent:
  - Flow Control Window: granted receiver buffer
  - Congestion Window: “network granted” capacity (cwnd)
- Minimum of both windows is the maximum number of bytes that can be sent
- With connection establishment, the sender initializes the congestion window to one maximum segment size (MSS)
  - MSS is the maximum number of application data that can be send in one segment!!!
- A segment with MSS bytes of application data is sent

**Slow Start Algorithm**

- If an acknowledgement arrives before timeout, double the congestion window, otherwise reset it to the initial value. Thus a “grope” takes place up to the transmission capacity.
- Enlargement stops with reaching the flow control window
- Refinement by introduction of a threshold value ssthresh
  - At the beginning 64 Kbyte
  - Only linear enlargement by one maximum segment size (Congestion Avoidance)
  - With a timeout the threshold value is put back to half of the maximum window size reached before
TCP Slow Start

● When connection begins, increase rate exponentially until first loss event:
  ● Double CongWin every RTT
  ● Done by incrementing CongWin for every ACK received

● Summary:
  ● Initial rate is slow but ramps up exponentially fast
TCP Slow Start

Congestion Avoidance, groping to the maximum capacity

Overload assumed, reduce the data amount

Be more careful in the next attempt

Slow start, fast utilization of the free capacity

Start with one segment of MSS size

Prof. Dr. Mesut Güneş • cst.mi.fu-berlin.de • Telematics • Chapter 7: Transport Layer
Refinement: Inferring loss

- After 3 dup ACKs:
  - CongWin is cut in half
  - Window then grows linearly
- But after timeout event:
  - CongWin instead set to 1 MSS;
  - Window then grows exponentially
  - To a threshold, then grows linearly

- Philosophy
  - 3 dup ACKs indicates network capable of delivering some segments
  - Timeout indicates a “more alarming” congestion scenario
Fast Retransmit and Fast Recovery

- Slow Start is not well suited when only a single packet is lost...
  - Time-out period often relatively long ➔ Long delay before resending lost packet

- **Fast Retransmit**
  - The **receiver** sends a duplicate ACK immediately when an out-of-order segment arrives
  - When the **sender** has received 3 duplicate ACKs, it retransmits the missing segment.
    - Hopefully, the acknowledgement for the repeated segment arrives before a timeout
Fast Retransmit

Resending a segment after triple duplicate ACK
Questions

Q: When should the exponential increase switch to linear?
A: When CongWin gets to 1/2 of its value before timeout.

Implementation:
Variable Threshold
At loss event, Threshold is set to 1/2 of CongWin just before loss event
Fast Retransmit and Fast Recovery

- Fast Retransmit has to be enhanced to be useful in overload situations...

- **Fast Recovery**
  - When the third ACK is received, reduce
    \[ \text{ssthresh} = \max(\text{ssthresh}/2, 2 \times \text{MSS}) \]
  - Retransmit the missing segment, set
    \[ \text{cwnd} = \text{ssthresh} + 3 \times \text{MSS} \]
  - For each more duplicated ACK, increment \text{cwnd} by \text{MSS}
  - This reduces \text{cwnd} by the amount of lost segments to adapt to the network situation
  - If the new \text{cwnd} (and the receiver’s buffer) allows, send a segment
  - When the next (normal) ACK arrives, set \text{cwnd} to \text{ssthresh} to go on normally
Fast Retransmit and Fast Recovery

Chapter 7: Transport Layer
Summary: TCP Congestion Control

- When CongWin is below Threshold, sender in **slow-start phase**, window grows exponentially.
- When CongWin is above Threshold, sender is in **congestion-avoidance phase**, window grows linearly.
- When a **triple duplicate ACK** occurs, Threshold set to CongWin/2 and CongWin set to Threshold.
- When **timeout** occurs, Threshold set to CongWin/2 and CongWin is set to 1 MSS.
## TCP sender congestion control

<table>
<thead>
<tr>
<th>State</th>
<th>Event</th>
<th>TCP Sender Action</th>
<th>Commentary</th>
</tr>
</thead>
<tbody>
<tr>
<td>Slow Start (SS)</td>
<td>ACK receipt for previously unacked data</td>
<td>CongWin = CongWin + MSS, If (CongWin &gt; Threshold) set state to “Congestion Avoidance”</td>
<td>Resulting in a doubling of CongWin every RTT</td>
</tr>
<tr>
<td>Congestion Avoidance (CA)</td>
<td>ACK receipt for previously unacked data</td>
<td>CongWin = CongWin+MSS * (MSS/CongWin)</td>
<td>Additive increase, resulting in increase of CongWin by 1 MSS every RTT</td>
</tr>
<tr>
<td>SS or CA</td>
<td>Loss event detected by triple duplicate ACK</td>
<td>Threshold = CongWin/2, CongWin = Threshold, Set state to “Congestion Avoidance”</td>
<td>Fast recovery, implementing multiplicative decrease. CongWin will not drop below 1 MSS.</td>
</tr>
<tr>
<td>SS or CA</td>
<td>Timeout</td>
<td>Threshold = CongWin/2, CongWin = 1 MSS, Set state to “Slow Start”</td>
<td>Enter slow start</td>
</tr>
<tr>
<td>SS or CA</td>
<td>Duplicate ACK</td>
<td>Increment duplicate ACK count for segment being acked</td>
<td>CongWin and Threshold not changed</td>
</tr>
</tbody>
</table>

Prof. Dr. Mesut Güneş • cst.mi.fu-berlin.de • Telematics • Chapter 7: Transport Layer
Transmission Control Protocol (TCP)

TCP Throughput
TCP throughput

- What’s the average throughput of TCP as a function of window size and RTT?
  - Ignore slow start
- Let $W$ be the window size when loss occurs.
- When window is $W$, throughput is $W/RTT$
- Just after loss, window drops to $W/2$, throughput to $W/2RTT$.
- Average throughout: $(W/RTT + W/2RTT)/2 \Rightarrow 0.75W/RTT$
TCP Futures: TCP over “long, fat pipes”

- Example: 1500 byte segments, 100ms RTT, want 10 Gbps throughput
- Requires window size $W = 83,333$ in-flight segments
- Throughput in terms of loss rate:

\[
\frac{1.22 \cdot MSS}{RTT \sqrt{L}}
\]

⇒ $L = 2 \cdot 10^{-10}$
- New versions of TCP for high-speed
Transmission Control Protocol (TCP)

Fairness
TCP Fairness

- Fairness goal: if $k$ TCP sessions share the same bottleneck link of bandwidth $R$, each should have an average rate of $R/K$. 

![Diagram showing TCP connections and bottleneck router with capacity $R$.]
Why is TCP fair?

Two competing sessions:

- Additive increase gives slope of 1, as throughout increases
- Multiplicative decrease decreases throughput proportionally
Fairness (more)

- Fairness and UDP
  - Multimedia apps often do not use TCP
    - do not want rate throttled by congestion control
  - Instead use UDP:
    - pump audio/video at constant rate, tolerate packet loss
  - Research area: TCP friendly

- Fairness and parallel TCP connections
  - nothing prevents app from opening parallel connections between 2 hosts.
  - Web browsers do this
  - Example: link of rate $R$ supporting 9 connections;
    - new app asks for 1 TCP, gets rate $R/10$
    - new app asks for 11 TCPs, gets $R/2$
Transmission Control Protocol (TCP)
TCP in Wireless Networks
TCP in Wireless Networks

- Theoretically the transport layer protocol should be independent of lower layers
  - But TCP is optimized for wired networks
  - TCP assumes that packet loss is due to congestion in the network

- In wireless networks packet loss occurs due to the medium
  - Thus, performance of TCP in wireless networks is poor
  - Many approaches to solve the performance problem

- Indirect TCP (as an example)
  - The end-to-end connection is broken in two parts
Transmission Control Protocol (TCP)
TCP and Security
TCP and Security: Syn-Flood

- During connection establishment the client does not finish the Three Way Handshake procedure
  - A half open connection
  - Operating system reserves resources
TCP and Security: Syn-Flood

- **Countermeasure: Syn cookies**
  - Server does not create a half-open connection
  - Server computes an initial sequence number $y$ based on a hash function
    - This is the cookie
  - When the client returns with ACK, the server recomputes the hash function and checks it
    - For legitimate connection, the check will be successful

\[
\text{Client} \quad \rightarrow \quad \text{Server} \quad \\
\text{SYN, SEQ}=x \quad \rightarrow \quad \text{y}=h() \quad \text{ACK}=x+1
\]

\[
\text{SYN, SEQ}=y, \text{ACK}=x+1 \quad \rightarrow \quad \text{ACK}=y+1, \text{SEQ}=x+1
\]
Some Tools
Some Tools / netstat

- **netstat**: Displays protocol statistics and TCP/IP network connections
  - **Flags**
    - **n**: display IP addresses
    - **b**: display executable
    - **r**: routing table

```
x:\>netstat -n
```

Active Connections

<table>
<thead>
<tr>
<th>Proto</th>
<th>Local Address</th>
<th>Foreign Address</th>
<th>State</th>
</tr>
</thead>
<tbody>
<tr>
<td>TCP</td>
<td>127.0.0.1:3055</td>
<td>127.0.0.1:3056</td>
<td>ESTABLISHED</td>
</tr>
<tr>
<td>TCP</td>
<td>127.0.0.1:3056</td>
<td>127.0.0.1:3055</td>
<td>ESTABLISHED</td>
</tr>
<tr>
<td>TCP</td>
<td>160.45.114.21:2114</td>
<td>130.133.8.114:80</td>
<td>CLOSE_WAIT</td>
</tr>
<tr>
<td>TCP</td>
<td>160.45.114.21:3027</td>
<td>160.45.113.73:1025</td>
<td>ESTABLISHED</td>
</tr>
<tr>
<td>TCP</td>
<td>160.45.114.21:3029</td>
<td>160.45.113.73:1025</td>
<td>ESTABLISHED</td>
</tr>
<tr>
<td>TCP</td>
<td>160.45.114.21:3043</td>
<td>160.45.113.89:1200</td>
<td>ESTABLISHED</td>
</tr>
<tr>
<td>TCP</td>
<td>160.45.114.21:3362</td>
<td>207.46.108.69:1863</td>
<td>ESTABLISHED</td>
</tr>
<tr>
<td>TCP</td>
<td>160.45.114.21:3704</td>
<td>130.133.8.114:80</td>
<td>CLOSE_WAIT</td>
</tr>
<tr>
<td>TCP</td>
<td>160.45.114.21:3705</td>
<td>130.133.8.114:80</td>
<td>CLOSE_WAIT</td>
</tr>
<tr>
<td>TCP</td>
<td>160.45.114.21:3907</td>
<td>160.45.114.28:139</td>
<td>ESTABLISHED</td>
</tr>
<tr>
<td>TCP</td>
<td>160.45.114.21:3916</td>
<td>160.45.113.100:445</td>
<td>ESTABLISHED</td>
</tr>
</tbody>
</table>
Some Tools

- TTCP – Test TCP
  - A tool to measure throughput in a network via TCP and UDP

- Iperf
  - Similar tool to test network throughput
Summary

● The TCP/IP reference model has only two protocols on the transport layer
  ● UDP for connectionless, but lightweight protocol
  ● TCP for connection oriented and reliable communication
    ● Connection establishment
    ● Flow control / congestion control
    ● Connection termination
    ● Fairness
● Some tools
  ● netstat
  ● iperf
  ● ttcp