TCP: Transmission Control Protocol
Part I : Protocol basics

Surasak Sanguanpong
nguan@ku.ac.th
http://www.cpe.ku.ac.th/~nguan
Last updated: July 30, 1999
Agenda

- Services provided by TCP
- TCP format
- How TCP reliability is achieved
- Sliding window
- TCP Connection
- TCP State
TCP encapsulation

- with Ethernet frame

<table>
<thead>
<tr>
<th>Ethernet hdr</th>
<th>IP header</th>
<th>TCP header</th>
<th>data</th>
</tr>
</thead>
</table>

segment
TCP & UDP Services

- TCP: Transmission Control Protocol
  - RFC 793
  - connection-oriented service
  - full duplex
  - reliable service by adding more overhead to manage acknowledgment, flow control, timer
TCP: Transmission Control Protocol

- TCP performs typical transport layer functions:
  - passed data to relevant application-level services
  - mux and demux data from applications to and from IP layer
  - error recovery
  - flow control data stream (avoid buffer overflow)
TCP properties

- byte stream with full duplex transferring
- adaptive to LAN/WAN
- congestion avoidance and control
TCP data stream

- TCP provides a full duplex service that simultaneously manages two streams of data.
- Stream of octets passed between sender/receiver.
Ports

- Port - a 16 bit address allocated for the most common application layer services
- UDP and TCP use port addressing to deliver info to applications
- Servers are known by ports number
  - FTP 20, TELNET 23, SMTP 25, HTTP 80
- Port numbers are generally allocated by
  - 0 --not used
  - 1-255 --Reserved ports for well-known services
  - 256-1023 --Other reserved ports
  - 1024-65535 --user-defined server ports
- Unix store general used ports in /etc/services
Sockets

- **socket**: a pair of the IP address and the port number

  `<158.108.33.3, 3000>`

**Diagram:**

- IP address is unique to a node, the port is unique on a node
- the socket gives a unique identification of an application layer service
Socket address

- A connection is identified by the socket address at its two ends
  - client socket: 158.108.33.3,3000; 158.108.2.71,21
  - server socket: 158.108.2.71,21; 158.108.33.3,3000;
Socket multiple connection

- server’s unique socket address can be accessed simultaneously by clients

---

Client 1
IP: 158.108.33.2
Port: 3000

Client 2
IP: 158.108.33.3
Port: 3120

Server
IP: 158.108.2.71
Port: 21
Transmission Control Protocol

- TCP passed block of data to IP, consisting of the TCP header and application layer data, called segment
- adding reliability in TCP is achieved by
  - Error detection and correction (due to segments corrupted)
  - Flow control (prevent a transmitter overrunning a receiver owing a resource limitations)
  - Resequencing (IP can deliver datagrams in any order)
  - Removing duplicate segments (due to error-recovery mechanisms used by TCP)
How TCP handles reliability

- Using sequence numbers to identify data
- positive acknowledgments of data received in the correct sequence
- retransmission of segments which have not been acknowledged within a (variable) time limit
- Let’s see these mechanisms in TCP header
## TCP header

<table>
<thead>
<tr>
<th>Field</th>
<th>Length</th>
</tr>
</thead>
<tbody>
<tr>
<td>source port</td>
<td>16</td>
</tr>
<tr>
<td>destination port</td>
<td>16</td>
</tr>
<tr>
<td>sequence number</td>
<td>32</td>
</tr>
<tr>
<td>acknowledgment number</td>
<td>32</td>
</tr>
<tr>
<td>data offs</td>
<td>4</td>
</tr>
<tr>
<td>resv</td>
<td>6</td>
</tr>
<tr>
<td>flag</td>
<td>6</td>
</tr>
<tr>
<td>window size</td>
<td>16</td>
</tr>
<tr>
<td>checksum</td>
<td>16</td>
</tr>
<tr>
<td>urgent pointer</td>
<td>16</td>
</tr>
<tr>
<td>options and padding</td>
<td></td>
</tr>
</tbody>
</table>
TCP header details (I)

- **source, destination port:16,16** - identify applications at ends of the connection
- **sequence:32** - indicates 1\textsuperscript{st} data octet in this segment
- **acknowledgment:32** - next expected sequence number, valid only when the ACK bit (reside in flag) is set
TCP header details (II)

- **data offset**: 4 - 32 bit words offset tells the receiver where user data begins
- **reserved**: 6 - not used
- **flag**: 6
  - URG: validity of urgent pointer field
  - ACK: validity of acknowledge field
  - PSH: push request (pass segment to appl layer immediately)
  - RST: reset the connection
  - SYN: initial synchronization
  - FIN: sender at end of byte stream
TCP header details (III)

- **window:16** - advertise amount of buffer space this node has allocated
- **checksum:16** - 16 bits 1’s complement of pseudo header, TCP header and data
- **urgent pointer:16** - byte position of data that should be processed first
- **options** - variable length option e.g. MSS (max segment size) tells destination node
Sliding window principle

- send and wait for acknowledgment
- no ACK within a certain time, retransmit the packet
- use for flow control:
  - prevent sender from overloading receiver with data, e.g. high-performance server to slow PC
  - congestion inside network, e.g. router performance, slow link speed
- How to provide flow control?
  - set the appropriate size of sliding window size
Sliding window flow control

- Receiver “advertises” its window size in acknowledgments.
- Sender will adjust its allowed to send pointer as receiver’s advertisement.

_no! no! I can load only 200 kg._
Sliding window: small window size

- 1 byte window size utilizes efficiency of channel in half (half-duplex transmission)
- why not send many packets and get back cumulative ACK?
Sliding window: larger window size

- A larger window size allows more data to be transmitted pending acknowledgment
- Window size specifies how many bytes the receiver is willing to accept
Sequence number in segment

- Data continuously sent more than segment
- Need not to wait for acknowledgment every segment

I received all up to 799!
Retransmit a loss segment

host A

\[
\begin{align*}
data \text{ bytes } & \ 500-599 \\
data \text{ bytes } & \ 600-699 \\
data \text{ bytes } & \ 700-799
\end{align*}
\]

\[\text{ACK 700}\]

host B

\[
\begin{align*}
data \text{ bytes } & \ 700-799
\end{align*}
\]
Error recovery (I)

- receiver has to send ACK with sequence number
- sender reset timer when receives ACK
Error recovery (II)

- on time out, sender will retransmit the segment
- this mechanism is used for error recovery
Sliding window buffer

- sender groups its packet to be transmitted with window indication
Sliding window example

- movement of the right and left edges of the window

<table>
<thead>
<tr>
<th></th>
<th>....999</th>
<th>1000...</th>
<th>1099</th>
<th>1100...</th>
<th>1199</th>
<th>1200...</th>
</tr>
</thead>
<tbody>
<tr>
<td>initial</td>
<td>....999</td>
<td>1000...</td>
<td>1099</td>
<td>1100...</td>
<td>1199</td>
<td>1200...</td>
</tr>
<tr>
<td></td>
<td>SndUna</td>
<td>SndNxt</td>
<td></td>
<td>SndUna+SndWnd</td>
<td></td>
<td></td>
</tr>
<tr>
<td>send 100 bytes</td>
<td>....999</td>
<td>1000...</td>
<td>1099</td>
<td>1100...</td>
<td>1199</td>
<td>1200...</td>
</tr>
<tr>
<td></td>
<td>SndUna</td>
<td>SndNxt</td>
<td></td>
<td>SndUna+SndWnd</td>
<td></td>
<td></td>
</tr>
<tr>
<td>more 100 bytes</td>
<td>....999</td>
<td>1000...</td>
<td>1099</td>
<td>1100...</td>
<td>1199</td>
<td>1200...</td>
</tr>
<tr>
<td></td>
<td>SndUna</td>
<td>SndNxt</td>
<td></td>
<td>SndNxt,SndUna+SndWnd</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ACK 100 bytes</td>
<td>....999</td>
<td>1000...</td>
<td>1099</td>
<td>1100...</td>
<td>1199</td>
<td>1200</td>
</tr>
<tr>
<td></td>
<td>SndUna</td>
<td>SndNxt</td>
<td></td>
<td>SndUna+SndWnd</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
TCP in actions

- before data could be transferred, a connection must be opened
  - servers do passive open (listen)
  - clients do active open (connect)
- when it finished, the connection is closed
- TCP has general 3 phases
  - connection setup phase
  - data phase
  - connection close phase
TCP connection establishment

- TCP uses **3-way handshake** to establish a connection
  - exchange the sequence number
  - ensures that both ends are ready and sync sequence number
Transfer phase

- Simple example with terminal connection such as Telnet. Host echoes back each received character.
TCP Connection close

- use FIN flag to close connection
Open/Close mechanisms

- **Half open** - one end has closed, aborted without the knowledge of the other end
  - host may be crashed, power off
  - no detection if no data transfer
  - reset segment (RST bit) is sent when detected
- **Half close** - one end of connection terminated its output, but still receiving data from the other end
- **Simultaneous open** - both end perform an active open to each other
- **Simultaneous close** - both end perform an active open to each other
TCP state diagram: open

- **CLOSED**
  - Appl passive opens; send nothing

- **LISTEN**
  - Passive open
  - Appl active opens; send SYN

- **SYN_RCVD**
  - Got SYN; send SYN-ACK
  - Got RST

- **SYN_SENT**
  - Active open
  - Got SYN; send SYN, ACK
  - Got ACK; send ACK

- **ESTABLISHED**
  - Got SYN; send SYN-ACK
  - Got ACK; send nothing
  - Appl closes; send FIN

- **(normal) client transition**
- **(normal) server transition**
TCP state diagram: close

**ESTABLISHED**
- got FIN; send ACK
- appl close; send FIN

**FIN_WAIT_1**
- got FIN; send ACK
- got ACK; send FIN
- send nothing

**TIME_WAIT**
- 2MSL time-out

**FIN_WAIT_2**
- got FIN; send ACK
- got ACK; send FIN
- send nothing

**CLOSING**
- got FIN; send ACK

**CLOSE_WAIT**
- appl closes; send FIN

**LAST_ACK**
- got ACK; send nothing

**passive close**

**active close**

(back to CLOSED)

(normal) client transition

(normal) server transition