Data Communication & Networks
G22.2262-001

Session 9 - Main Theme
The Internet Transport Protocols: TCP, UDP

Dr. Jean-Claude Franchitti

New York University
Computer Science Department
Courant Institute of Mathematical Sciences
Agenda

- Internet Transport Protocols
- Transport Layer Addressing
- Standard Services and Port Numbers
- TCP Overview
- Reliability in an Unreliable World
- TCP Flow Control
- Why Startup / Shutdown Difficult?
- TCP Connection Management
- Timing Problem
- Implementation Policy Options
- UDP: User Datagram Protocol
- Conclusion
Part I

Internet Transport Protocols
Internet Transport Protocols

- Two Transport Protocols Available
  - Transmission Control Protocol (TCP)
    - connection oriented
    - most applications use TCP
    - RFC 793
  - User Datagram Protocol (UDP)
    - Connectionless
    - RFC 768
Part II

Transport Layer Addressing
Transport Layer Addressing

- Communications endpoint addressed by:
  - IP address (32 bit) in IP Header
  - Port number (16 bit) in TP Header\(^1\)
  - Transport protocol (TCP or UDP) in IP Header

\(^1\) TP => Transport Protocol (UDP or TCP)
Part III

Standard Services and Port Numbers
# Standards Services and Port Numbers

<table>
<thead>
<tr>
<th>Service</th>
<th>TCP</th>
<th>UDP</th>
</tr>
</thead>
<tbody>
<tr>
<td>echo</td>
<td>7</td>
<td>7</td>
</tr>
<tr>
<td>daytime</td>
<td>13</td>
<td>13</td>
</tr>
<tr>
<td>netstat</td>
<td>15</td>
<td></td>
</tr>
<tr>
<td>ftp-data</td>
<td>20</td>
<td></td>
</tr>
<tr>
<td>ftp</td>
<td>21</td>
<td></td>
</tr>
<tr>
<td>telnet</td>
<td>23</td>
<td></td>
</tr>
<tr>
<td>smtp</td>
<td>25</td>
<td></td>
</tr>
<tr>
<td>time</td>
<td>37</td>
<td>37</td>
</tr>
<tr>
<td>domain</td>
<td>53</td>
<td>53</td>
</tr>
<tr>
<td>finger</td>
<td>79</td>
<td></td>
</tr>
<tr>
<td>http</td>
<td>80</td>
<td></td>
</tr>
<tr>
<td>pop-2</td>
<td>109</td>
<td></td>
</tr>
<tr>
<td>pop</td>
<td>110</td>
<td></td>
</tr>
<tr>
<td>sunrpc</td>
<td>111</td>
<td>111</td>
</tr>
<tr>
<td>uucp-path</td>
<td>117</td>
<td></td>
</tr>
<tr>
<td>nntp</td>
<td>119</td>
<td></td>
</tr>
<tr>
<td>talk</td>
<td></td>
<td>517</td>
</tr>
</tbody>
</table>
Part IV

TCP: Overview
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
TCP Header
## TCP Segment Structure

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>source port #</td>
<td>Source port number</td>
</tr>
<tr>
<td>dest port #</td>
<td>Destination port number</td>
</tr>
<tr>
<td>sequence number</td>
<td>Sequence number</td>
</tr>
<tr>
<td>acknowledgement number</td>
<td>Acknowledgement number</td>
</tr>
<tr>
<td>rcvr window size</td>
<td>Receiver window size</td>
</tr>
<tr>
<td>ptr urgent data</td>
<td>Pointer to urgent data</td>
</tr>
<tr>
<td>Options (variable length)</td>
<td>Options area (variable length)</td>
</tr>
<tr>
<td>application data</td>
<td>Application data (variable length)</td>
</tr>
</tbody>
</table>

### Flags
- **URG**: URGent data
- **ACK**: ACKnowledgement
- **PSH**: push data now
- **RST**: Reset connection
- **SYN**: synchronize sequence numbers
- **FIN**: connection termination

### Other Fields
- **head**
- **len**
- **not used**
- **Internet checksum** (as in UDP)

The receiver window size is counted by bytes of data (not segments!).

The receiver willing to accept is # bytes.
Part V

Reliability in an Unreliable World
Reliability in an Unreliable World

- IP offers best-effort (unreliable) delivery
- TCP uses IP
- TCP provides completely reliable transfer
- How is this possible? How can TCP realize:
  - Reliable connection startup?
  - Reliable data transmission?
  - Graceful connection shutdown?
Reliability Data Transmission

- **Positive acknowledgment**
  - Receiver returns short message when data arrives
  - Called *acknowledgment*

- **Retransmission**
  - Sender starts timer whenever message is transmitted
  - If timer expires before acknowledgment arrives, sender *retransmits* message
  - THIS IS NOT A TRIVIAL PROBLEM! – more on this later
Part VI

TCP Flow Control
TCP Flow Control

- **Receiver**
  - Advertises available buffer space
  - Called *window*
  - This is known as a *CREDIT* policy

- **Sender**
  - Can send up to entire window before ACK arrives
  - Each acknowledgment carries new window information
    - Called *window advertisement*
    - Can be zero (called *closed window*)
  - Interpretation: I have received up through X, and can take Y more octets
Credit Scheme

- Decouples flow control from ACK
  - May ACK without granting credit and vice versa
- Each octet has sequence number
- Each transport segment has seq number, ack number and window size in header
Use of Header Fields

- When sending, seq number is that of first octet in segment
- ACK includes AN=i, W=j
- All octets through SN=i-1 acknowledged
  - Next expected octet is i
- Permission to send additional window of W=j octets
  - i.e. octets through i+j-1
Credit Allocation

Transport Entity A

- ...1000 1001 2400 2401...
  A may send 1400 octets

- ...1000 1001 1601 2401...
  A shrinks its transmit window with each transmission

- ...1000 1001 2001 2401...

- ...1600 1601 2001 2601...
  A adjusts its window with each credit

- ...1600 1601 2600 2601...
  A exhausts its credit

- ...2600 2601 4000 4001...
  A receives new credit

Transport Entity B

- ...1000 1001 2400 2401...
  B is prepared to receive 1400 octets, beginning with 1001

- ...1600 1601 2601...
  B acknowledges 3 segments (600 octets), but is only prepared to receive 200 additional octets beyond the original budget (i.e., B will accept octets 1601 through 2600)

- ...1600 1601 2001 2601...

- ...1600 1601 2600 2601...
  B acknowledges 5 segments (1000 octets) and restores the original amount of credit
TCP Flow Control

**flow control**

sender won’t overrun receiver’s buffers by transmitting too much, too fast

receiver: explicitly informs sender of (dynamically changing) amount of free buffer space

- **RcvWindow field** in TCP segment

sender: keeps the amount of transmitted, unACKed data less than most recently received **RcvWindow**

**RcvBuffer** = size of TCP Receive Buffer

**RcvWindow** = amount of spare room in Buffer

receiver buffering
TCP Seq. #’s and ACKs

**Seq. #’s:**
- byte stream “number” of first byte in segment’s data

**ACKs:**
- seq # of next byte expected from other side
- cumulative ACK

**Q:** how receiver handles out-of-order segments
- A: TCP spec doesn’t say, - up to implementor
## TCP ACK Generation

**[RFC 1122, RFC 2581]**

<table>
<thead>
<tr>
<th>Event</th>
<th>TCP Receiver action</th>
</tr>
</thead>
<tbody>
<tr>
<td>in-order segment arrival, no gaps, everything else already ACKed</td>
<td>delayed ACK. Wait up to 500ms for next segment. If no next segment, send ACK</td>
</tr>
<tr>
<td>in-order segment arrival, no gaps, one delayed ACK pending</td>
<td>immediately send single cumulative ACK</td>
</tr>
<tr>
<td>out-of-order segment arrival higher-than-expect seq. # gap detected</td>
<td>send duplicate ACK, indicating seq. # of next expected byte</td>
</tr>
<tr>
<td>arrival of segment that partially or completely fills gap</td>
<td>immediate ACK if segment starts at lower end of gap</td>
</tr>
</tbody>
</table>
TCP: Retransmission Scenarios

Host A
Seq=92, 8 bytes data
ACK=100

Host B
Seq=100, 20 bytes data
ACK=100

lost ACK scenario

X

loss

Host A
Seq=92, 8 bytes data
ACK=100

Host B
Seq=100, 20 bytes data
ACK=100

premature timeout, cumulative ACKs

timeout

time

Seq=92, 8 bytes data

timeout

time

Seq=92, 8 bytes data
Part VII

Why Startup / Shutdown Difficult?
Why Startup / Shutdown Difficult?

- Segments can be
  - Lost
  - Duplicated
  - Delayed
  - Delivered out of order
  - Either side can crash
  - Either side can reboot
- Need to avoid duplicate “shutdown” message from affecting later connection
Part VIII

TCP Connection Management
TCP Connection Management

- Recall: TCP sender, receiver establish “connection” before exchanging data segments
- initialize TCP variables:
  - seq. #s
  - buffers, flow control info (e.g. RcvWindow)
- client: connection initiator
  Socket clientSocket = new Socket("hostname","port number");
- server: contacted by client
  Socket connectionSocket = welcomeSocket.accept();

Three way handshake:

Step 1: client end system sends TCP SYN control segment to server
  - specifies initial seq #

Step 2: server end system receives SYN, replies with SYNACK control segment
  - ACKs received SYN
  - allocates buffers
  - specifies server-> receiver initial seq. #
TCP Connection Management (OPEN)

Client

Opening

SYN

SYNACK

ACK

Server

Opening

Established

Closed

29
Closing a connection:

client closes socket:
```
clientSocket.close();
```

**Step 1:** client end system sends TCP FIN control segment to server

**Step 2:** server receives FIN, replies with ACK. Closes connection, sends FIN.
TCP Connection Management (cont.)

**Step 3:** client receives FIN, replies with ACK.
- Enters “timed wait” - will respond with ACK to received FINs

**Step 4:** server, receives ACK. Connection closed.

**Note:** with small modification, can handle simultaneous FINs.
TCP Connection Management (cont.)

TCP client lifecycle

TCP server lifecycle

TCP client lifecycle

TCP server lifecycle
Part IX

Timing Problem
Timing Problem!

The delay required for data to reach a destination and an acknowledgment to return depends on traffic in the internet as well as the distance to the destination. Because it allows multiple application programs to communicate with multiple destinations concurrently, TCP must handle a variety of delays that can change rapidly.

How does TCP handle this.....
Solving Timing Problem

- Keep estimate of round trip time on each connection
- Use current estimate to set retransmission timer
- Known as *adaptive retransmission*
- Key to TCP’s success
TCP Round Trip Time & Timeout

• **Q:** how to set TCP timeout value?
  • longer than RTT
    – note: RTT will vary
  • too short:
    premature timeout
    – unnecessary retransmissions
  • too long: slow reaction to segment loss

• **Q:** how to estimate RTT?
  • *SampleRTT*: measured time from segment transmission until ACK receipt
    – ignore retransmissions, cumulatively ACKed segments
  • *SampleRTT* will vary, want estimated RTT “smoother”
    – use several recent measurements, not just current *SampleRTT*
TCP Round Trip Time & Timeout

\[ \text{EstimatedRTT} = (1-x) \times \text{EstimatedRTT} + x \times \text{SampleRTT} \]

- Exponential weighted moving average
- influence of given sample decreases exponentially fast
- typical value of \(x\): 0.1

Setting the timeout

- \text{EstimatedRTT} plus “safety margin”
- large variation in \text{EstimatedRTT} \rightarrow larger safety margin

\[ \text{Timeout} = \text{EstimatedRTT} + 4 \times \text{Deviation} \]

\[ \text{Deviation} = (1-x) \times \text{Deviation} + x \times |\text{SampleRTT}-\text{EstimatedRTT}| \]
Part X

*Implementation Policy Options*
Implementation Policy Options

- Send
- Deliver
- Accept
- Retransmit
- Acknowledge
Send

- If no push or close TCP entity transmits at its own convenience (IFF send window allows!)
- Data buffered at transmit buffer
- May construct segment per data batch
- May wait for certain amount of data
Deliver (to application)

- In absence of push, deliver data at own convenience
- May deliver as each in-order segment received
- May buffer data from more than one segment
Accept

- Segments may arrive out of order
- In order
  - Only accept segments in order
  - Discard out of order segments
- In windows
  - Accept all segments within receive window
Retransmit

- TCP maintains queue of segments transmitted but not acknowledged
- TCP will retransmit if not ACKed in given time
  - First only
  - Batch
  - Individual
Acknowledgement

- Immediate
  - as soon as segment arrives.
  - will introduce extra network traffic
  - Keeps sender’s pipe open

- Cumulative
  - Wait a bit before sending ACK (called “delayed ACK”)
  - Must use timer to insure ACK is sent
  - Less network traffic
  - May let sender’s pipe fill if not timely!
Part XI

UDP: User Datagram Protocol
UDP: User Datagram Protocol [RFC 768]

• “no frills,” “bare bones” Internet transport protocol
• “best effort” service, UDP segments may be:
  – lost
  – delivered out of order to app
• connectionless:
  – no handshaking between UDP sender, receiver
  – each UDP segment handled independently of others

Why is there a UDP?
• no connection establishment (which can add delay)
• simple: no connection state at sender, receiver
• small segment header
• no congestion control: UDP can blast away as fast as desired
UDP: more

- often used for streaming multimedia apps
  - loss tolerant
  - rate sensitive
- other UDP uses
  - DNS
  - SNMP
- reliable transfer over UDP: add reliability at application layer
  - application-specific error recover!

**UDP segment format**

<table>
<thead>
<tr>
<th>source port #</th>
<th>dest port #</th>
</tr>
</thead>
<tbody>
<tr>
<td>length</td>
<td>checksum</td>
</tr>
</tbody>
</table>

Length, in bytes of UDP segment, including header

Application data (message)
UDP Uses

- Inward data collection
- Outward data dissemination
- Request-Response
- Real time application
Part XII

Conclusion
Assignment & Readings

- Assignment #9 (optional for extra credit due 12/20/12)
- Readings
  - Chapter 3 (3.5)
  - RFC 793 (introduction, sections 1 and 2)
Next Session:
Network Congestion