PART XII

RELIABLE STREAM TRANSPORT SERVICE (TCP)
Transmission Control Protocol (TCP)

- Major transport service in the TCP/IP suite
- Used for most Internet applications (esp. World Wide Web)
TCP Characteristics

- Stream orientation
- Virtual circuit connection
- Buffered transfer
- Unstructured stream
- Full duplex connection
- Reliability
Providing Reliability

- Traditional technique: Positive Acknowledgement with Retransmission (PAR)
  - Receiver sends *acknowledgement* when data arrives
  - Sender starts timer whenever transmitting
  - Sender retransmits if timer expires before acknowledgement arrives
Illustration Of Acknowledgements

- Time moves from top to bottom in the diagram
Illustration Of Recovery After Packet Loss

**Events At Sender Site**
- Send Packet 1
- Start Timer
- ACK would normally arrive at this time
- Timer Expires
- Retransmit Packet 1
- Start Timer
- Receive ACK 1
- Cancel Timer

**Network Messages**
- Packet lost

**Events At Receiver Site**
- Packet should arrive
- ACK should be sent
- Receive Packet 1
- Send ACK 1
The Problem With Simplistic PAR

A simple positive acknowledgement protocol wastes a substantial amount of network bandwidth because it must delay sending a new packet until it receives an acknowledgement for the previous packet.

- Problem is especially severe if network has long latency
Solving The Problem

- Allow multiple packets to be outstanding at any time
- Still require acknowledgements and retransmission
- Known as *sliding window*
Illustration Of Sliding Window

- Window size is fixed
- As acknowledgement arrives, window moves forward
Why Sliding Window Works

Because a well-tuned sliding window protocol keeps the network completely saturated with packets, it obtains substantially higher throughput than a simple positive acknowledgement protocol.
Illustration Of Sliding Window

Events At Sender Site | Network Messages | Events At Receiver Site
---|---|---
Send Packet 1 |  | Receive Packet 1  
Send Packet 2 | Send ACK 1  
Send Packet 3 | Receive Packet 2  
Receive ACK 1 | Send ACK 2  
Receive ACK 2 | Receive Packet 3  
Receive ACK 3 | Send ACK 3
Sliding Window Used By TCP

- Measured in byte positions
- Illustration

Bytes through 2 are acknowledged
Bytes 3 through 6 not yet acknowledged
Bytes 7 through 9 waiting to be sent
Bytes above 9 lie outside the window and cannot be sent
1. RCUK controls window size
2. RCUK sends window size in each ACK window advertising
Layering Of The Three Major Protocols

Conceptual Layering

<table>
<thead>
<tr>
<th>Application</th>
</tr>
</thead>
<tbody>
<tr>
<td>Reliable Stream (TCP)</td>
</tr>
<tr>
<td>Internet (IP)</td>
</tr>
<tr>
<td>Network Interface</td>
</tr>
</tbody>
</table>
TCP Ports, Connections, And Endpoints

- Endpoint of communication is application program
- TCP uses protocol port number to identify application
- TCP connection between two endpoints identified by four items
  - Sender’s IP address
  - Sender’s protocol port number
  - Receiver’s IP address
  - Receiver’s protocol port number
TCP Ports, Connections, And Endpoints

Connection from:
host 18.26.0.36 port 1069
to
host 128.10.2.3 port 25

Connection from:
host 128.9.0.32 port 1184
to
host 128.10.2.3 port 53

Connection from:
host 128.2.254.139 port 1184
to
host 128.10.2.3 port 53
An Important Idea About Port Numbers

Because TCP identifies a connection by a pair of endpoints, a given TCP port number can be shared by multiple connections on the same machine.
Passive And Active Opens

- Two sides of a connection
- One side waits for contact
  - A server program
  - Uses TCP’s *passive open*
- One side initiates contact
  - A client program
  - Uses TCP’s *active open*
TCP Segment Format

<table>
<thead>
<tr>
<th>Offsets</th>
<th>Octet</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
</tr>
</thead>
<tbody>
<tr>
<td>Octet</td>
<td>Bit</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>4</td>
<td>32</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>64</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>12</td>
<td>96</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>15</td>
<td>128</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>20</td>
<td>160</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

- Source port
- Destination port
- Sequence number
- Acknowledgment number (if ACK set)
- Window Size
- Window Size
- Checksum
- Urgent pointer (if URG set)
- Options (if Data Offset > 5, padded at end with "0" bytes if necessary)

BEGINNING OF PAYLOAD (DATA)

- Offset specifies header size (offset of data) in 32-bit words
RETRANSMIT HAS HAPPENED

EWS = EFFECTIVE WINDOW SIZE
WS = WINDOW SIZE

START WITH EWS = WS
AFTER EVERY RETRANSMIT
EWS = EWS / 2
-window

\[ EWS \]

\[ \text{RETRANSMIT} \]

\[ 250 \]

\[ \text{RETRANSMIT} \]

\[ 125 \]

\[ \text{RETRANSMIT} \]

\[ 62 \]

\[ 0 \]
EWS

ACK -- window 5000

ACK------ window 10,000

ACK

ACK

150
Flow Control And TCP Window

- Receiver controls flow by telling sender size of currently available buffer measured in bytes
- Called *window advertisement*
- Each segment, including data segments, specifies size of window *beyond acknowledged byte*
- Window size may be zero (receiver cannot accept additional data at present)
- Receiver can send additional acknowledgement later when buffer space becomes available
TCP Checksum Computation

- Covers entire segment (header plus data)
- Required (unlike UDP)
- Pseudo header included in computation as with UDP
TCP Pseudo Header

<table>
<thead>
<tr>
<th>0</th>
<th>8</th>
<th>16</th>
<th>31</th>
</tr>
</thead>
<tbody>
<tr>
<td>SOURCE IP ADDRESS</td>
<td>DESTINATION IP ADDRESS</td>
<td>ZERO</td>
<td>PROTOCOL</td>
</tr>
</tbody>
</table>

[Diagram of TCP Pseudo Header with hand-drawn elements]
TCP Retransmission

- Designed for Internet environment
  - Delays on one connection vary over time
  - Delays vary widely between connections
- Fixed value for timeout will fail
  - Waiting too long introduces unnecessary delay
  - Not waiting long enough wastes network bandwidth with unnecessary retransmission
- Retransmission strategy must be adaptive
Adaptive Retransmission

- TCP keeps estimate of round-trip time (RTT) on each connection
- Round-trip estimate derived from observed delay between sending segment and receiving acknowledgement
- Timeout for retransmission based on current round-trip estimate
Difficulties With Adaptive Retransmission

- The problem is knowing when to retransmit
- Segments or ACKs can be lost or delayed, making round-trip estimation difficult or inaccurate
- Round-trip times vary over several orders of magnitude between different connections
- Traffic is bursty, so round-trip times fluctuate wildly on a single connection
Difficulties With Adaptive Retransmission (continued)

- Load imposed by a single connection can congest routers or networks
- Retransmission can *cause* congestion
- Because an internet contains diverse network hardware technologies, there may be little or no control for intra-network congestion
Solution: Smoothing

- Adaptive retransmission schemes keep a statistically smoothed round-trip estimate
- Smoothing keeps running average from fluctuating wildly, and keeps TCP from overreacting to change
- Difficulty: choice of smoothing scheme
Original Smoothing Scheme

- Let RTT be current (old) average round-trip time
- Let NRT be a new sample
- Compute
  \[ RTT = \alpha \times RTT + \beta \times NRT \]
  where
  \[ \alpha + \beta = 1 \]
- Example: \( \alpha = 0.8, \beta = 0.2 \)
- Large \( \alpha \) makes estimate less susceptible to a single long delay (more stable)
- Large \( \beta \) makes estimate track changes in round-trip time quickly
Problems With Original Scheme

- Associating ACKs with transmissions
  - TCP acknowledges receipt of data, not receipt of transmission
  - Assuming ACK corresponds to most recent transmission can cause instability in round-trip estimate (Cypress syndrome)
  - Assuming ACK corresponds to first transmission can cause unnecessarily long timeout
  - Both assumptions lead to lower throughput
Partridge/Karn Scheme†

- Solves the problem of associating ACKs with correct transmission
- Specifies ignoring round-trip time samples that correspond to retransmissions
- Separates timeout from round-trip estimate for retransmitted packets

† Also called *Karn’s Algorithm*
Partridge/Karn Scheme
(continued)

• Starts (as usual) with retransmission timer as a function of round-trip estimate

• Doubles retransmission timer value for each retransmission without changing round-trip estimate

• Resets retransmission timer to be function of round-trip estimate when ACK arrives for nonretransmitted segment
Flow Control And Congestion

- Receiver advertises window that specifies how many additional bytes it can accept
- Window size of zero means sender must not send normal data (ACKs and urgent data allowed)
- Receiver can never decrease window beyond previously advertised point in sequence space
- Sender chooses effective window smaller than receiver’s advertised window if congestion detected
Jacobson / Karels
Congestion Control

- Assumes long delays (packet loss) due to congestion
- Uses successive retransmissions as measure of congestion
- Reduces effective window as retransmissions increase
- Effective window is minimum of receiver’s advertisement and computed quantity known as the *congestion window*
Multiplicative Decrease

- In steady state (no congestion), the congestion window is equal to the receiver’s window.
- When segment lost (retransmission timer expires), reduce congestion window by half.
- Never reduce congestion window to less than one maximum sized segment.
Jacobson/Karels Slow Start

- Used when starting traffic or when recovering from congestion
- Self-clocking startup to increase transmission rate rapidly as long as no packets are lost
- When starting traffic, initialize the congestion window to the size of a single maximum sized segment
- Increase congestion window by size of one segment each time an ACK arrives without retransmission
Jacobson / Karels Congestion Avoidance

- When congestion first occurs, record one-half of last successful congestion window (flightsize) in a threshold variable
- During recovery, use slow start until congestion window reaches threshold
- Above threshold, slow down and increase congestion window by one segment per window (even if more than one segment was successfully transmitted in that interval)
Jacobson/Karels Congestion Avoidance
(continued)

- Increment window size on each ACK instead of waiting for complete window

  \[ \text{increase} = \text{segment / window} \]

Let \( N \) be segments per window, or

\[ N = \frac{\text{congestion\_window}}{\text{max segment size}} \]

so

\[ \text{increase} = \frac{\text{segment}}{N} \]

\[ = \frac{\text{MSS bytes}}{N} \]

\[ = \frac{\text{MSS}}{(\text{congestion\_window/ MSS})} \]

or

\[ \text{increase} = \frac{\text{MSS} \times \text{MSS}}{\text{congestion\_window}} \]
Changes In Delay

- Original smoothing scheme tracks the mean but not changes
- To track changes, compute
  \[
  \text{DIFF} = \text{SAMPLE} - \text{RTT} \\
  \text{RTT} = \text{RTT} + \delta \times \text{DIFF} \\
  \text{DEV} = \text{DEV} + \delta (|\text{DIFF}| - \text{DEV})
  \]
- DEV estimates mean deviation
- \(\delta\) is fraction between 0 and 1 that weights new sample
- Retransmission timer is weighted average of RTT and DEV:
  \[
  \text{RTO} = \mu \times \text{RTT} + \phi \times \text{DEV}
  \]
- Typically, \(\mu = 1\) and \(\phi = 4\)
Computing Estimated Deviation

- Extremely efficient (optimized) implementation possible
  - Scale computation by $2^n$
  - Use integer arithmetic
  - Choose $\delta$ to be $1/2^n$
  - Implement multiplication or division by powers of 2 with shifts
  - Research shows $n = 3$ works well
Measurement Of Internet Delays For 100 Successive Packets At 1 Second Intervals
TCP Round-Trip Estimation For Sampled Internet Delays
TCP Details

- Data flow may be shut down in one direction
- Connections started reliably, and terminated gracefully
- Connection established (and terminated) with a 3-way handshake
3-Way Handshake
For Connection Startup

Events At Site 1
Send SYN seq=x
Receive SYN + ACK segment
Send ACK y+1

Network Messages

Events At Site 2
Receive SYN segment
Send SYN seq=y, ACK x+1
Receive ACK segment
3-Way Handshake
For Connection Shutdown

Events At Site 1
(application closes connection)
Send FIN seq=x

Receive ACK segment

Receive FIN + ACK segment
Send ACK y+1

Network Messages

Events At Site 2
Receive FIN segment
Send ACK x+1
(inform application)

(application closes connection)
Send FIN seq=y, ACK x+1

Receive ACK segment
TCP Finite State Machine

- **CLOSED**
  - begin
  - passive open
  - close

- **LISTEN**
  - syn / syn + ack
  - active open / syn
  - send / syn

- **SYN RECVD**
  - reset
  - syn / syn + ack
  - close / fin

- **SYN SENT**
  - close / timeout / reset
  - syn + ack / ack

- **ESTABLISHED**
  - fin / ack
  - close / fin

- **FIN WAIT-1**
  - close / fin
  - fin / ack
  - ack /

- **CLOSING**
  - fin / ack
  - fin-ack / ack
  - ack /

- **TIME WAIT**
  - timeout after 2 segment lifetimes
  - fin / ack

- **LAST ACK**
  - ack /
TCP Urgent Data

- Segment with urgent bit set contains pointer to last octet of urgent data
- Urgent data occupies part of normal sequence space
- Urgent data can be retransmitted
- Receiving TCP should deliver urgent data to application "immediately" upon receipt
TCP Urgent Data
(continued)

- Two interpretations of standard
  - Out-of-band data interpretation
  - Data mark interpretation
Data-Mark Interpretation Of Urgent Data

- Has become widely accepted
- Single data stream
- Urgent pointer marks end of urgent data
- TCP informs application that urgent data arrived
- Application receives all data in sequence
- TCP informs application when end of urgent data reached
Data-Mark Interpretation
Of Urgent Data
(continued)

- Application
  - Reads all data from one stream
  - Must recognize start of urgent data
  - Must buffer normal data if needed later
- Urgent data marks *read* boundary
Urgent Data Delivery

- Receiving application placed in *urgent mode*
- Receiving application leaves urgent mode after reading urgent data
- Receiving application acquires *all* available urgent data when in urgent mode
Fast Retransmit

- Coarse-grained clock used to implement RTO
  - Typically 300 to 500ms per tick
- Timer expires up to 1s after segment dropped
- Fast retransmission
  - Sender uses three duplicate ACKs as trigger
  - Sender retransmits “early”
  - Sender reduces congestion window to half
Other TCP Details

- Silly Window Syndrome (SWS) avoidance
- Nagle algorithm
- Delayed ACKs
- For details, read the text
Comparison Of UDP And TCP

<table>
<thead>
<tr>
<th>Reliable Stream (TCP)</th>
<th>User Datagram (UDP)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Internet (IP)</td>
<td></td>
</tr>
<tr>
<td>Network Interface</td>
<td></td>
</tr>
</tbody>
</table>

- TCP and UDP lie between applications and IP
- Otherwise, completely different
## Comparison Of UDP and TCP

<table>
<thead>
<tr>
<th>UDP</th>
<th>TCP</th>
</tr>
</thead>
<tbody>
<tr>
<td>between apps. and IP packets called datagrams</td>
<td>between apps. and IP packets called segments</td>
</tr>
<tr>
<td>unreliable</td>
<td>reliable</td>
</tr>
<tr>
<td>checksum optional</td>
<td>checksum required</td>
</tr>
<tr>
<td>connectionless</td>
<td>connection-oriented</td>
</tr>
<tr>
<td>record boundaries</td>
<td>stream interface</td>
</tr>
<tr>
<td>intended for LAN</td>
<td>useful over WAN or LAN</td>
</tr>
<tr>
<td>no flow control</td>
<td>flow control</td>
</tr>
<tr>
<td>1-to-1, 1-many, many-1</td>
<td>1-to-1</td>
</tr>
<tr>
<td>allows unicast, multicast or broadcast</td>
<td>unicast only</td>
</tr>
</tbody>
</table>
TCP Vs. UDP Traffic

Around 95% of all bytes and around 85-95% of all packets on the Internet are transmitted using TCP.

– Eggert, et. al. CCR
Summary Of TCP

- Major transport service in the Internet
- Connection oriented
- Provides end-to-end reliability
- Uses adaptive retransmission
- Includes facilities for flow control and congestion avoidance
- Uses 3-way handshake for connection startup and shutdown