

Transport Protocols

Arjan Duresi
Louisiana State University

End-to-End Protocols

- Common end-to-end services
 - guarantee message delivery
 - deliver messages in the same order they are sent
 - deliver at most one copy of each message
 - support arbitrarily large messages
 - support synchronization
 - allow the receiver to flow control the sender
 - support multiple application processes on each host



- TCP
 - Key features
 - Header format
 - Mechanisms
 - Implementation choices
 - Slow start congestion avoidance
- UDP

TCP

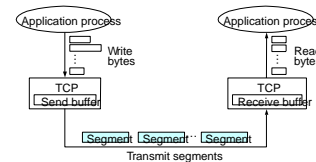
- Transport Control Protocol
- Key Services:
 - Send: Please send when convenient
 - Data stream push: Please send it all now, if possible.
 - Urgent data signaling: Destination TCP! please give this urgent data to the user (Urgent data is delivered in sequence. Push at the should be explicit if needed.)
 - Note: Push has no effect on delivery. Urgent requests quick delivery

End-to-End Protocols

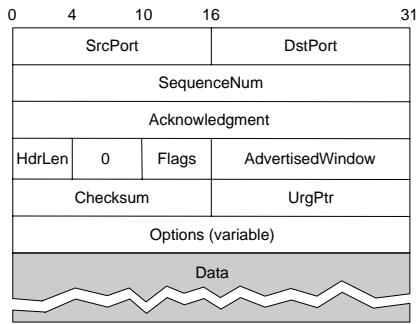
- Underlying best-effort network
 - drop messages
 - re-orders messages
 - delivers duplicate copies of a given message
 - limits messages to some finite size
 - delivers messages after an arbitrarily long delay

TCP Overview

- Connection-oriented
- Full duplex
- Byte-stream
 - app writes bytes
 - TCP sends *segments*
 - app reads bytes
- Flow control: keep sender from overrunning receiver
- Congestion control: keep sender from overrunning network



Segment Format

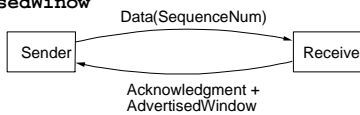


TCP Header

- Source Port (16 bits): Identifies source user process
20 = FTP, 23 = Telnet, 53 = DNS, 80 = HTTP, ...
- Destination Port (16 bits)
- Sequence Number (32 bits): Sequence number of the first byte in the segment. If SYN is present, this is the initial sequence number (ISN) and the first data byte is ISN+1.
- Ack number (32 bits): Next byte expected
- Data offset (4 bits): Number of 32-bit words in the header
- Reserved (6 bits)

Segment Format (cont)

- Each connection identified with 4-tuple:
 - (SrcPort, SrcIPAddr, DsrPort, DstIPAddr)
- Sliding window + flow control
 - acknowledgment, SequenceNum, AdvertisedWindow
- Flags
 - SYN, FIN, RESET, PUSH, URG, ACK
- Checksum
 - pseudo header + TCP header + data



TCP Header (Cont)

- Control (6 bits): Urgent pointer field significant, Ack field significant, Push function, Reset the connection, Synchronize the sequence numbers, No more data from sender



- Window (16 bits): Will accept [Ack] to [Ack]+[window]

TCP Header Format

Source Port	Dest Port	Seq No	Ack No	Data Offset	Resvd	Control	Window
16	16	32	32	4	6	6	16
Check-sum	Urgent	Options	Pad	Data			
16	16	x	y		← Size in bits		

TCP Header (Cont)

- Checksum (16 bits): covers the segment plus a pseudo header. Includes the following fields from IP header: source and dest adr, protocol, segment length. Protects from IP misdelivery.
- Urgent pointer (16 bits): Points to the byte following urgent data. Lets receiver know how much data it should deliver right away.
- Options (variable):
Max segment size (does not include TCP header, default 536 bytes), Window scale factor, Selective Ack permitted, Timestamp, No-Op, End-of-options

TCP Options

Kind	Length	Meaning
0	1	End of Valid options in header
1	1	No-op
2	4	Maximum Segment Size
3	3	Window Scale Factor
8	10	Timestamp

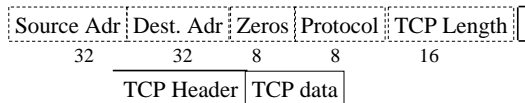
- End of Options: Stop looking for further option
- No-op: Ignore this byte. Used to align the next option on a 4-byte word boundary
- MSS: Does not include TCP header

TCP Service Responses

- Open ID: Informs the name assigned to the pending request
- Open Failure: Your open request failed
- Open Success: Your open request succeeded
- Deliver: Reports arrival of data
- Closing: Remote TCP has issued a close request
- Terminate: Connection has been terminated
- Status Response: Here is the connection status
- Error: Reports service request or internal error

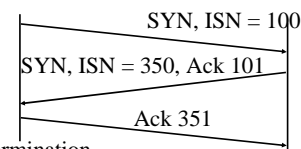
TCP Checksum

- Checksum is the 16-bit one's complement of the one's complement sum of a pseudo header of information from the IP header, the TCP header, and the data, padded with zero octets at the end (if necessary) to make a multiple of two octets.
- Checksum field is filled with zeros initially
- TCP length (in octet) is not transmitted but used in calculations.
- Efficient implementation in RFC1071.



TCP Mechanisms

- Connection Establishment
 - Three way handshake
 - SYN flag set ⇒ Request for connection



- Connection Termination
 - Close with FIN flag set
 - Abort

TCP Service Requests

- Unspecified passive open: Listen for connection requests from any user (port)
- Full passive open: Listen for connection requests from specified port
- Active open: Request connection
- Active open with data: Request connection and transmit data
- Send: Send data
- Allocate: Issue incremental allocation for receive data
- Close: Close the connection gracefully
- Abort: Close the connection abruptly
- Status: Report connection status

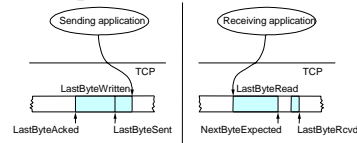
Data Transfer

- Stream: Every byte is numbered modulo 2^{32} .
- Header contains the sequence number of the first byte
- Flow control: Credit = number of bytes
- Data transmitted at intervals determined by TCP
 - Push ⇒ Send now
- Urgent: Send this data in ordinary data stream with urgent pointer
- If TPDU not intended for this connection is received, the "reset" flag is set in the outgoing segment

Implementation Policies (Choices)

- Send Policy:
 - Too little \Rightarrow More overhead. Too large \Rightarrow Delay
 - Push \Rightarrow Send now, if possible.
- Delivery Policy:
 - May store or deliver each in-order segment.
 - Urgent \Rightarrow Deliver now, if possible.
- Accept Policy:
 - May or May not discard out-of-order segments

Sliding Window Revisited



- Sending side
 - $LastByteAcked \leq LastByteSent$
 - $LastByteSent \leq LastByteWritten$
 - buffer bytes between $LastByteAcked$ and $LastByteWritten$
- Receiving side
 - $LastByteRead < NextByteExpected$
 - $NextByteExpected \leq LastByteRcvd + 1$
 - buffer bytes between $NextByteRead$ and $LastByteRcvd$

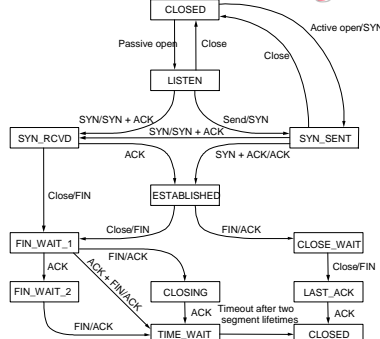
Implementation Policies (Cont)

- Retransmit Policy:
 - First only
 - Retransmit all
 - Retransmit individual (maintain separate timer for each segment)
- Ack Policy:
 - Immediate (no piggybacking)
 - Cumulative (wait for outgoing data or timeout)

Flow Control

- Send buffer size: $MaxSendBuffer$
- Receive buffer size: $MaxRcvBuffer$
- Receiving side
 - $LastByteRcvd - LastByteRead \leq MaxRcvBuffer$
 - $AdvertisedWindow = MaxRcvBuffer - (NextByteExpected - NextByteRead)$
- Sending side
 - $LastByteSent - LastByteAcked \leq AdvertisedWindow$
 - $EffectiveWindow = AdvertisedWindow - (LastByteSent - LastByteAcked)$
 - $LastByteWritten - LastByteAcked \leq MaxSendBuffer$
 - block sender if $(LastByteWritten - LastByteAcked) + y > MaxSenderBuffer$
- Always send ACK in response to arriving data segment
- Persist when $AdvertisedWindow = 0$

State Transition Diagram



Protection Against Wrap Around

- 32-bit SequenceNum

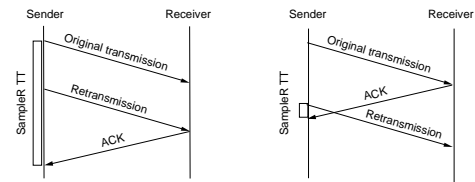
Bandwidth	Time Until Wrap Around
T1 (1.5 Mbps)	6.4 hours
Ethernet (10 Mbps)	57 minutes
T3 (45 Mbps)	13 minutes
FDDI (100 Mbps)	6 minutes
STS-3 (155 Mbps)	4 minutes
STS-12 (622 Mbps)	55 seconds
STS-24 (1.2 Gbps)	28 seconds

Keeping the Pipe Full

16-bit Advertised Window

Bandwidth	Delay x Bandwidth Product
T1 (1.5 Mbps)	18KB
Ethernet (10 Mbps)	122KB
T3 (45 Mbps)	549KB
FDDI (100 Mbps)	1.2MB
STS-3 (155 Mbps)	1.8MB
STS-12 (622 Mbps)	7.4MB
STS-24 (1.2 Gbps)	14.8MB

Karn/Partridge Algorithm



- Do not sample RTT when retransmitting
- Double timeout after each retransmission

TCP Extensions

- Implemented as header options
- Store timestamp in outgoing segments
- Extend sequence space with 32-bit timestamp (PAWS)
- Shift (scale) advertised window

Jacobson/ Karels Algorithm

- New Calculations for average RTT
- $\text{Diff} = \text{SampleRTT} - \text{EstRTT}$
- $\text{EstRTT} = \text{EstRTT} + (\delta \times \text{Diff})$
- $\text{Dev} = \text{Dev} + \delta(|\text{Diff}| - \text{Dev})$
 - where δ is a factor between 0 and 1
- Consider variance when setting timeout value
- $\text{TimeOut} = \mu \times \text{EstRTT} + \phi \times \text{Dev}$
 - where $\mu = 1$ and $\phi = 4$
- Notes
 - algorithm only as good as granularity of clock (500ms on Unix)
 - accurate timeout mechanism important to congestion control (later)

Adaptive Retransmission (Original Algorithm)

- Measure **sampleRTT** for each segment/ ACK pair
- Compute weighted average of RTT
 - $\text{EstRTT} = \alpha \times \text{EstRTT} + \beta \times \text{SampleRTT}$
 - where $\alpha + \beta = 1$
 - α between 0.8 and 0.9
 - β between 0.1 and 0.2
- Set timeout based on **EstRTT**
 - $\text{TimeOut} = 2 \times \text{EstRTT}$

Slow Start Flow Control

- Window = Flow Control Avoids receiver overrun
- Need congestion control to avoid network overrun
- The sender maintains two windows:
 - Credits from the receiver
 - Congestion window from the network
 - Congestion window is always less than the receiver window
- Starts with a congestion window (CWND) of 1 segment (one max segment size)
 - ⇒ Do not disturb existing connections too much.
- Increase CWND by 1 every time an ack is received

Slow Start (Cont)

- If packets lost, remember slow start threshold (SSThresh) to $CWND/2$
Set CWND to 1
Increment by 1 per ack until SSThresh
Increment by $1/CWND$ per ack afterwards

Louisiana State University 10 – TCP -31 CSC7702 FALL06

FRR (Cont)

- Upon receiving the third duplicate Ack:
 - Set SSThresh to $1/2$ of current CWND
 - Retransmit the missing segment
 - Set CWND to SSThresh+3
- For each successive duplicate Ack:
 - Increment CWND by 1 MSS
 - New packets are transmitted if allowed by CWND
- Upon receiving the next (non-duplicate) Ack:
 - Set CWND to SSThresh \Rightarrow Enter linear growth phase
- Receiver caches out-of-order data.

Louisiana State University 10 – TCP -34 CSC7702 FALL06

Slow Start (Cont)

- At the beginning, SSThresh = Receiver window
- After a long idle period (exceeding one round-trip time), reset the congestion window to one.
- Exponential growth phase is also known as “Slow start” phase
- The linear growth phase is known as “congestion avoidance phase”

Louisiana State University 10 – TCP -32 CSC7702 FALL06

User Datagram Protocol (UDP)

- Connectionless end-to-end service
- No flow control. No error recovery (no acks)
- Provides port addressing
- Error detection (Checksum) optional. Applies to pseudo-header (same as TCP) and UDP segment. If not used, it is set to zero.
- Used by network management

Source Port	Dest Port	Length	Check-sum
16	16	16	16

← Size in bits

Louisiana State University 10 – TCP -35 CSC7702 FALL06

Fast Retransmit and Recovery

- If 3 duplicate acks are received for the same packet, assume that the next packet has been lost. Retransmit it right away. Retransmit only one packet.
- Helps if a single packet is lost. Does not help if multiple packets lost.
- Ref: Stevens, Internet draft

Louisiana State University 10 – TCP -33 CSC7702 FALL06

Summary

- TCP provides reliable full-duplex connections.
- TCP Streams, credit flow control
- Slow-start, Fast retransmit/recovery
- UDP is connectionless and simple. No flow/error control.

Louisiana State University 10 – TCP -36 CSC7702 FALL06