**INTRO**

TL Protocols are implemented in the end systems : Not midway routers

IMP : A Socket is Created for Each Connection

TWO Protocols : TCP and UDP

Services of TL constrained by the Network Layer : except Reliability and security (encryption)

Extends IP’s delivery service b/w two end systems to a system b/w two processes running on the end systems

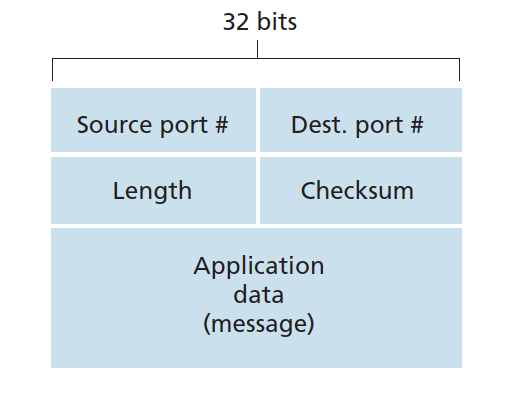
**MULTIPLEXING and DEMULTIPLEXING**

* A process can have one or more Sockets
* TL Delivers and receives data thru sockets : Not thru process
* Each Socket has a unique identifier
* Demux : Job of delivering data in a TL segment to the correct socket
* Mux : Job of gathering chunks at source, encapsulating each chunk in a segment (With header info) and passing these segments in the Network Layer
* These operations require
  + Sockets have a unique ID Number
  + Each segment has special fields that indicate where the segment has to be delivered. These include Source Port number and Dest Port Number
* Each port number is 16 Bit : 0 to 65535.
  + 0 to 1023 are called Well-Known Port Numbers : Cannot use them (reserved)
* UDP identified by 2 tuple : (Destination IP and Dest Port) along with the same for Source
* TCP is 4 Tuple : (Source IP and Port , Dest IP and Port) : All four fields used
* A Socket is Created for Each Connection and then closed after the connection is done

**UDP**

* No-Frills, Bare Bones Transport Protocol
* Just does Mux/Demux, Light Error Checking : Thats it
* Messages from app process -> Attaches Src and Dst ports for mux/demux - and two more things -> Passes to network layer
* NO HANDSHAKE
* Why is it Good
  + Finer App level control over the data sent
    - TCP not ready for real time things (Congestion, will keep trying to get reply)
  + No Connection Establishment
    - Just blasts the data away : No delay in conn estab
  + No Connection State
    - TCP maintains state (Params and variables) : Heavier
    - Server can handle more active UDP than TCP for this reason
  + Small Packet Overhead
    - **TCP has 20 Bytes while UDP has 8**
* Major Problem : NO CONGESTION CONTROL
  + Also affects TCP as high loss rate ma decrease TCP rates
* Can introduce reliability by doing something in App Layer

Segment Structure

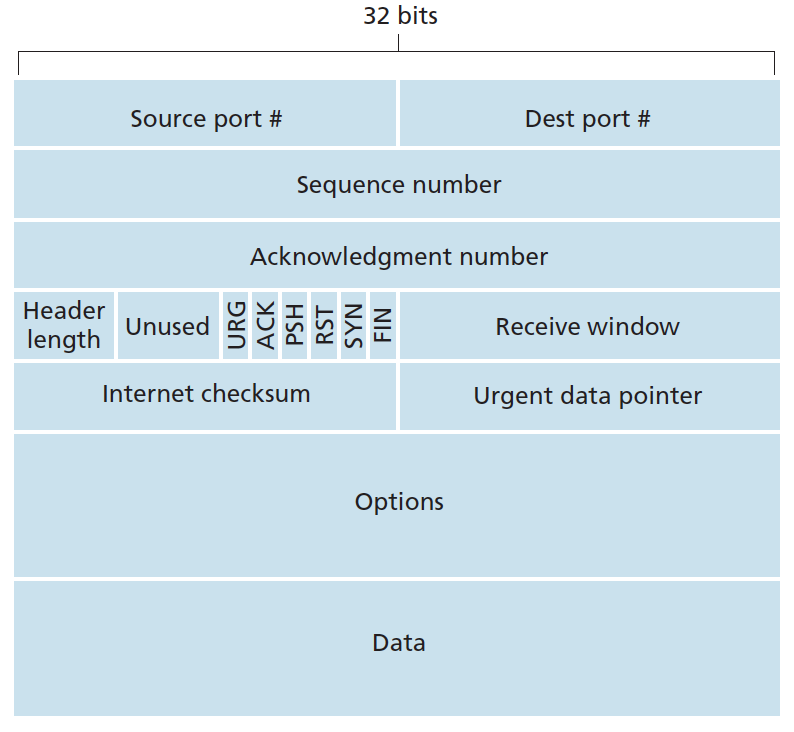


* In App data : Either DNS Query or Response, or Multimedia Audio
* **FOUR** fields each of **2 BYTES**
* Length is number of bytes of the segment **(Header + Data)**
* Checksum is for Error Checking
  + 1’s Complement of Adding the other three headers (Src and Dst Port + Length)
* IMP : Cannot Recover from Error : Segment is just Dropped/Passed with warning

**CONNECTION ORIENTED TRANSPORT : TCP**

* Conn Oriented : Handshake happens before anything else
* Message is sent as **BYTE Stream**
* Runs only in the two end systems
* Full Duplex Service
  + App layer data can flow in both directions
* Point to Point
  + Single Sender - Single Receiver
* 3- way Handshake
  + Client Sends Special Request -> Servers Sends special response -> Client sends 2nd special request
  + First two segments have no Payload( App layer data), Third May have
* Max data (**Only App Layer data considered)** that can be put in a segment is called Maximum Segment Size
  + Length of largest link layer frame
  + Set MMS to ensure that a TCP Segment + TCP Header (40 Bytes) will fit
  + MTU (Max Transmission Unit) = MSS + TCP Headers

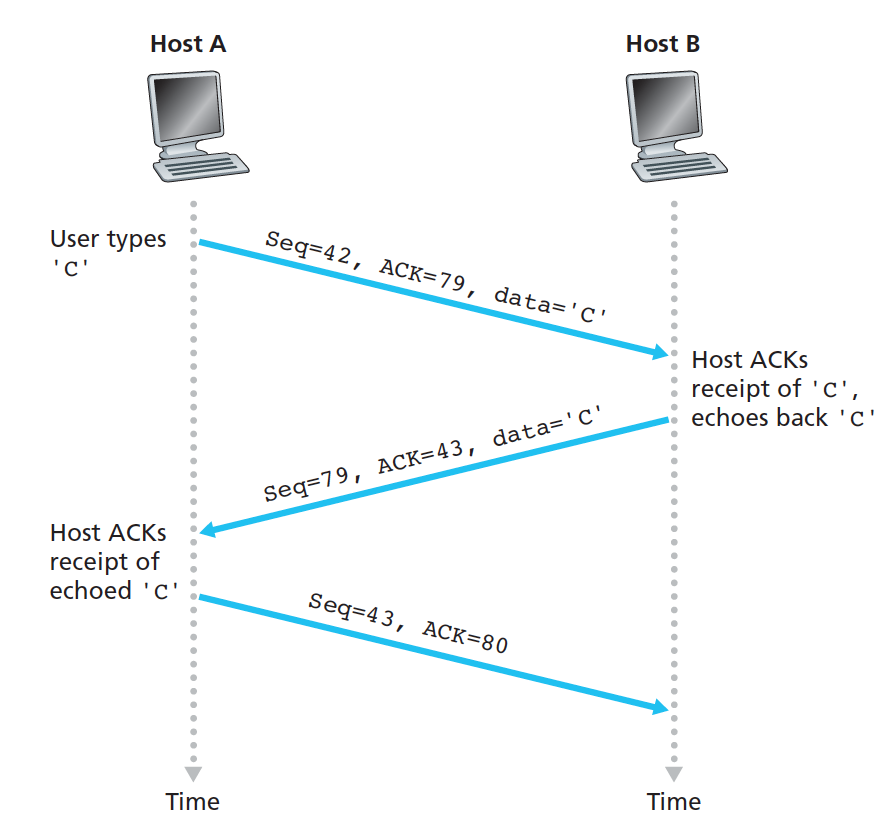
**SEGMENT STRUCTURE**

****

* MSS Sets max size of Data
  + Size of Remote login TCP may just be 21 bytes (20 for TCP and 1 data)
* Source and Destination Ports
* CheckSum Field
* 32 bit Sequence and 32 bit Ack Number
  + Used for Reliable Data transfer
* 16 Bit Receive bit window
  + Used in Flow Control
  + Number of Bytes the receiver is willing to accept
* 4 bit Header length field : Where the data actually starts
* Options field
  + Used to get MSS or Window Scaling factor
* FLAGS
  + ACK bit : If ack is valid
  + RST/SYN/FIN : Conn setup and teardown
  + PSH : Receiver must pass data to upper layer asap
  + URG : IF data is urgent

**SEQUENCE NUMBER AND ACKNOWLEDGMENT NUMBER**

* Critical to Reliable data transfer
* The sequence number for a segment is therefore the byte-stream number of the first byte in the segment. (Literally the number of the byte)
* The acknowledgment number that Host A puts in its segment is the sequence number of the next byte Host A is expecting from Host B
* \* Cumulative Acks : TCP only acknowledges bytes up to the first missing byte in the stream



**RTT and Timeout**

* Sample RTT : for a segment is the amount of time between when the segment is sent and when an acknowledgment for the segment is received.
  + Is estimated for a transmitted but unack segment
  + Not calculated for retransmitted segment
  + Fluctuates : Congestion control and Varying load
* Estimated RTT : Avg of Sample RTT : Formula
  + EstimatedRTT =
    - Alpha is usually 0.125 (⅛)
  + More weight on recent Sample than old
* DevRTT : How RTT Deviates from Estimated (Variation of RTT)
  + - Beta is usually 0.25(¼)
* Timeout Interval : EstimatedRTT + 4\*DevRTT
  + Initially 1 sec
  + If Timeout occurs : Double its value
  + As soon as segment is received : Compute using formula
  + Thus it is first exponential if ack dont come
* ERROR RECOVERY : Hybrid of Go Back N and Selective Repeat

FLOW CONTROL

* TCP provides a flow-control service to its applications to eliminate the possibility of the sender overflowing the receiver’s buffer.
* Flow control is thus a speed-matching service—matching the rate at which the sender is sending against the rate at which the receiving application is reading
* A variable called **receive window** maintained
  + Tell sender how much free space is left
  + Let buffer allotted be called RcvBuffer
  + At Receiver
    - Variables : LastByteRead and LastByteRvcd are kept
    - RcvBuffer LastByteRcvd - LastByteRead
      * Where rwnd is amount of space left
  + At Sender
    - Variables : LastByteSent and LastByteAcked are kept
    - IMP : If rwnd is 0, then Sender continuously sends 1 byte data to B and wait till the new ack got shows a non-zero rwnd
* UDP has no flow control
  + If buffer overflows, segments are dropped

PROPER CONN MANAGEMENT

START

* Step 1
  + Client tells server it wants to connect
  + Send a special TCP Segment with SYN bit = 1
  + No App Layer Data
  + Client randomly? Chooses a sequence number and put in TCP SYN Field
* Step 2
  + Server receives the segment (Hopefully), extracts SYN Segment
  + Allocates necessary TCP buffers
  + Sends Reply
    - No App Data
    - Again Syn Bit is 1
    - Ack field has the given sequence number
    - Sends its own sequence number
  + Called SYNACK Segment
* Step 3
  + Now the Client allocates buffer/variables
  + Replies by sending the Ack+1 and seqence number +1
  + May contain App-Layer data
  + SYN Bit is now 0 till end

END

* Client
  + Client sends a Segment with FIN bit set to 1
  + Goes to FIN\_WAIT\_1 state
    - Waits for segment from server with ack
  + If it gets ack : Goes to FIN\_WAIT\_2 state
    - Wait for another segment from server with FIN bit 1
  + When it gets the segment, it send ack and goes to TIME\_WAIT State
  + May resend (Not sure), wait for 30 sec to 2 min, then Formally closes

