Chapter 17 – TCP Introduction

TCP (Transmission Control Protocol)

A reliable, connection oriented transport protocol
(c.f. UDP) an unreliable connection–less transport protocol

From an application point of view TCP provides a reliable *byte pipe.*

The pipe can be read or written in units of virtually any amount reliably.
It is impossible for the receiver to dynamically determine the size of the writes

TCP and the OS cooperate to synchronize the sender and receiver

Receiver process blocks when no data exists to be read
Sender process blocks when *"too much"* data gets in the system.

Connection oriented service

Like phone service..
You must explicitly request a connection.
All connections are two party (no broadcast or multicast)

Overview of TCP functionality

UDP creates a single IP datagram for each `write` or `sendto`
TCP dynamically decides how much data to pack into an IP datagram.
This size is called the *segment size.*

TCP maintains a timer for each outstanding segment.
If the segment times out, it is retransmitted
The timer setting is *adaptive.*

ACK's are sent shortly after receipt of a segment.

Incoming packets with checksum errors are simply discarded (no NAKs)

TCP resequences incoming segments if necessary.

TCP detects and discards duplicated data.

TCP provides flow control that permits a slow receiver to constrain the rate at which a fast sender sends.

Objectives of TCP are the same as those of a reliable datalink protocol

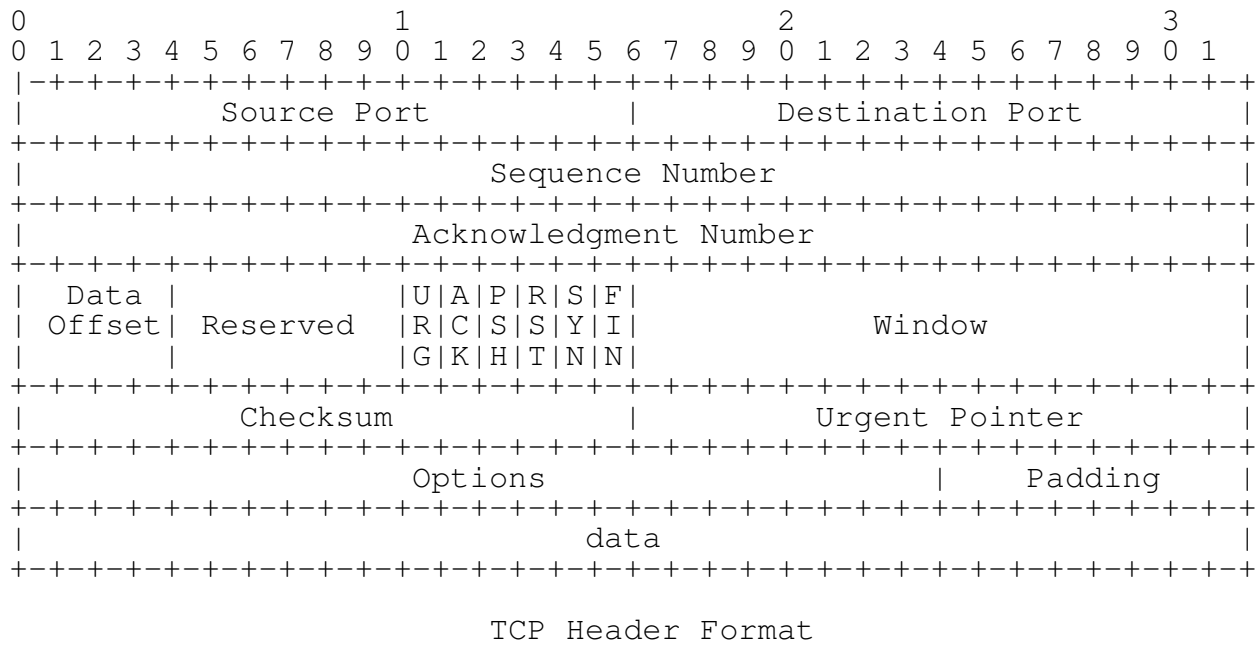
Provide a *virtual* channel that is error free on
A *physical* channel that is unreliable

Construction of a reliable transport on IP is more complicated than a reliable datalink protocol because of three problems

	IP	***Wire***
Service order	Not guaranteed	FIFO
Latency	Variable	Fixed
Bandwidth	Variable	Fixed

TCP Header

Follows the IP Header in a datagram
IP header – 20 bytes normally
TCP header – 20 bytes normally



TCP Header Format

Port numbers – As in UDP each packet has send and receive is associated with a logical TSAP address. A TCP connection is uniquely defined by a {(source–IP, source–port), (dest–IP, dest–port))}

Sequence number – The sequence number associated with the first byte in this packet ==> sequence numbers jump by packet size .. not by 1.

Ack number – The number of the next byte that the sender expects to receive on this connection = 1 + the sequence number of the last byte received correctly and in sequence.

Flags

URG – The urgent pointer is valid
ACK – The ack number is valid
PSH – The receiver should push this data to the app ASAP
RST – Reset (terminate) the connection
SYN – Initiate a connection
FIN – Normal termination of a connection

Window size – Other end is free to send new data with seq #'s between ack# and ack# + window

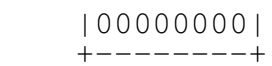
Urgent – Offset of the *last* byte of urgent data

Options – MSS is the most common.
Currently defined options include (kind indicated in octal):

Kind	Length	Meaning
0	–	End of option list.
1	–	No-Operation.
2	4	Maximum Segment Size.

Specific Option Definitions

End of Option List



Kind=0

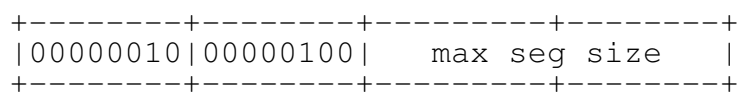
This option code indicates the end of the option list. This might not coincide with the end of the TCP header according to the Data Offset field. This is used at the end of all options, not the end of each option, and need only be used if the end of the options would not otherwise coincide with the end of the TCP header.

No−Operation

Kind=1

This option code may be used between options, for example, to align the beginning of a subsequent option on a word boundary. There is no guarantee that senders will use this option, so receivers must be prepared to process options even if they do not begin on a word boundary.

Maximum Segment Size



Kind=2 Length=4
Maximum Segment Size Option Data: 16 bits

If this option is present, then it communicates the maximum receive segment size at the TCP which sends this segment. This field must only be sent in the initial connection request (i.e., in segments with the SYN control bit set). If this option is not used, any segment size is allowed.