# Introduction to TCP :

TCP (Transmission Control Protocol) was specifically designed to provide a reliable end-to-end byte stream over an unreliable internetwork. An internetwork differs from a single network because different parts may have wildly different topologies, bandwidths, delays, packet sizes, and other parameters. TCP was designed to dynamically adapt to properties of the internetwork and to be robust in the face of many kinds of failures.

# TCP / IP Protocol



Each machine supporting TCP has a TCP transport entity, either a library procedure, a user process, or part of the kernel. In all cases, it manages TCP streams and interfaces to the IP layer. A TCP entity accepts user data streams from local processes, breaks them up into pieces not exceeding 64 KB (in practice, often 1460 data bytes in order to fit in a single Ethernet frame with the IP and TCP headers), and sends each piece as a separate IP datagram. When datagrams containing TCP data arrive at a machine, they are given to the TCP entity, which re-constructs the original byte streams.

# TCP / IP Protocol



The IP layer gives no guarantee that datagrams will be delivered properly, so it is up to TCP to time out and retransmit them as need be. Datagrams that do arrive may well do so in the wrong order; it is also up to TCP to reassemble them into messages in the proper sequence. In short, TCP must furnish the reliability that most users want and that IP does not provide.

## The TCP Protocol :

A key feature of TCP, and one which dominates the protocol design, is that every byte on a TCP connection has its own 32-bit sequence number. When the Internet began, the lines between routers were mostly 56-kbps leased lines, so a host blasting away at full speed took over 1 week to cycle through the sequence numbers. At modem network speeds, the sequence numbers can be consumed at an alarming rate. Separate 32-bit sequence numbers are used for acknowledgements and for the window mechanism.

# TCP / IP Protocol



The sending and receiving TCP entities exchange data in the form of segments. A TCP segment consists of a fixed 20-byte header (plus an optional part) followed by zero or more data bytes. The TCP software decides how big segments should be. It can accumulate data from several writes into one segment or can split data from one write over multiple segments.

Two limits restrict the segment size. First, each segment, including the TCP header, must fit in the 65,515 -byte IP payload. Second, each network has a maximum transfer unit, or MTU, and each segment must fit in the MTU. In practice, the MTU is generally 1500 bytes (the Ethernet payload size) and thus defines the upper bound on segment Size.

# TCP / IP Protocol



The basic protocol used by TCP entities is the sliding window protocol. When a sender transmits a segment, it also starts a timer. When the segment arrives at the destination, the receiving TCP entity sends back a segment (with data if any exist, otherwise without data) bearing an acknowledgement number equal to the next sequence number it expects to receive. If the sender's timer goes off before the acknowledgement is received, the sender transmits the segment again.

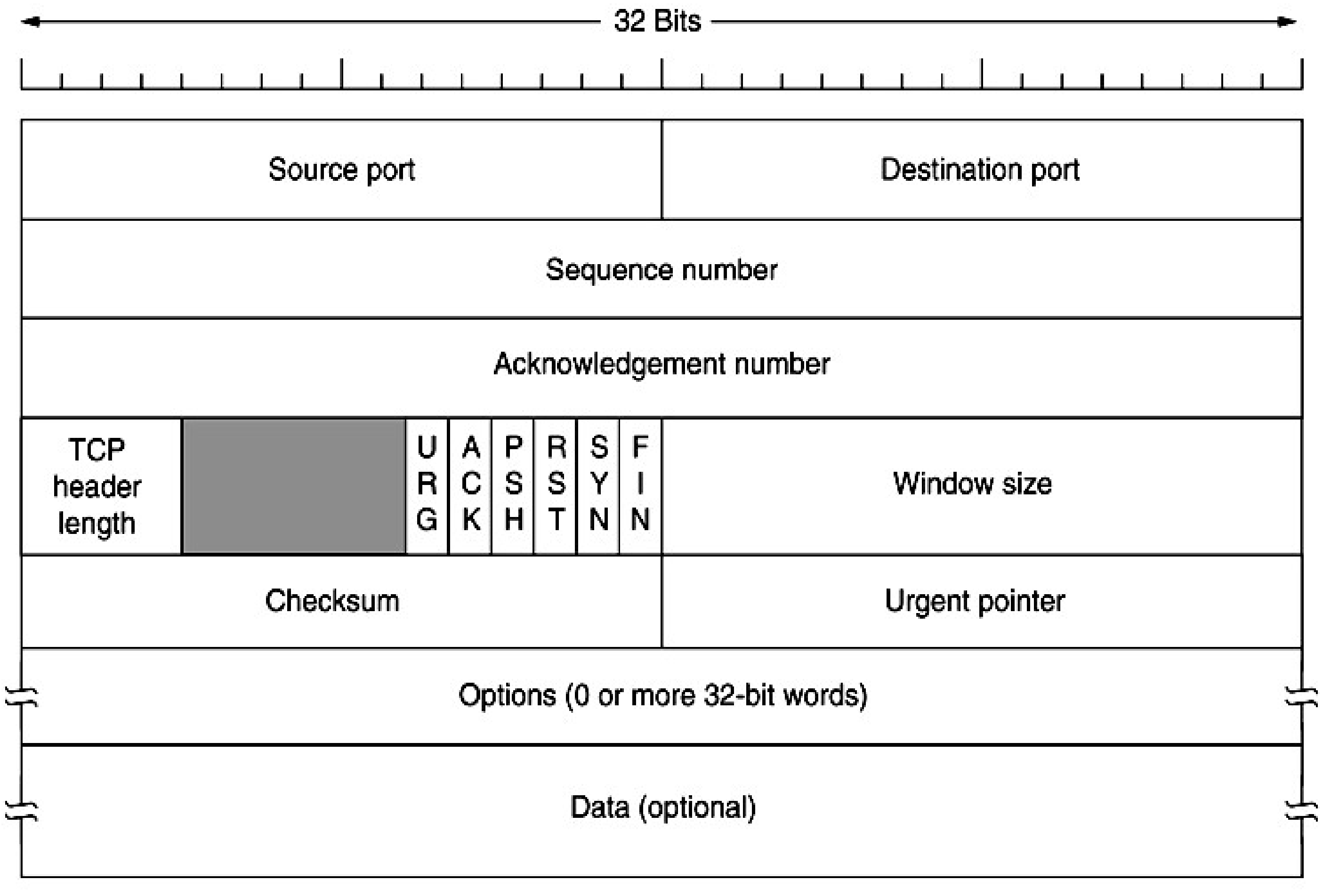
# TCP / IP Protocol



Although this protocol sounds simple, there are a number of sometimes subtle ins and outs, which we will cover below. Segments can arrive out of order, so bytes 3072-4095 can arrive but cannot be acknowledged because bytes 2048-3071 have not turned up yet. Segments can also be delayed so long in transit that the sender times out and retransmits them. The retransmissions may include different byte ranges than the original transmission, requiring a careful administration to keep track of which bytes have been correctly received so far. However, since each byte in the stream has its own unique offset, it can be done.

TCP must be prepared to deal with these problems and solve them in an efficient way. A considerable amount of effort has gone into optimizing the performance of TCP streams, even in the face of network problems.

# The TCP Segment Header :



# TCP / IP Protocol



Above figure shows the layout of a TCP segment. Every segment begins with a fixed-format, 20-byte header. The fixed header may be followed by header options. After the options, if any, up to 65,535 - 20 - 20 = 65,495 data bytes may follow, where the first 20 refer to the IP header and the second to the TCP header. Segments without any data are legal and are commonly used for acknowledgements and control messages.

Let us dissect the TCP header field by field. The *Source port* and *Destination port* fields identify the local end points of the connection. The well-known ports are defined at *www.iana.org* but each host can allocate the others as it wishes. A port plus its host's IP address forms a 48-bit unique end point. The source and destination end points together identify the connection.

# TCP / IP Protocol



| Port | Protocol | Use |
|---|---|---|
| 21 | FTP | File transfer |
| 23 | Telnet | Remote login |
| 25 | SMTP | E-mail |
| 69 | TFTP | Trivial file transfer protocol |
| 79 | Finger | Lookup information about a user |
| 80 | HTTP | World Wide Web |
| 110 | POP-3 | Remote e-mail access |
| 119 | NNTP | USENET news |

**Some assigned ports**

The *Sequence number* and *Acknowledgement number* fields perform their usual functions. Note that the latter specifies the next byte expected, not the last byte correctly received. Both are 32 bits long because every byte of data is numbered in a TCP stream.

The *TCP header length* tells how many 32-bit words are contained in the TCP header. This information is needed because the *Options* field is of variable length, so the header is, too. Technically, this field really indicates the start of the data within the segment, measured in 32-bit words, but that number is just the header length in words, so the effect is the same.

Next comes a 6-bit field that is not used. The fact that this field has survived intact for over a quarter of a century is testimony to how well thought out TCP is.

# TCP / IP Protocol



Now come six 1-bit flags. *URG* is set to 1 if the *Urgent pointer* is in use. The *Urgent pointer* is used to indicate a byte offset from the current sequence number at which urgent data are to be found. This facility is in lieu of interrupt messages.

The *ACK* bit is set to 1 to indicate that the *Acknowledgement number* is valid. If *ACK* is 0, the segment does not contain an acknowledgement so the *Acknowledgement number* field is ignored.

The *PSH* bit indicates PUSHed data. The receiver is hereby kindly requested to deliver the data to the application upon arrival and not buffer it until a full buffer has been received (which it might otherwise do for efficiency).

**The *RST* bit is used to reset a connection that has become confused due to a host crash or some other reason. It is also used to reject an invalid segment or refuse an attempt to open a connection. In general, if you get a segment within *RST* bit on, you have a problem on your hands.**

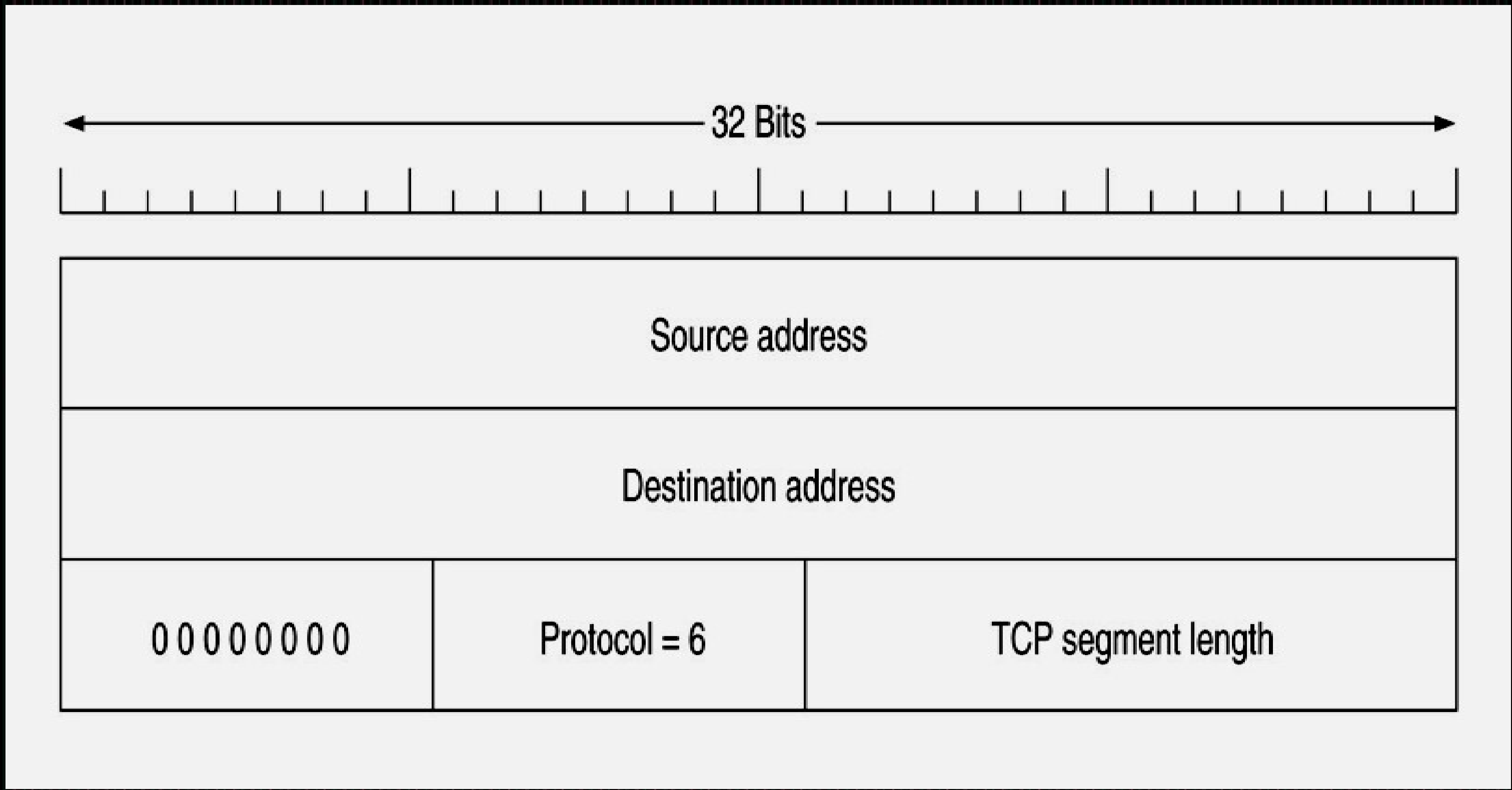
**The *SYN* bit is used to establish connections. The connection request has *SYN* = 1 and *ACK* = 0 to indicate that the piggyback acknowledgement field is not in use. The connection reply does bear an acknowledgement, so it has *SYN* = 1 and *ACK* = 1. In essence the *SYN* bit is used to denote CONNECTION REQUEST and CONNECTION ACCEPTED, with the *ACK* bit used to distinguish between those two possibilities.**

**The *FIN* bit is used to release a connection. It specifies that the sender has no more data to transmit. However, after closing a connection, the closing process may continue to receive data indefinitely. Both *SYN* and *FIN* segments have sequence numbers and are thus guaranteed to be processed in the correct order.**

**Flow control in TCP is handled using a variable-sized sliding window. The *Window size* field tells how many bytes may be sent starting at the byte acknowledged. A *Window size* field of 0 is legal and says that the bytes up to and including *Acknowledgement number* - 1 have been received, but that the receiver is currently badly in need of a rest and would like no more data for the moment. The receiver can later grant permission to send by transmitting a segment with the same *Acknowledgement number* and a nonzero *Window size* field.**

A *Checksum* is also provided for extra reliability. It checksums the header, the data, and the conceptual pseudoheader shown in Fig.-C. When performing this computation, the TCP *Checksum* field is set to zero and the data field is padded out with an additional zero byte if its length is an odd number. The checksum algorithm is simply to add up all the 16-bit words in one's complement and then to take the one's complement of the sum. As a consequence, when the receiver performs the calculation on the entire segment, including the *Checksum* field, the result should be 0.

# TCP / IP Protocol




The pseudoheader included in the TCP checksum.

# TCP / IP Protocol



The pseudoheader contains the 32-bit IP addresses of the source and destination machines, the protocol number for TCP (6), and the byte count for the TCP segment (including the header). Including the pseudoheader in the TCP checksum computation helps detect misdelivered packets, but including it also violates the protocol hierarchy since the IP addresses in it belong to the IP layer, not to the TCP layer. UDP uses the same pseudoheader for its checksum.

# TCP / IP Protocol



The *Options* field provides a way to add extra facilities not covered by the regular header. The most important option is the one that allows each host to specify the maximum TCP payload it is willing to accept. Using large segments is more efficient than using small ones because the 20-byte header can then be accounted over more data, but small hosts may not be able to handle big segments.

During connection setup, each side can announce its maximum and see its partner's. If a host does not use this option, it defaults to a 536-byte payload. All Internet hosts are required to accept TCP segments of 536 + 20 = 556 bytes. The maximum segment size in the two directions need not be the same.

# TCP / IP Protocol



For lines with high bandwidth, high delay, or both, the 64- KB window is often a problem. On a T3 line (44.736 Mbps), it takes only 12 msec to output a full 64-KB window. If the round-trip propagation delay is 50 msec (which is typical for a transcontinental fiber), the sender will, be idle 3/4 of the time waiting for acknowledgements. On a satellite connection, the situation is even worse.

A larger window size would allow the sender to keep pumping data out, but using the 16-bit *Window size* field, there is no way to express such a size. This number allows both sides to shift the *Window size* field up to 14 bits to the left, thus allowing windows of up to 2^30 bytes. Most TCP implementations now support this option.

# TCP / IP Protocol

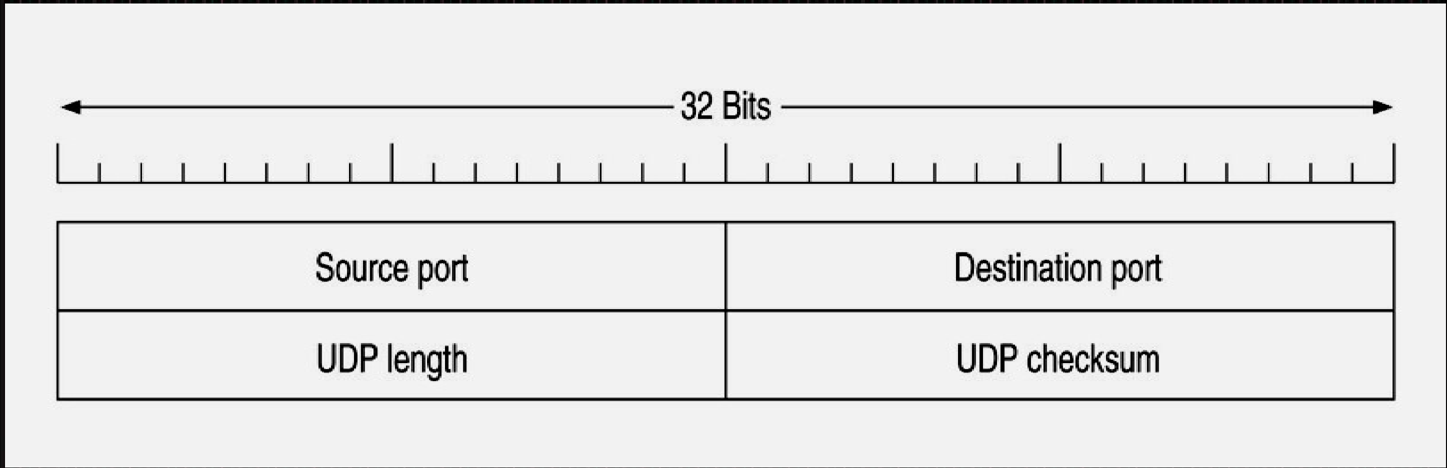


Another option proposed and now widely implemented is the use of the selective repeat instead of go back n protocol. If the receiver gets one bad segment and then a large number of good ones, the normal TCP protocol will eventually time out and retransmit all the unacknowledged segments, including all those that were received correctly (i.e., the go back n protocol). After it gets these, it can acknowledge all the buffered data, thus reducing the amount of data retransmitted.

## Introduction to UDP :

The Internet protocol suite supports a connectionless transport protocol, UDP (User Datagram Protocol). UDP provides a way for applications to send encapsulated IP datagrams and send them without having to establish a connection.

UDP transmits segments consisting of an 8-byte header followed by the pay-load. The header is shown in Figure. The two ports serve to identify the end points within the source and destination machines. When a UDP packet arrives, its payload is handed to the process attached to the destination port.




**In fact, the main value of having UDP over just using raw IP is the addition of the source and destination ports. Without the port fields, the transport layer would not know what to do with the packet. With them, it delivers segments correctly.**

**The source port is primarily needed when a reply must be sent back to the source. By copying the *source port* field from the incoming segment into the *destination port* field of the outgoing segment, the process sending the reply can specify which process on the sending machine is to get it.**

**The *UDP length* field includes the 8-byte header and the data. The *UDP checksum* is optional and stored as 0 if not computed (a true computed 0 is stored as all ls). Turning it off is foolish unless the quality of the data does not matter (e.g., digitized speech).**

**It is probably worth mentioning explicitly some of the things that UDP does *not* do. It does not do flow control, error control, or retransmission upon receipt of a bad segment. All of that is up to the user processes. What it does do is pro-vide an interface to the IP protocol with the added feature of demultiplexing multiple processes using the ports. That is all it does. For applications that need to have precise control over the packet flow, error control, or timing, UDP provides just what the doctor ordered.**

DEPARTMENT OF COMPUTER SCIENCE, SARDAR PATEL UNIVERSITY, V. V. NAGAR.

One area where UDP is especially useful is in client-server situations. Often, the client sends a short request to the server and expects a short reply back. If either the request or reply is lost, the client can just time out and try again. Not only is the code simple, but fewer messages are required (one in each direction) than with a protocol requiring an initial setup.

An application that uses UDP this way is DNS (the Domain Name System),