Transmission Control Protocol - TCP

The Transmission Control Protocol (TCP), documented in RFC 793, makes up for IP's deficiencies by providing reliable, stream-oriented connections that hide most of IP's shortcomings.

Introduction

- TCP is a *standard protocol* with STD number 6. TCP is described by *RFC 793 - Transmission Control Protocol*. Its status is *recommended*, but in practice every TCP/IP implementation which is not used exclusively for routing will include TCP.

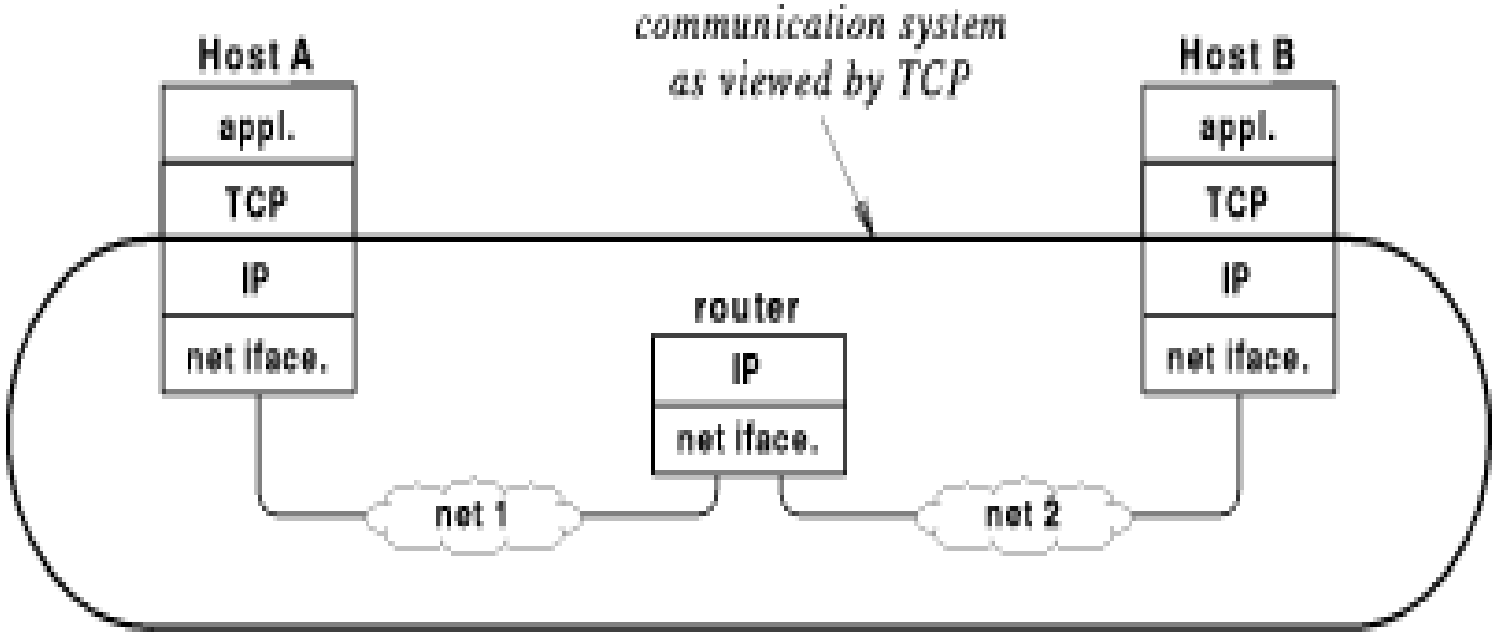
TCP

- *Transmission Control Protocol* (TCP) is the most widely used transport protocol
- Provides *reliable data delivery* by using IP *unreliable datagram delivery*
- Compensates for loss, delay, duplication and similar problems in Internet components
- Reliable delivery is high-level, familiar model for construction of applications

Using IP for data delivery

- TCP uses IP for data delivery (like UDP)
- Endpoints are identified by ports (like UDP)
- Allows multiple connections on each host
- Ports may be associated with an application or a process
- IP treats TCP like data and does not interpret any contents of the TCP message

IP treats TCP like data and does not interpret any contents of the TCP message



TCP travels in IP datagrams

Internet routers only look at IP header to forward datagrams

TCP at destination interprets TCP messages

Introduction

- TCP provides considerably more facilities for applications than UDP, notably error recovery, flow control and reliability. TCP is a *connection-oriented* protocol unlike UDP which is *connection-less*. Most of the user application protocols, such as TELNET and FTP, use TCP.

TCP Concept

- The primary purpose of TCP is *to provide reliable logical circuit or connection service between pairs of processes.* It does *not* assume reliability from the lower-level protocols (such as IP) so TCP must guarantee this itself.

TCP Features

- TCP can be characterised by the following facilities it provides for the applications using it:
  - Stream Data Transfer
  - Full Duplex
  - Multiplexing
  - Logical Connections
  - Reliability
  - Flow Control

Stream Data Transfer

- From the application's viewpoint, TCP transfers *a contiguous stream of bytes* through the internet. The application does not have to bother with chopping the data into basic blocks or datagrams. TCP does this by grouping the bytes in *TCP segments*, which are passed to IP for transmission to the destination. Also, TCP itself decides how to segment the data and it may forward the data at its own convenience.
- Sometimes, an application needs to be sure that all the data passed to TCP has actually been transmitted to the destination. For that reason, a *push* function is defined. It will push all remaining TCP segments still in storage to the destination host. The normal *close connection* function also pushes the data to the destination.

Stream Data Transfer

- Application delivers arbitrarily large chunks of data to TCP as a``stream"
- TCP breaks this data into *segments*, each of which fits into an IP datagram
- Original stream is numbered by bytes
- Segment contains *sequence number* of data bytes

Reliability

- TCP assigns a sequence number to each byte transmitted, and expects a positive acknowledgment (ACK) from the receiving TCP. If the ACK is not received within a timeout interval, the data is retransmitted. As the data is transmitted in blocks (TCP segments) only the sequence number of the first data byte in the segment is sent to the destination host. The receiving TCP uses the sequence numbers to rearrange the segments when they arrive out of order, and to eliminate duplicate segments.

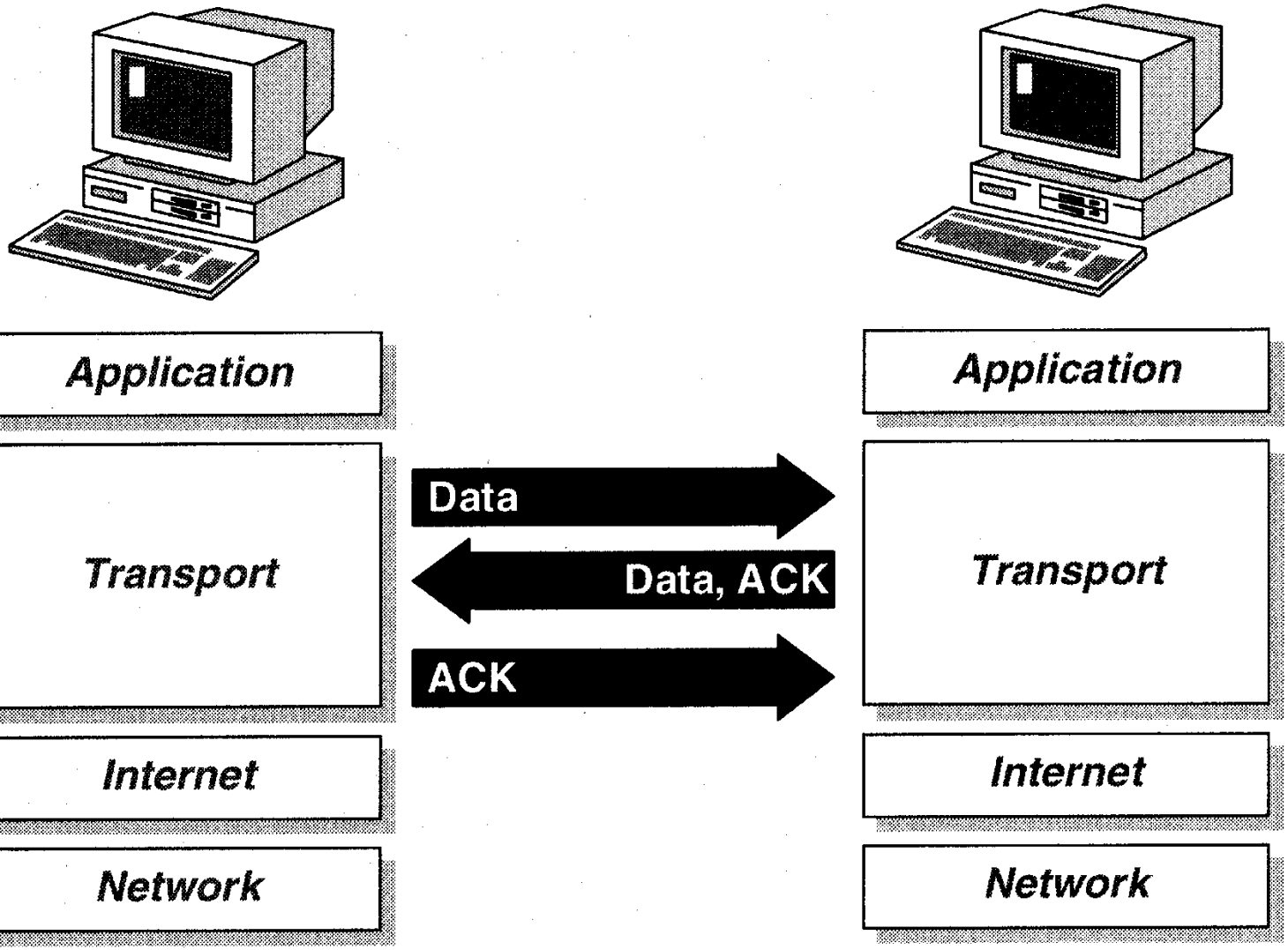
Reliability

- TCP uses many techniques to provide reliable delivery
- Recovers from
  - Lost packets
  - Duplicate packets
  - Delayed packets
  - Corrupted data
  - Transmission speed mismatches
  - Congestion
  - System reboots

Lost Packets

- TCP uses *positive acknowledgement with retransmission* to achieve reliable data delivery
- Recipient sends *acknowledgement* control messages (ACK) to sender to verify successful receipt of data
- Sender sets timer when data transmitted; if timer expires before acknowledgement arrives, sender retransmits (with new timer)

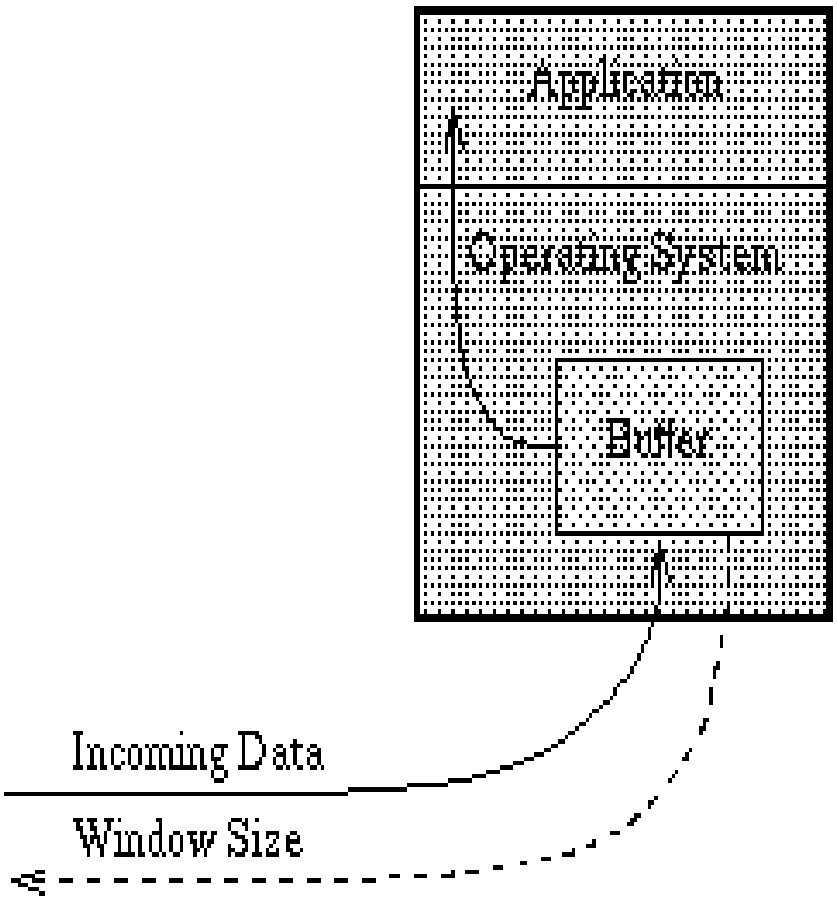
Lost Packets



Flow Control

- The receiving TCP, when sending an ACK back to the sender, also indicates to the sender the number of bytes it can receive beyond the last received TCP segment, without causing overrun and overflow in its internal buffers. This is sent in the ACK in the form of the highest sequence number it can receive without problems. This mechanism is also referred to as a *window*-mechanism.

Flow Control



- TCP's method of flow control is called *sliding window*.
- Sliding window algorithms are a method of flow control for network data transfers. TCP, the Internet's stream transfer protocol, uses a sliding window algorithm.
- A sliding window algorithm places a buffer between the application program and the network data flow. For TCP, the buffer is typically in the operating system kernel, but this is more of an implementation detail than a hard-and-fast requirement.

Multiplexing

- Is achieved through the use of ports, just as with UDP.

Logical Connections

- The reliability and flow control mechanisms described above require that TCP initializes and maintains certain status information for each ``data stream''. The combination of this status, including sockets, sequence numbers and window sizes, is called a logical connection. Each connection is uniquely identified by the pair of sockets used by the sending and receiving processes.

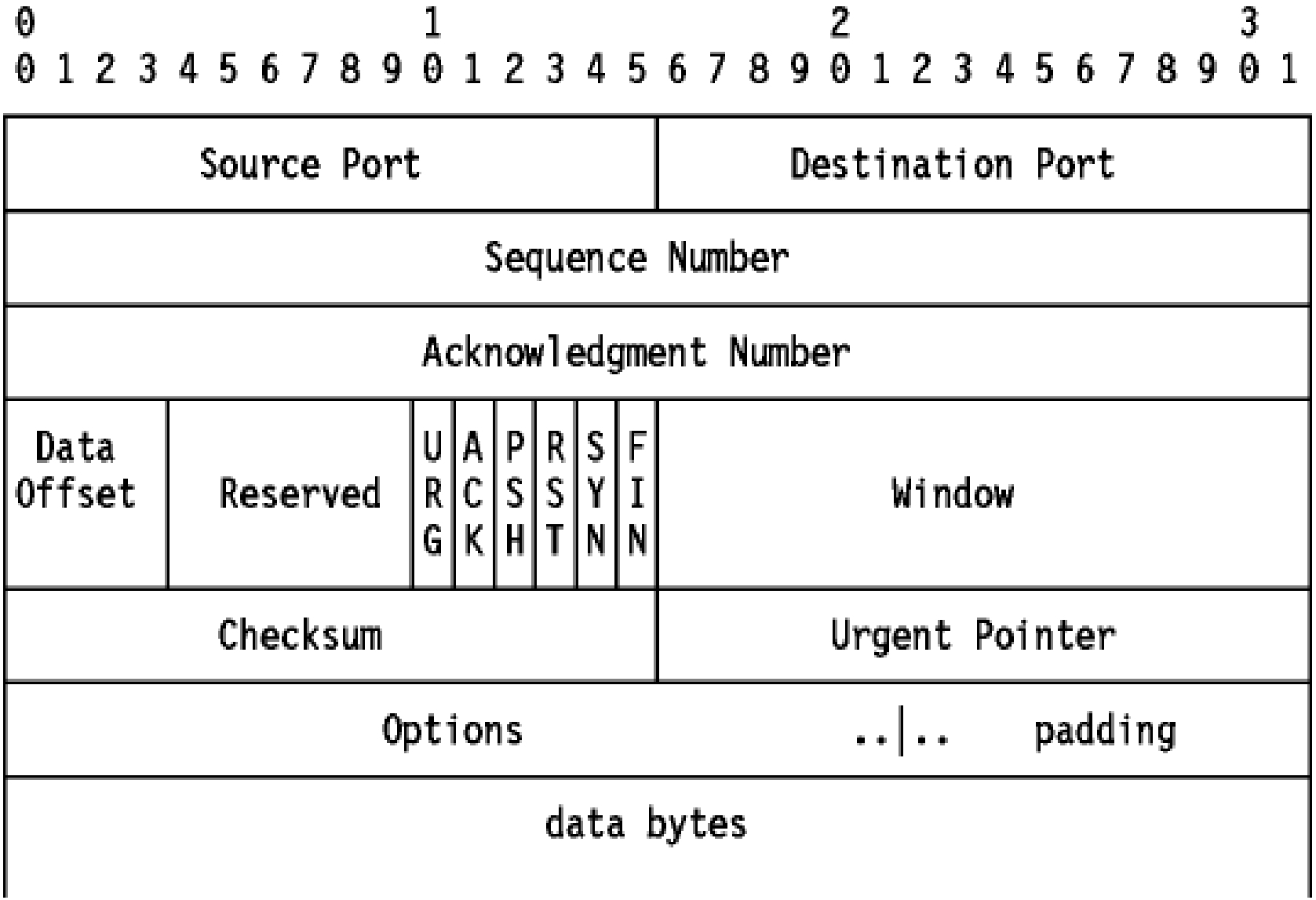
Full Duplex

- TCP provides for concurrent data streams in both directions.

Features of TCP - Summary

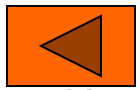
- *Connection oriented*: Application requests connection to destination and then uses connection to deliver data to transfer data
- *Point-to-point*: A TCP connection has two endpoints
- *Reliability*: TCP guarantees data will be delivered without loss, duplication or transmission errors
- *Full duplex*: The endpoints of a TCP connection can exchange data in both directions simultaneously
- *Stream interface*: Application delivers data to TCP as a continuous *stream*, with no record boundaries; TCP makes no guarantees that data will be received in same blocks as transmitted
- *Reliable connection startup*: *Three-way handshake* guarantees reliable, synchronized startup between endpoints
- *Graceful connection shutdown*: TCP guarantees delivery of all data after endpoint shutdown by application

TCP Segment Format



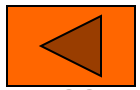
TCP Segment Format

- **Source Port**
  - The 16-bit source port number, used by the receiver to reply.
- **Destination Port**
  - The 16-bit destination port number.
- **Sequence Number**
  - The sequence number of the first data byte in this segment. If the SYN control bit is set, the sequence number is the initial sequence number (n) and the first data byte is n+1.



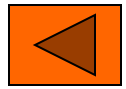
TCP Segment Format

- **Acknowledgment Number**
  - If the ACK control bit is set, this field contains the value of the next sequence number that the receiver is expecting to receive.
- **Data Offset**
  - The number of 32-bit words in the TCP header. It indicates where the data begins.
- **Reserved**
  - Six bits reserved for future use; must be zero.



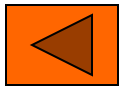
TCP Segment Format - 6 Flags

- **URG**
  - Indicates that the urgent pointer field is significant in this segment.
- **ACK**
  - Indicates that the acknowledgment field is significant in this segment.
- **PSH**
  - Push function.
- **RST**
  - Resets the connection.
- **SYN**
  - Synchronizes the sequence numbers.
- **FIN**
  - No more data from sender.



TCP Segment Format

- **Window**
  - Used in ACK segments. It specifies the number of data bytes beginning with the one indicated in the acknowledgment number field which the receiver (= the sender of this segment) is willing to accept.

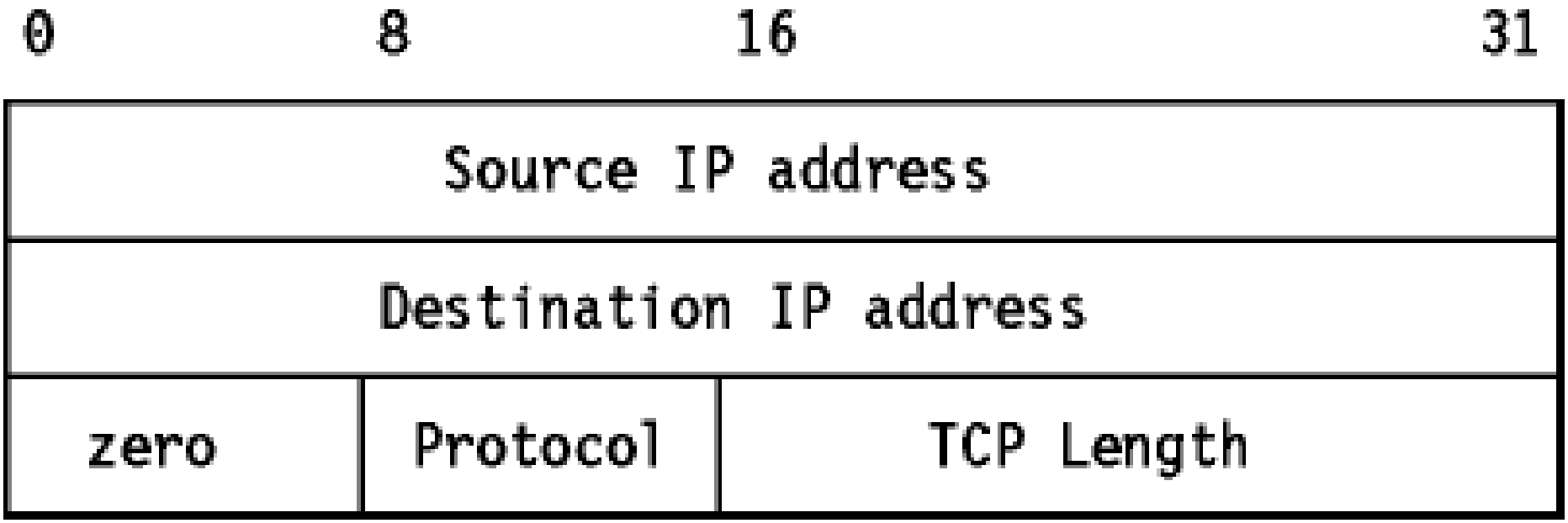


TCP Segment Format

Checksum

- The 16-bit one's complement of the one's complement sum of all 16-bit words in a pseudo-header, the TCP header and the TCP data. While computing the checksum, the checksum field itself is considered zero.
- The pseudo-header is the same as that used by UDP for calculating the checksum. It is a pseudo-IP-header, only used for the checksum calculation, with the format shown in Figure - Pseudo-IP Header:

Figure: Pseudo-IP Header



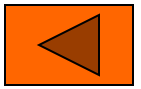
TCP Segment Format

Urgent Pointer

- Points to the first data octet following the urgent data. Only significant when the URG control bit is set.

Options

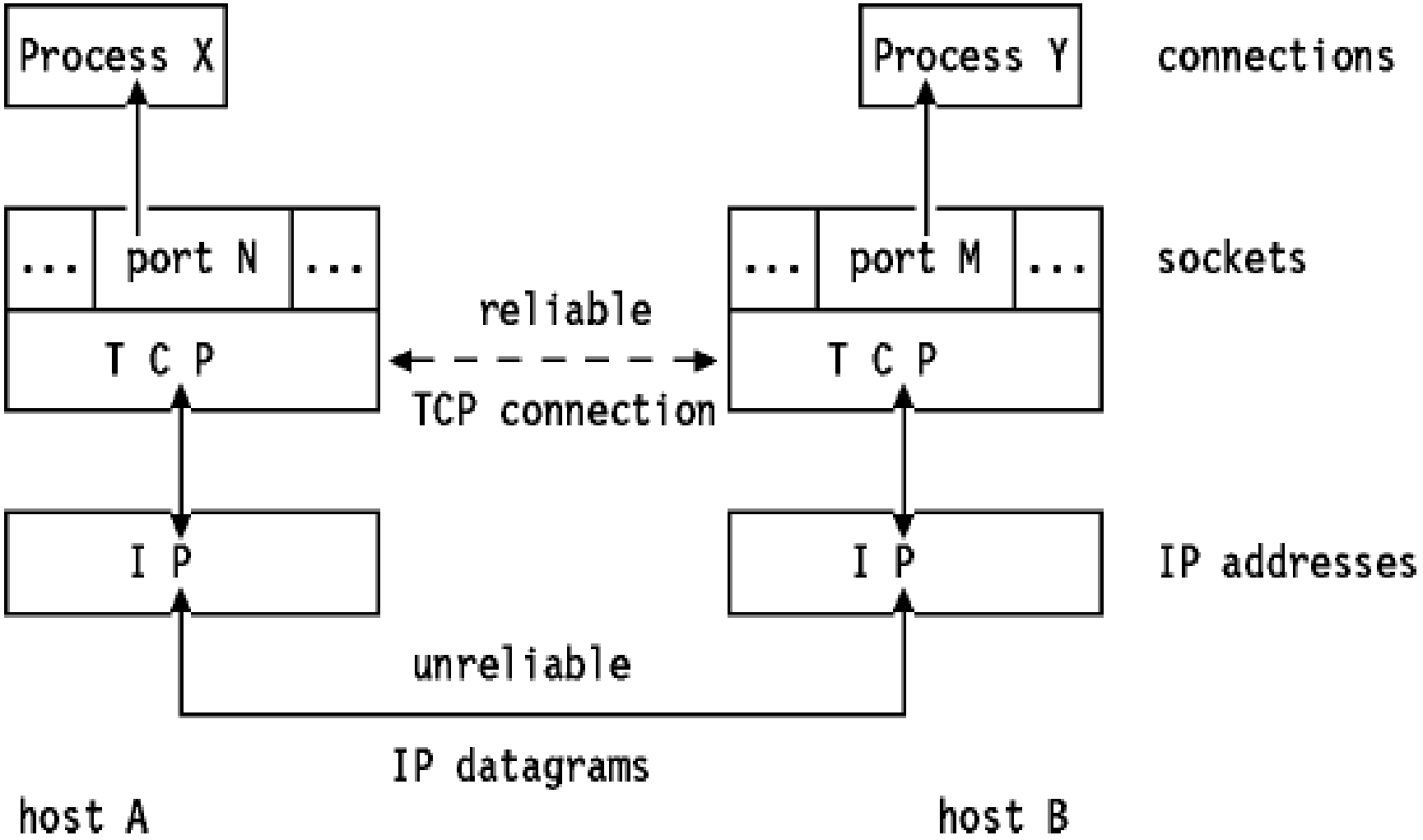
- Just as in the case of IP datagram options, options can be either:
  - A single byte containing the option number, or
  - A variable length option in the following format:



TCP Segment Format - Options

- There are currently only three options defined:
- ***Kind Length Meaning***
- 0 - End of option list.
- 1 - No-Operation.
- 2 4 Maximum Segment Size.

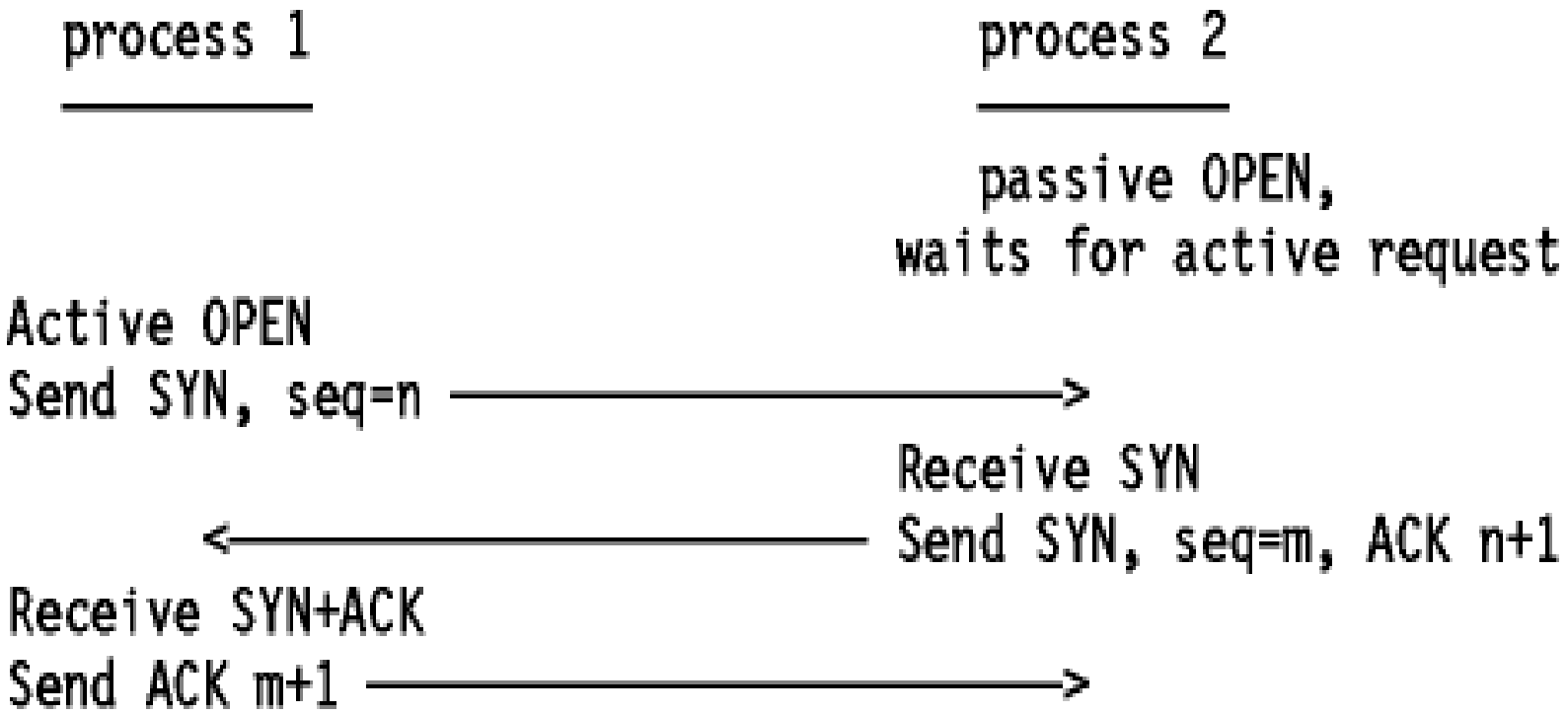
***Figure: TCP Connection* - Processes X and Y communicate over a TCP connection carried by IP datagrams**



TCP Connection Establishment

- Before any data can be transferred, a connection has to be established between the two processes. One of the processes (usually the server) issues a *passive OPEN* call, the other an *active OPEN* call. The passive OPEN call remains dormant until another process tries to connect to it by an active OPEN.
- On the network, three TCP segments are exchanged:

Figure: TCP Connection Establishment

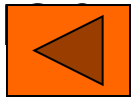


The connection is now established and the two data streams (one in each direction) have been initialized (sequence numbers)

Three-way Handshake

- TCP uses *three-way handshake* for reliable connection establishment and termination
  - Host 1 sends segment with SYN bit set and random sequence number
  - Host 2 responds with segment with SYN bit set, acknowledgement to Host 1 and random sequence number
  - Host 1 responds with acknowledgement
- TCP will retransmit lost segments

⌘ Note that the exchanged TCP segments include the initial sequence numbers from both sides, to be used on subsequent data transfers. Random sequence numbers ensure synchronisation between endpoints



Acknowledgments and Retransmission

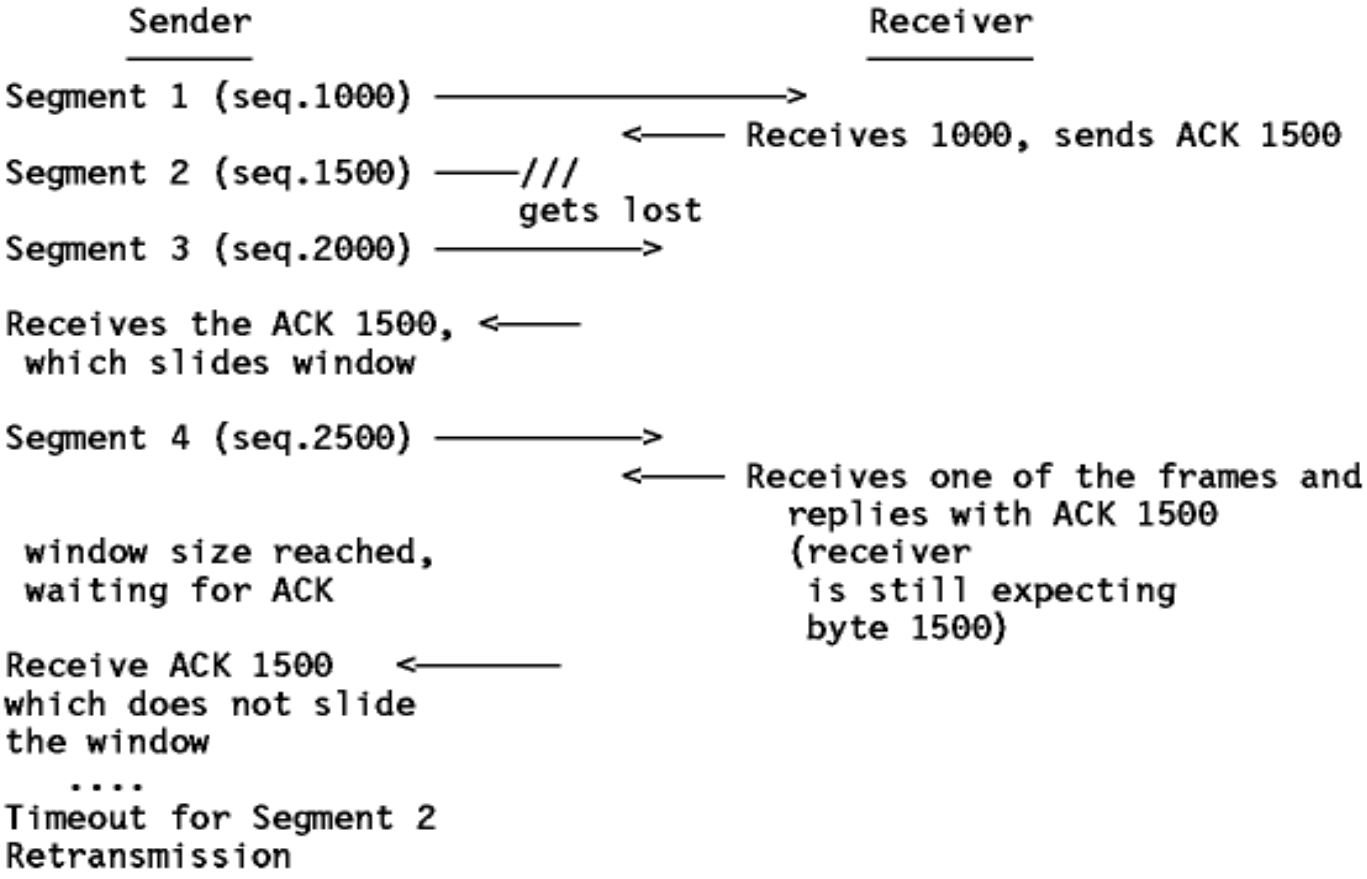
- TCP sends data in variable length segments. Sequence numbers are based on a byte count. *Acknowledgements specify the sequence number of the next byte that the receiver expects to receive.*
- Now suppose that a segment gets lost or corrupted. In this case, the receiver will acknowledge all further well-received segments with an acknowledgement referring to the first byte of the missing packet. The sender will stop transmitting when it has sent all the bytes in the window. Eventually, a timeout will occur and the missing segment will be retransmitted.

Acknowledgment

- Receiver sends segment with sequence number of acknowledged *data* (not segments)
- One ACK can acknowledge many segments

Acknowledgments and Retransmission

- Suppose a window size of 1500 bytes, and segments of 500 bytes.



Acknowledgments and Retransmission

- A problem now arises, since the sender does know that segment 2 is lost or corrupted, but doesn't know anything about segments 3 and 4. The sender should at least retransmit segment 2, but it could also retransmit segments 3 and 4 (since they are within the current window). It is possible that:
  - 1. Segment 3 has been received, and for segment 4 we don't know: it could be received, but ACK didn't reach us yet, or it could be lost also.
  - 2. Segment 3 was lost, and we received the ACK 1500 upon the reception of segment 4.

Acknowledgments and Retransmission

- Each TCP implementation is free to react to a timeout as the implementers wish. It could retransmit only segment 2, but in case 2 above, we will be waiting again until segment 3 times out. In this case, we lose all of the throughput advantages of the window mechanism. Or TCP might immediately resend all of the segments in the current window.
- Whatever the choice, maximal throughput is lost. This is because the ACK does not contain a second acknowledgement sequence number indicating the actual frame received.

TCP Timers

- TCP uses several timers to ensure that excessive delays are not encountered during communications. Several of these timers are elegant, handling problems that are not immediately obvious at first analysis. The timers used by TCP are examined in the following sections, which reveal their roles in ensuring that data is properly sent from one connection to another.

Variable Timeout Intervals

- Each TCP should implement an algorithm to adapt the timeout values to be used for the round trip time of the segments. To do this, TCP records the time at which a segment was sent, and the time at which the ACK is received. A weighted average is calculated over several of these round trip times, to be used as a timeout value for the next segment(s) to be sent.
- This is an important feature, since delays may be variable on an internet, depending on multiple factors, such as the load of an intermediate low-speed network or the saturation of an intermediate IP gateway.

Picking a Timeout Value

- Timeout should be based on *round trip time* (RTT)
- Sender can't know RTT of any packet before transmission
- Sender picks *retransmission timeout* (RTO) based on *previous* RTTs
- Specific method is call *adaptive retransmission algorithm*

TCP Timers

- TCP Maintains Several Timers. Following are the important ones:
  - Connection Establishment Timer
  - Retransmission Timer
  - Delayed ACK Timer
  - Persistence Timer
  - The Keepalive Timer and Idle Timer
  - The Quiet Timer

Connection Establishment Timer

- A connection establishment timer starts when a SYN is sent to establish a new connection. If a response is not received within 75 seconds, the connection establishment is aborted.

The Retransmission Timer

- A retransmission timer is set when TCP sends data. If the data is not acknowledged by the other end when this timer expires, TCP retransmits the data. The value of this timer is calculated dynamically, based on the round-trip time measured by TCP for this connection, and based on the number of times this data segment has been retransmitted. The retransmission timer is bounded by TCP to be between 1 and 64 seconds.

The Retransmission Timer

- The retransmission timer manages retransmission timeouts (RTOs), which occur when a preset interval between the sending of a datagram and the returning acknowledgment is exceeded. The value of the timeout tends to vary, depending on the network type, to compensate for speed differences. If the timer expires, the datagram is retransmitted with an adjusted RTO, which is usually increased exponentially to a maximum preset limit. If the maximum limit is exceeded, connection failure is assumed, and error messages are passed back to the upper-layer application.

The Retransmission Timer

- Values for the retransmission timeout are determined by measuring the average time that data takes to be transmitted to another machine and the acknowledgement received back, which is called the round-trip time, or RTT. From experiments, these RTTs are averaged by a formula that develops an expected value, called the smoothed round-trip time, or SRTT. This value is then increased to account for unforeseen delays.

Delayed ACK Timer

- A delayed ACK timer is set when TCP received data that must be acknowledged, but need not be acknowledged immediately. Instead, TCP waits up to 200 ms before sending the ACK. If, during this 200 ms time period, TCP has data to send on this connection, the pending acknowledgement is sent along with the data.

The Persistence Timer

- The persistence timer handles a fairly rare occurrence. It is conceivable that a receive window might have a value of 0, causing the sending machine to pause transmission. The message to restart sending might be lost, causing an infinite delay. The persistence timer waits a preset time and then sends a one-byte segment at predetermined intervals to ensure that the receiving machine is still clogged.
- A persist timer is set when the other end of the connection advertises a window of 0, stopping TCP from sending data. Since window advertisements from the other end are not sent reliably, there's a chance that a future window update, allowing TCP to send some data, can be lost. Therefore, if TCP has data to send and the other end advertises a window of 0, the persist timer is set and when it expires, 1 byte of data is sent to see if the window has opened.

The Persistence Timer

- The receiving machine resends the zero window-size message after receiving one of these status segments, if it is still backlogged. If the window is open, a message giving the new value is returned, and communications are resumed.

The Keep-Alive Timer and the Idle Timer

- Both the keep-alive timer and the idle timer were added to the TCP specifications after their original definition. The keep-alive timer sends an empty packet at regular intervals to ensure that the connection to the other machine is still active. If no response has been received after sending the message by the time the idle timer has expired, the connection is assumed to be broken.

The Keep-Alive Timer and the Idle Timer

- A keepalive timercan be set by the process using the SO_KEEPALIVE socket option. If the connection is idle for 2 hours, the keepalive timer expires and a special segment is sent to the other end, forcing it to response. If the expected response is received, TCP knows that the other host is still up, and TCP won't probe it again until the connection is idle for another 2 hours. Other responses to the keepalive probe tell TCP that the other host has crashed and rebooted. If no response is received to a fixed number of keepalive probes, TCP assumes that the other end has crashed , although it can't distinguish between the other end being down and a temporary lack of connectivity to the other end.

The Keep-Alive Timer and the Idle Timer

- The keep-alive timer value is usually set by an application, with values ranging from 5 to 45 seconds. The idle timer is usually set to 360 seconds.

The Quiet Timer

- After a TCP connection is closed, it is possible for datagrams that are still making their way through the network to attempt to access the closed port. The quiet timer is intended to prevent the just-closed port from reopening again quickly and receiving these last datagrams.
- The quiet timer is usually set to twice the maximum segment lifetime (the same value as the Time to Live field in an IP header), ensuring that all segments still heading for the port have been discarded. Typically, this can result in a port being unavailable for up to 30 seconds, prompting error messages when other applications attempt to access the port during this interval.

TCP Sliding Window

- TCP uses sliding window for flow control
- Receiver specifies window
  - Called *window advertisement*
  - Specifies which *bytes* in the data stream can be sent

⌘Carried in segment along with ACK

⌘Sender can transmit any bytes, in any size segment, between last acknowledged byte and within window size

Silly Window Syndrome

- Under some circumstances, sliding window can result in transmission of many small segments
- If receiver window full, and receiving application consumes a few data bytes, receiver will advertise small window
- Sender will immediately send small segment to fill window
- Inefficient in processing time and network bandwidth
- Solutions:
  - Receiver delays advertising new window
  - Sender delays sending data when window is small

Congestion Control

- Excessive traffic can cause packet loss
  - Transport protocols respond with retransmission
  - Excessive retransmission can cause *congestion collapse*
- TCP interprets packet loss as an indicator of congestion
- Sender uses TCP *congestion control* and slows transmission of packets
  - Sends single packet
  - If acknowledgement returns without loss, sends two packets
- When TCP sends one-half window size, rate of increase slows

TCP Ports

- TCP uses the same port principle as UDP to provide multiplexing.
- Like UDP, TCP uses well-known and ephemeral ports.
- Each side of a TCP connection has a socket which can be identified by the triple <TCP, IP address, port number>. This is also called a half-association. If two processes are communicating over TCP, they have a logical connection that is uniquely identifiable by the two sockets involved, that is by the combination <TCP, local IP address, local port, remote IP address, remote port>. Server processes are able to manage multiple conversations through a single port.

Ports

- As some higher-level programs are themselves protocols, standardised in the TCP/IP protocol suite, such as TELNET and FTP, they use the same port number in all TCP/IP implementations. Those "assigned" port numbers are called well-known ports and the standard applications well-known services.

Well-known Ports

- The "well-known" ports are controlled and assigned by the Internet Assigned Numbers Authority (IANA) and on most systems can only be used by system processes or by programs executed by privileged users.
- The assigned "well-known" ports occupy port numbers in the range 0 to 1023. The ports with numbers in the range 1024-65535 are not controlled by the IANA and on most systems can be used by ordinary user-developed programs.

Ports

- Confusion due to two different applications trying to use the same port numbers on one host is avoided by writing those applications to request an available port from TCP/IP. Because this port number is dynamically assigned, it may differ from one invocation of an application to the next.
- UDP, TCP and ISO TP-4 all use the same "port principle". To the extent possible, the same port numbers are used for the same services on top of UDP, TCP and ISO TP-4.

Frequently Used TCP Port Numbers

- Port Number Process Name Description
- 1 TCPMUX TCP Port Service Multiplexer
- 5 RJE Remote Job Entry
- 7 ECHO Echo
- 9 DISCARD Discard
- 11 USERS Active Users
- 13 DAYTIME Daytime
- 17 Quote Quotation of the Day
- 19 CHARGEN Character generator
- 20 FTP-DATA File Transfer ProtocoloData
- 21 FTP File Transfer ProtocoloControl
- 23 TELNET Telnet
- 25 SMTP Simple Mail Transfer Protocol
- 27 NSW-FE NSW User System Front End
- 29 MSG-ICP MSG-ICP
- 31 MSG-AUTH MSG Authentication
- 33 DSP Display Support Protocol
- 35 Private Print Servers
- 37 TIME Time
- 39 RLP Resource Location Protocol

Frequently Used TCP Port Numbers

- 41 GRAPHICS Graphics
- 42 NAMESERV Host Name Server
- 43 NICNAME Who Is
- 49 LOGIN Login Host Protocol
- 53 DOMAIN Domain Name Server
- 67 BOOTPS Bootstrap Protocol Server
- 68 BOOTPC Bootstrap Protocol Client
- 69 TFTP Trivial File Transfer Protocol
- 79 FINGER Finger
- 101 HOSTNAME NIC Host Name Server
- 102 ISO-TSAP ISO TSAP
- 103 X400 X.400
- 104 X400SND X.400 SND
- 105 CSNET-NS CSNET Mailbox Name Server
- 109 POP2 Post Office Protocol v2
- 110 POP3 Post Office Protocol v3
- 111 RPC Sun RPC Portmap
- 137 NETBIOS-NS NETBIOS Name Service
- 138 NETBIOS-DG NETBIOS Datagram Service
- 139 NETBIOS-SS NETBIOS Session Service
- 146 ISO-TP0 ISO TP0
- 147 ISO-IP ISO IP
- 150 SQL-NET SQL NET
- 153 SGMP SGMP
- 156 SQLSRV SQL Service
- 160 SGMP-TRAPS SGMP TRAPS
- 161 SNMP SNMP
- 162 SNMPTRAP SNMPTRAP
- 163 CMIP-MANAGE CMIP/TCP Manager
- 164 CMIP-AGENT CMIP/TCP Agent
- 165 XNS-Courier Xerox
- 179 BGP Border Gateway Protocol

Sockets

- Let us first consider the following terminology:
  - A *socket* is a special type of *file handle* which is used by a process to request network services from the operating system.
  - A *socket address* is the triple:
    - {*protocol, local-address, local-process*}
    - In the TCP/IP suite, for example:
    - {*tcp, 193.44.234.3, 12345*}
  - A *conversation* is the communication link between two processes.

Sockets

- An *association* is the 5-tuple that completely specifies the two processes that comprise a connection:
  - {*protocol, local-address, local-process, foreign-address, foreign-process*}
  - In the TCP/IP suite, for example:
  - {*tcp, 193.44.234.3, 1500, 193.44.234.5, 21*}
  - could be a valid association.
- A *half-association* is either:
  - {*protocol, local-address, local-process*} or
  - {*protocol, foreign-address, foreign-process*}
  - which specify each half of a connection.

Sockets

- The half-association is also called a *socket* or a *transport address*. That is, a socket is an end point for communication that can be named and addressed in a network.

Sockets

- Two processes communicate via TCP sockets. The socket model provides a process with a full-duplex byte stream connection to another process. The application need not concern itself with the management of this stream; these facilities are provided by TCP.

Summary

- TCP provides end-to-end reliable bytestream delivery
- IP used for delivery to destination host
- Protocol ports demultiplex to destination application
- Additional techniques develop reliable delivery from IP messages
- Positive acknowledgement with retransmission
- Sequence numbers detect missing, duplicate and out-of-order data
- Sliding window flow control
- Three-way handshake
- Congestion control

TCP State Diagram