UNIT IV

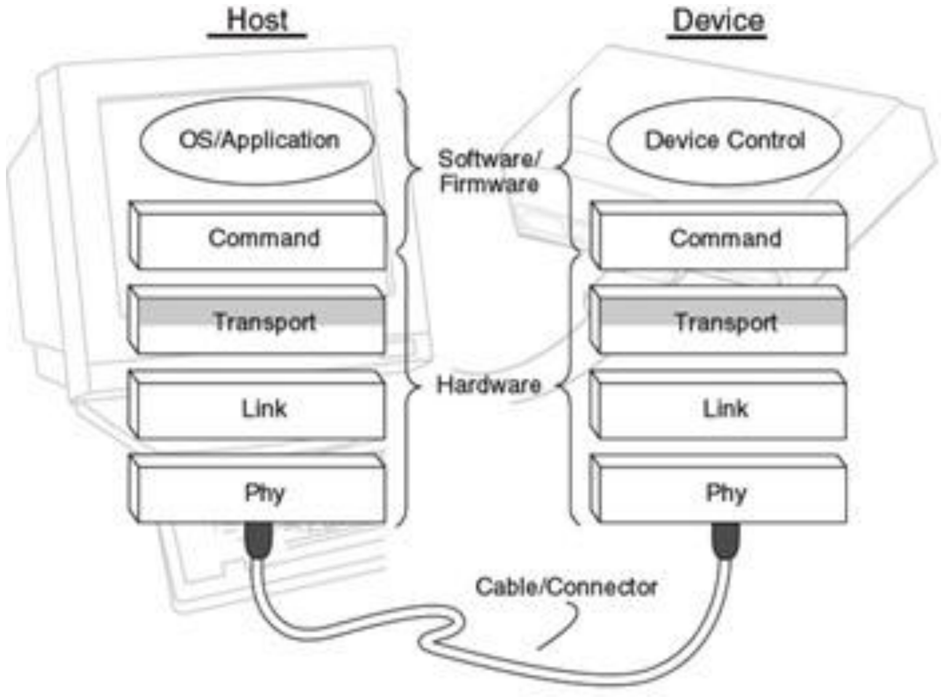
TRANSPORT LAYER

Overview of Transport layer:

In computer networking, a **transport layer** provides end-to-end or host-to-host communication services for applications within a layered architecture of network components and protocols. The transport layer provides services such as connection-oriented data stream support, reliability, flow control, and multiplexing.

Transport layer implementations are contained in both the TCP/IP model (RFC 1122), which is the foundation of the Internet, and the Open Systems Interconnection (OSI) model of general networking, however, the definitions of details of the transport layer are different in these models. In the Open Systems Interconnection model the transport layer is most often referred to as **Layer 4** or **L4**.

The best-known transport protocol is the Transmission Control Protocol (TCP). It lent its name to the title of the entire Internet Protocol Suite, TCP/IP. It is used for connection-oriented transmissions, whereas the connectionless User Datagram Protocol (UDP) is used for simpler messaging transmissions. TCP is the more complex protocol, due to its stateful design incorporating reliable transmission and data stream services. Other prominent protocols in this group are the Datagram Congestion Control Protocol (DCCP) and the Stream Control Transmission Protocol (SCTP).



Services

Transport layer services are conveyed to an application via a programming interface to the transport layer protocols. The services may include the following features:

- Connection-oriented communication: It is normally easier for an application to interpret a connection as a data stream rather than having to deal with the underlying connection-less models, such as the datagram model of the User Datagram Protocol (UDP) and of the Internet Protocol (IP).
- Same order delivery: The network layer doesn't generally guarantee that packets of data will arrive in the same order that they were sent, but often this is a desirable feature. This is usually done through the use of segment numbering, with the receiver passing them to the application in order. This can cause head-of-line blocking.
- Reliability: Packets may be lost during transport due to network congestion and errors. By means of an error detection code, such as a checksum, the transport protocol may check that the data is not corrupted, and verify correct receipt by sending an ACK or NACK message to the sender. Automatic repeat request schemes may be used to retransmit lost or corrupted data.
- Flow control: The rate of data transmission between two nodes must sometimes be managed to prevent a fast sender from transmitting more data than can be supported by the receiving data buffer, causing a buffer overrun. This can also be used to improve efficiency by reducing buffer underrun.
- Congestion avoidance: Congestion control can control traffic entry into a telecommunications network, so as to avoid congestive collapse by attempting to avoid oversubscription of any of the processing or link capabilities of the intermediate nodes and networks and taking resource reducing steps, such as reducing the rate of sending packets. For example, automatic repeat requests may keep the network in a congested state; this situation can be avoided by adding congestion avoidance to the flow control, including slow-start. This keeps the bandwidth consumption at a low level in the beginning of the transmission, or after packet retransmission.
- Multiplexing: Ports can provide multiple endpoints on a single node. For example, the name on a postal address is a kind of multiplexing, and distinguishes between different recipients of the same location. Computer applications will each listen for information on their own ports, which enables the use of more than one network service at the same time. It is part of the transport layer in the TCP/IP model, but of the session layer in the OSI model.
- A transport layer protocol provides for **logical communication** between application processes running on different hosts. By "logical" communication, we mean that although the communicating application processes are not physically connected to each other (indeed, they may be on different sides of the planet, connected via numerous routers and a wide range of link types), from the applications' viewpoint, it is as if they were physically connected. Application processes use the logical communication provided by the transport layer to send messages to each other, free for the worry of the details of the

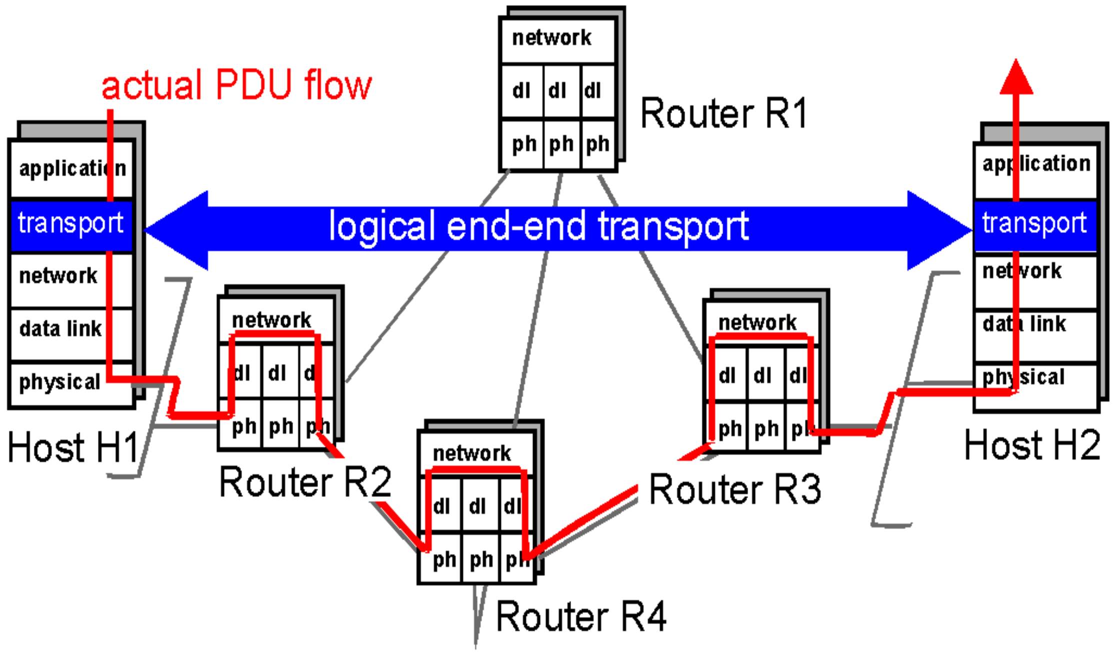
physical infrastructure used to carry these messages. Figure 3.1-1 illustrates the notion of logical communication.

- Transport layer protocols are implemented in the end systems but not in network routers. Network routers only act on the network-layer fields of the layer-3 PDUs; they do not act on the transport-layer fields.

- At the sending side, the transport layer converts the messages it receives from a sending application process into 4-PDUs (that is, transport-layer protocol data units). This is done by (possibly) breaking the application messages into smaller chunks and adding a transport-layer header to each chunk to create 4-PDUs. The transport layer then passes the 4-PDUs to the network layer, where each 4-PDU is encapsulated into a 3-PDU. At the receiving side, the transport layer receives the 4-PDUs from the network layer, removes the transport header from the 4-PDUs, reassembles the messages and passes them to a receiving application process.

- A computer network can make more than one transport layer protocol available to network applications. For example, the Internet has two protocols -- TCP and UDP. Each of these protocols provides a different set of transport layer services to the invoking application.

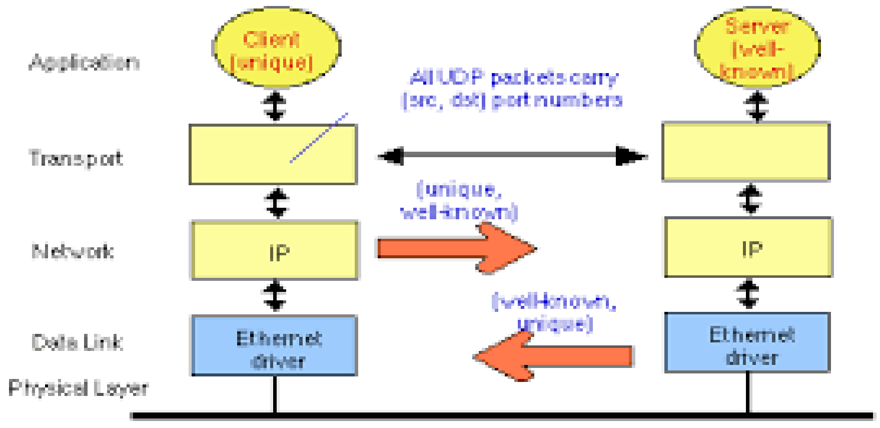
- All transport layer protocols provide an application multiplexing/demultiplexing service. This service will be described in detail in the next section. As discussed in Section 2.1, in addition to multiplexing/demultiplexing service, a transport protocol can possibly provide other services to invoking applications, including reliable data transfer, bandwidth guarantees, and delay guarantees.



UDP:

The **User Datagram Protocol** (**UDP**) is one of the core members of the Internet protocol suite. The protocol was designed by David P. Reed in 1980 and formally defined in RFC 768.

UDP uses a simple connectionless transmission model with a minimum of protocol mechanism. It has no handshaking dialogues, and thus exposes any unreliability of the underlying network protocol to the user's program. There is no guarantee of delivery, ordering, or duplicate protection. UDP provides checksums for data integrity, and port numbers for addressing different functions at the source and destination of the datagram.



With UDP, computer applications can send messages, in this case referred to as datagram, to other hosts on an Internet Protocol (IP) network without prior communications to set up special transmission channels or data paths. UDP is suitable for purposes where error checking and correction is either not necessary or is performed in the application, avoiding the overhead of such processing at the network interface level. Time-sensitive applications often use UDP because dropping packets is preferable to waiting for delayed packets, which may not be an option in a real-time system.[1] If error correction facilities are needed at the network interface level, an application may use the Transmission Control Protocol (TCP) or Stream Control Transmission Protocol (SCTP)

UDP (User Datagram Protocol) is an alternative communications protocol to Transmission Control Protocol (TCP) used primarily for establishing low-latency and loss tolerating connections between applications on the Internet. Both UDP and TCP run on top of the Internet Protocol (IP) and are sometimes referred to as UDP/IP or TCP/IP. Both protocols send short packets of data, called datagram.

UDP provides two services not provided by the IP layer. It provides port numbers to help distinguish different user requests and, optionally, a checksum capability to verify that the data arrived intact.

TCP has emerged as the dominant protocol used for the bulk of Internet connectivity owing to services for breaking large data sets into individual packets, checking for and resending lost packets and reassembling packets into the correct sequence. But these additional services come at a cost in terms of additional data overhead, and delays called latency.

UDP is an ideal protocol for network applications in which perceived latency is critical such as gaming, voice and video communications, which can suffer some data loss without adversely affecting perceived quality. In some cases, forward error correction techniques are used to improve audio and video quality in spite of some loss.

UDP can also be used in applications that require lossless data transmission when the application is configured to manage the process of retransmitting lost packets and correctly arranging received packets. This approach can help to improve the data transfer rate of large files compared with TCP.

Attributes

A number of UDP's attributes make it especially suited for certain applications.

- It is transaction-oriented, suitable for simple query-response protocols such as the Domain Name System or the Network Time Protocol.
- It provides datagram, suitable for modeling other protocols such as in IP tunneling or Remote Procedure Call and the Network File System.
- It is simple, suitable for bootstrapping or other purposes without a full protocol stack, such as the DHCP and Trivial File Transfer Protocol.
- The lack of retransmission delays makes it suitable for real-time applications such as Voice over IP, online games, and many protocols built on top of the Real Time Streaming Protocol.
- It is stateless, suitable for very large numbers of clients, such as in streaming media applications for example IPTV
- Works well in unidirectional communication, suitable for broadcast information such as in many kinds of service discovery and shared information such as broadcast time or Routing Information Protocol
- UDP provides application multiplexing (via port numbers) and integrity verification (via checksum) of the header and payload.[4] If transmission reliability is desired, it must be implemented in the user's application.

- The UDP header consists of 4 fields, each of which is 2 bytes (16 bits).[1] The use of the fields "Checksum" and "Source port" is optional in IPv4 (pink background in table). In IPv6 only the source port is optional (see below).
- Source port number
- This field identifies the sender's port when meaningful and should be assumed to be the port to reply to if needed. If not used, then it should be zero. If the source host is the client, the port number is likely to be an ephemeral port number. If the source host is the server, the port number is likely to be a well-known port number.[2]
- Destination port number
- This field identifies the receiver's port and is required. Similar to source port number, if the client is the destination host then the port number will likely be an ephemeral port number and if the destination host is the server then the port number will likely be a well-known port number.[2]
- Length
- A field that specifies the length in bytes of the UDP header and UDP data. The minimum length is 8 bytes because that is the length of the header. The field size sets a theoretical limit of 65,535 bytes (8 byte header + 65,527 bytes of data) for a UDP datagram. The practical limit for the data length which is imposed by the underlying IPv4 protocol is 65,507 bytes (65,535 − 8 byte UDP header − 20 byte IP header).[2]
- In IPv6 Jumbo grams it is possible to have UDP packets of size greater than 65,535 bytes.[5] RFC 2675 specifies that the length field is set to zero if the length of the UDP header plus UDP data is greater than 65,535.
- Checksum
- The checksum field is used for error-checking of the header and data. If no checksum is generated by the transmitter, the field uses the value all-zeros.[6] This field is not optional for IPv6.[7]

Reliable byte stream (TCP):

A **reliable byte stream** is a common service paradigm in computer networking; it refers to a byte stream in which the bytes which emerge from the communication channel at the recipient are exactly the same, and in exactly the same order, as they were when the sender inserted them into the channel.

The classic example of a reliable byte stream communication protocol is the Transmission Control Protocol, one of the major building blocks of the Internet.

A reliable byte stream is not the only reliable service paradigm which computer network communication protocols provide, however; other protocols (e.g. SCTP) provide a reliable message stream, i.e. the data is divided up into distinct units, which are provided to the consumer of the data as discrete objects.

Connection-oriented (TCP):

• Flow control: keep sender from overrunning receiver
• Congestion control: keep sender from overrunning network

Characteristics of TCP Reliable Delivery:

TCP provides a **reliable, byte-stream, full-duplex inter-process communications service** to application programs/processes. The service is **connection-oriented** and uses the concept of **port numbers** to identify processes.

Reliable

All data will be delivered correctly to the destination process, without errors, even though the underlying packet delivery service (IP) is unreliable -- see later.

Connection-oriented

Two process which desire to communicate using TCP must first request a **connection**. A connection is closed when communication is no longer desired.

Byte-stream

An application which uses the TCP service is unaware of the fact that data is broken into **segments** for transmission over the network.

Full-duplex

Once a TCP connection is established, application data can flow in both directions simultaneously -- note, however, that many application protocols do not take advantage of this.

Port Numbers

Port numbers identify processes/connections in TCP.

Edge Systems and Reliable Transport

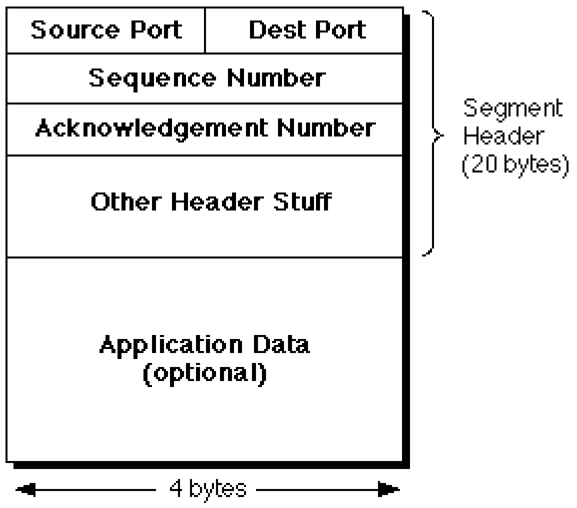
1. An **edge system** is any computer (host, printer, even a toaster...) which is "connected to" the Internet -- that is, it has access to the Internet's packet delivery system, but doesn't itself form part of that delivery system.

2. A **transport service** provides communications between application processes running on edge systems. As we have already seen, application processes communicate with each another using **application protocols** such as HTTP and SMTP. The interface between an application process and the transport service is normally provided using the **socket** mechanism.

Most application protocols require **reliable data transfer**, which in the Internet is provided by the **TCP** transport service/protocol. Note: some applications **do not** require reliability, so the unreliable **UDP** transport service/protocol is also provided as an alternative

TCP Segments

TCP slices (dices?) the incoming byte-stream data into **segments** for transmission across the Internet. A segment is a highly-structured data package consisting of an administrative **header** and some **application data**.



Source and Destination Port Numbers

We have already seen that TCP server processes wait for connections at a pre-agreed port number. At connection establishment time, TCP first allocates a **client port number** -- a port number by which the client, or initiating, process can be identified. Each segment contains both port numbers.

Segment and Acknowledgment Numbers

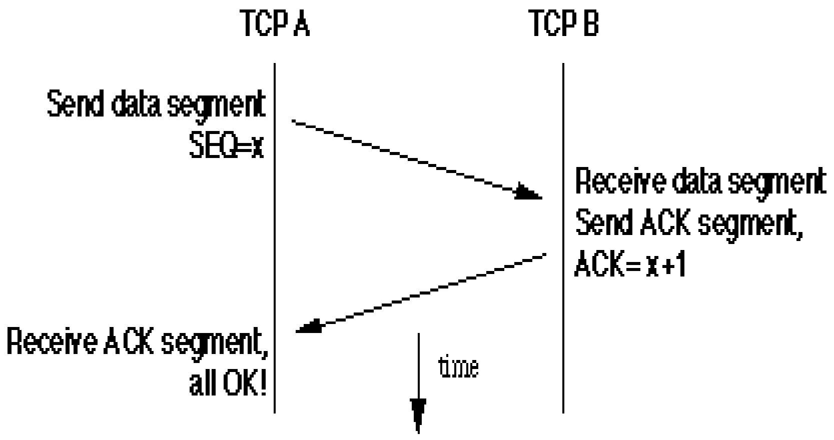
Every transmitted segment is identified with a 32-bit **Sequence number**[2], so that it can be explicitly acknowledged by the receipient. The Acknowledgment Number identifies the last segment recived by the originator of this segment.

Application Data

Optional because some segments convey only **control information** -- for example, an ACK segment has a valid acknowledgment number field, but no data. The data field can be any size up to the currently configured **MSS** for the whole segment.

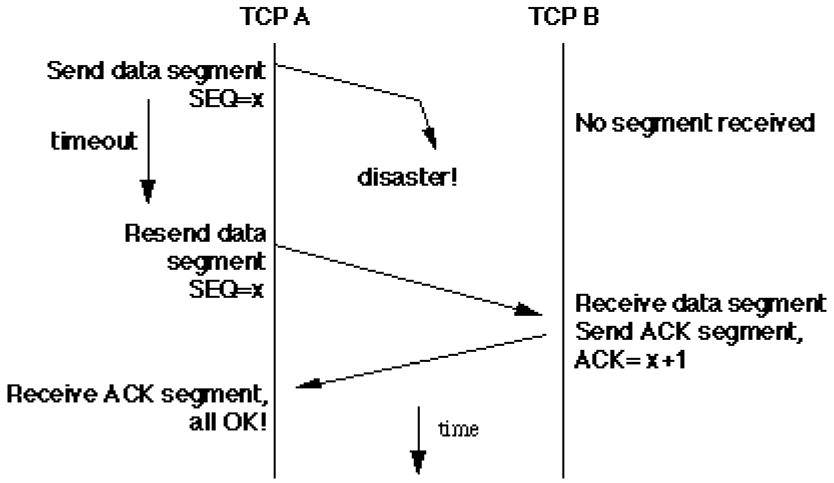
TCP Operation

When a segment is received correct and intact at its destination, an **acknowledgment** (ACK) segment is returned to the sending TCP. This ACK contains the sequence number of the last byte correctly received, incremented by 1[3]. ACKs are cumulative -- a single ACK can be sent for several segments if, for example, they all arrive within a short period of time.



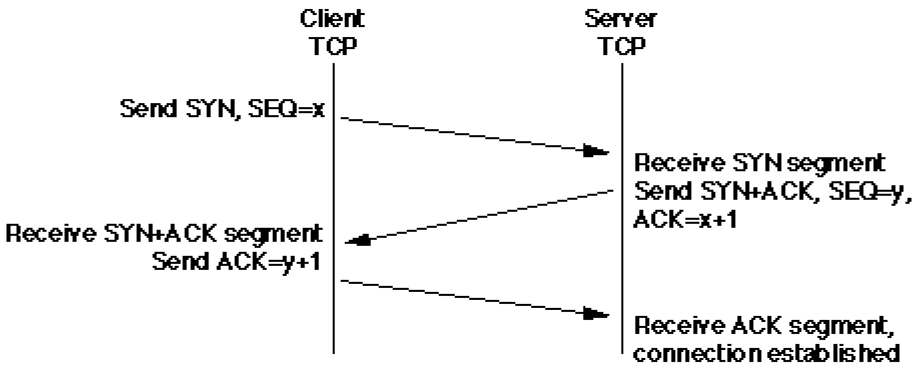
The network service can fail to deliver a segment. If the sending TCP waits for **too long**[4] for an acknowledgment, it times out and resends the segment, on the assumption that the datagram has been lost.

In addition, the network can potentially deliver duplicated segments, and can deliver segments out of order. TCP buffers or discards out of order or duplicated segments appropriately, using the byte count for identification.



TCP Connections:

An application process requests TCP to establish, or open, a (reliable) connection to a server process running on a specified edge-system, and awaiting connections at a known port number. After allocating an unused client-side port number[5], TCP initiates an exchange of connection establishment "control segments":



- This exchange of segments is called a **3-way handshake** (for obvious reasons), and is necessary because any one of the three segments can be lost, etc. The **ACK** and **SYN** segment names refer to "control bits" in the TCP header: for example, if the **ACK** bit is set, then this is an **ACK** segment.
- Each TCP chooses an **random initial sequence number** (the **x** and **y** in this example). This is crucial to the protocol's operation if there's a small chance that "old" segments (from a closed connection) might be interpreted as valid within the current connection.
- A connection is **closed** by another 3-way handshake of control segments. It's possible for a connection to be **half open** if one end requests close, and the other doesn't respond with an appropriate segment.

Optional: TCP Flow Control, Congestion Control and Slow Start

TCP attempts to make the best possible use of the underlying network, by sending data at the highest possible rate that won't cause segment loss. There are two aspects to this:

Flow Control

The two TCPs involved in a connection each maintain a **receive window** for the connection, related to the size of their **receive buffers**. For TCP "**A**", this is the maximum number of bytes that TCP "**B**" should send to it before "blocking" and waiting for an ACK. All TCP segments contain a **window** field, which is used to inform the other TCP of the sender's receive window size -- this is called "advertising a window size". At any time, for example, TCP **B** can have multiple segments "**in-flight**" -- that is, sent but not yet ACK'd -- up to TCP **A**'s advertised window.

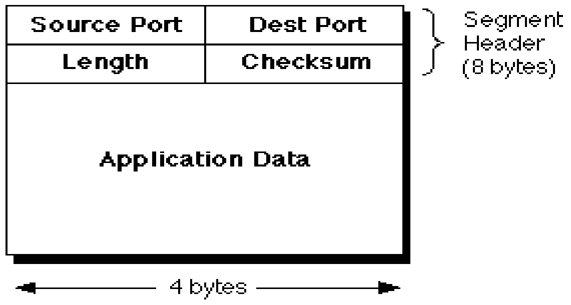
Congestion Avoidance and Control

When a connection is initially established, the TCPs know nothing at all about the speed, or capacity, of the networks which link them. The built-in "**slow start**" algorithm controls the rate at which segments are initially sent, as TCP tentatively discovers reasonable numbers for the connection's **Round Trip Time (RTT)** and its variability. TCP also slowly increases the number of segments "in-flight", since this increases the utilisation of the network.

Every TCP in the entire Internet is attempting to make full use of the available network, by increasing the number of "in-flight" segments it has outstanding. Ultimately there will come a point where the sum of the traffic, in some region of the network exceeds one or more router's buffer space, at which time segments will be dropped. When TCP "times out", and has to resend a dropped segment, it takes this as an indication that it (and all the other TCPs) have pushed the network just a little too hard. TCP immediately reduces its **congestion window** to a low value, and slowly, slowly allows it to increase again as ACKs are received.

User Datagram Protocol:

The **User Datagram Protocol (UDP)** provides an alternative, connectionless, transport service to TCP for applications where reliable stream service is not needed. UDP datagrams can be droppped, duplicated or delivered out of order, exactly as for IP.



The UDP transport service adds to IP the ability to deliver a datagram to a specified destination process using a port abstraction, in an analogous way to that used by TCP.

UDP segments (also commonly called **datagrams**, see later) have a minimal (8-byte) header. Data transfer with UDP has no initial connection overhead, and (obviously) no waiting for ACK segments as in TCP. Some typical UDP-based services include DNS, streaming multimedia and "Voice over IP" applications.

Connection management:

TCP Connection Management:
Recall:
TCP sender, receiver establish "connection" before exchanging data segments - to initialize TCP variables.

client:
connection initiaton Socket clientSocket = new Socket("hostname","port number");

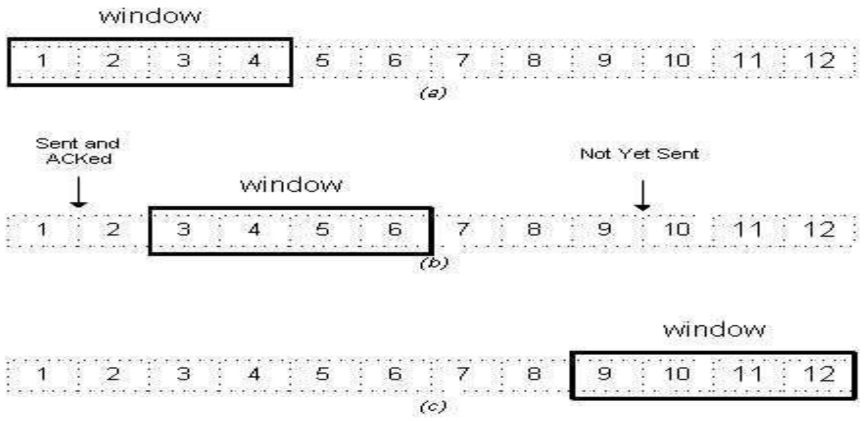
server:
Contacted by client Socket connectionSocket = welcome Socket. accept ();

Flow control:

- Based on window mechanism
- Aims at sharing the bandwidth fairly between the users
- Timeout and retransmission
- measurement of the round trip time (RTT)
- Estimating variance of RTT (Jacobson's algorithm)
- Exponential backoff of RTO
- Slow start
- Dynamic window control
- Fast retransmit
- Fast recovery
- Selective acknowledgement, SACK (an optional addition)

One of TCP's primary functions is to properly match the transmission rate of the sender to that of the receiver and the network. It is important for the transmission to be at a high enough rate to ensure good performance, but also to protect against overwhelming the network or receiving host.

TCP's 16-bit window field is used by the receiver to tell the sender how many bytes of data the receiver is willing to accept. Since the window field is limited to a maximum of 16 bits, this provides for a maximum window size of 65,535 bytes.
The window size advertised by the receiver tells the sender how much data, starting from the current position in the TCP data byte stream can be sent without waiting for further acknowledgements. As data is sent by the sender and then acknowledged by the receiver, the window slides forward to cover more data in the byte stream. This concept is known as a "sliding window".



The window boundary is eligible to be sent by the sender. Those bytes in the stream prior to the window have already been sent and acknowledged. Bytes ahead of the window have not been sent and must wait for the window to "slide" forward before they can be transmitted by the sender. A receiver can adjust the window size each time it sends acknowledgements to the sender. The maximum transmission rate is ultimately bound by the receiver's ability to accept and process data. However, this technique implies an implicit trust arrangement between the TCP sender and receiver. It has been shown that aggressive or **unfriendly** TCP software implementations can take advantage of this trust relationship to unfairly increase the transmission rate or even to intentionally cause network overload situations.

As we will see shortly, the sender and also the network can play a part in determining the transmission rate of data flow as well. It is important to consider the limitation on the window size of 65,535 bytes. Consider a typical internetwork that may have link speeds of up to 1 Gb/s or more. On a 1 Gb/s network 125,000,000 bytes can be transmitted in one second. TCP stations are communicating on this link, at best 65,535/125,000,000 or only about .0005 of the bandwidth will be used in each direction each second.

Recognizing the need for larger windows on high-speed networks, the Internet Engineering Task Force released a standard for a “window scale option” defined in RFC 1323. This standard effectively allows the window to increase from 16 to 32 bits or over 4 billion bytes of data in the window.Flow control is a function for **the control of the data flow** within an OSI layer or between adjacent layers. In other words it **limits the amount of data transmitted** by the sending transport entity to a level, or rate, that the receiver can manage.

Flow control is a good example of a protocol function that must be implemented in several layers of the OSI architecture model. At the transport level flow control will allow the transport protocol entity in a host to **restrict the flow of data over a logical connection** from the transport protocol entity in another host. However, one of the services of the network level is to **prevent congestion**. Thus the network level also uses flow control to restrict the flow of network protocol data units (NPDUs).

The flow control mechanisms used in the transport layer vary for the different classes of service. Since the different classes of service are determined by the quality of service of the underlying data network which transports the transport protocol data units (TPDUs), it is these which influence the type of flow control used.

Thus flow control becomes a much more **complex issue** at the transport layer than at lower levels like the data link level.

Two reasons for this are:

- Flow control must interact with transport users, transport entities, and the network service.
- Long and variable transmission delays between transport entities.

Flow control causes **Queuing amongst transport users, entities, and the network service**. We take a look at the four possible queues that form and what control policies are at work **here**.

The transport entity is responsible for generating one or more **transport protocol data units (TPDUs)** for passing onto the network layer. The network layer delivers the TPDUs to the receiving transport entity which then takes out the data and passes it on to the destination user. There are two reasons why the receiving transport entity would want to **control the flow of TPDUs**:

- The receiving user cannot keep up with the flow of data
- The receiving transport entity itself cannot keep up with the flow of TPDUs

When we say that a user or transport entity cannot keep up with the data flow, we mean that the **receiving buffers** are filling too quickly and will **overflow and lose data** unless the rate of incoming data is slowed.

Four possible ways to cope with the problem are:

- **Let it be and do nothing**
- **Refuse any more TPDUs from the network service**
- **Use a fixed sliding-window protocol**
- **Use a credit scheme**

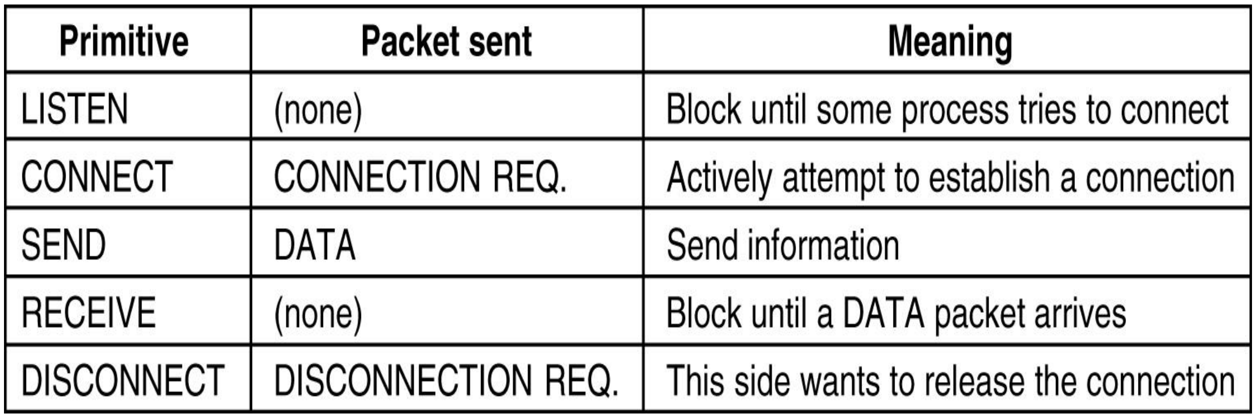
There are different issues to be considered with transport flow control over different levels of network service. The more unreliable the network service provided the more complex flow control mechanism that may be needed to be used by the Transport Layer. The credit scheme works well with the different network services although specific issues need to be addressed as with a **Reliable Non-sequencing Network Service** and an **Unreliable Network Service**.

The **credit scheme** seems most suited for flow control in the **transport layer** with all types of network service. It gives the receiver the **best control over data flow** and helps provide a smooth traffic flow. **Sequence numbering of credit allocations** handles the arrival of ACK/CREDIT TPDUs out of order, and a **window timer** will ensure **deadlock** does not occur in a network environment where TPDUs can be lost.

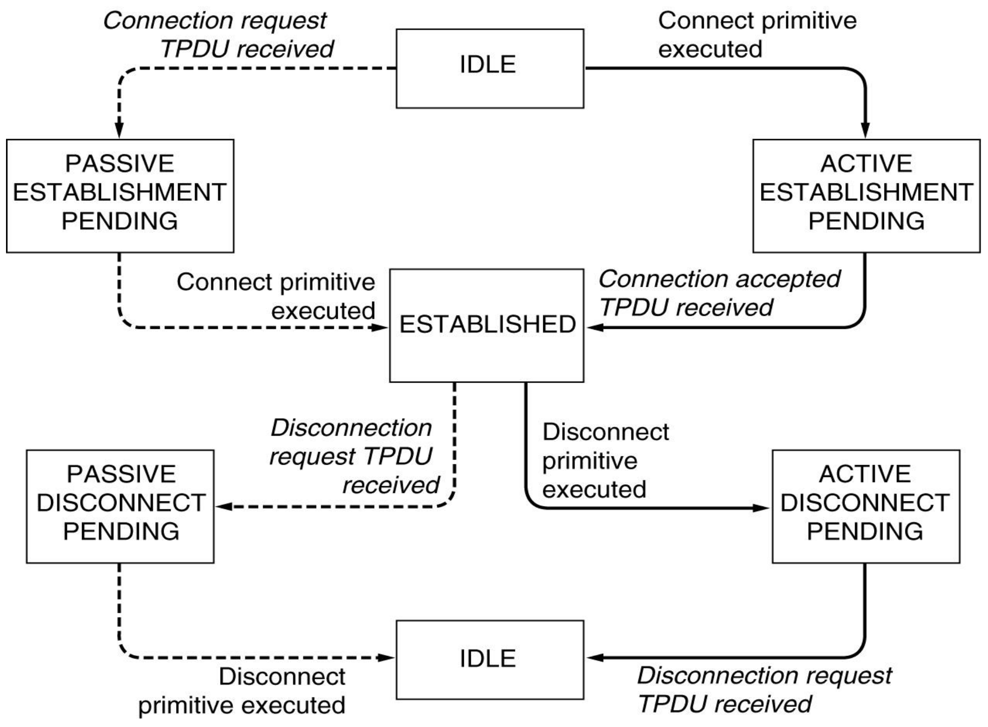
The Transport Services:

a) **Services Provided to the Upper Layers**
b) **Transport Service Primitives**
c) **Berkeley Sockets**

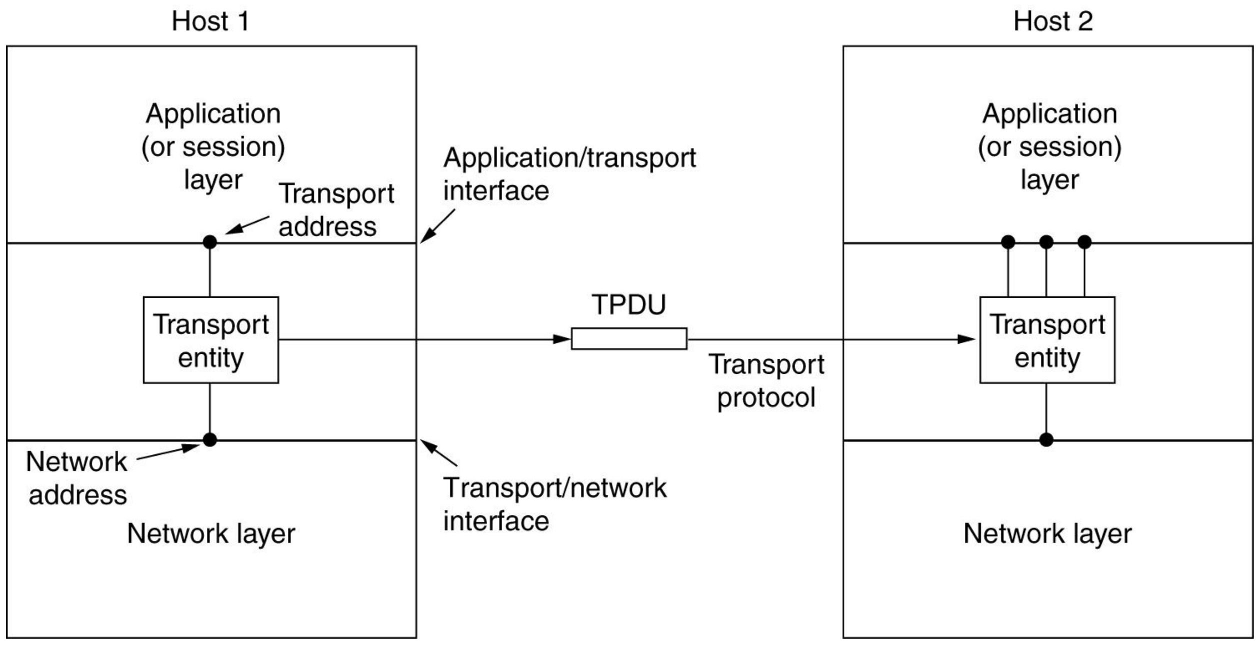
Transport Service Primitives



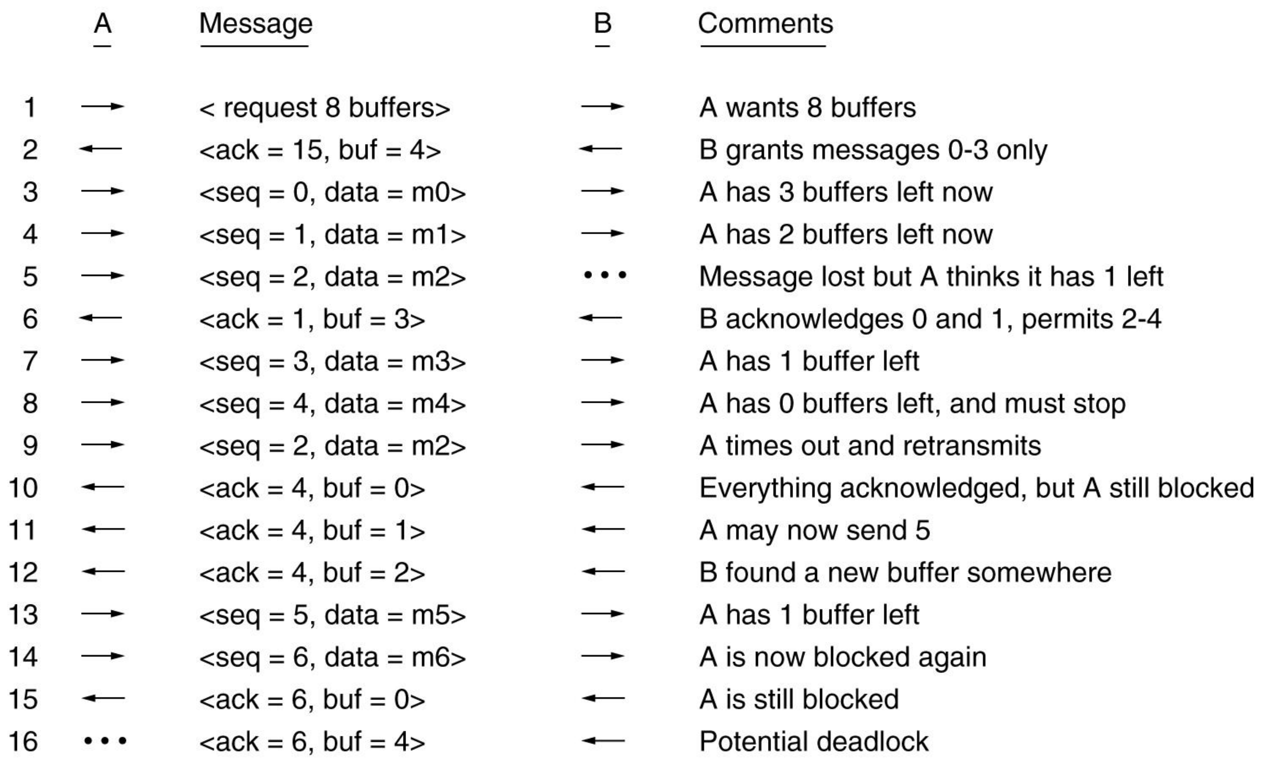
Primitive	Packet sent	Meaning
LISTEN	(none)	Block until some process tries to connect
CONNECT	CONNECTION REQ.	Actively attempt to establish a connection
SEND	DATA	Send information
RECEIVE	(none)	Block until a DATA packet arrives
DISCONNECT	DISCONNECTION REQ.	This side wants to release the connection


Connection request TPDU received
IDLE
Connect primitive executed
PASSIVE ESTABLISHMENT PENDING
ACTIVE ESTABLISHMENT PENDING
Connect primitive executed
ESTABLISHED
Connection accepted TPDU received
Disconnection request TPDU received
Disconnect primitive executed
PASSIVE DISCONNECT PENDING
ACTIVE DISCONNECT PENDING
IDLE
Disconnect primitive executed
Disconnection request TPDU received

Services Provided to the Upper Layers



Flow Control and Buffering:



	A	Message	B	Comments
1	→	< request 8 buffers>	→	A wants 8 buffers
2	←	<ack = 15, buf = 4>	←	B grants messages 0-3 only
3	→	<seq = 0, data = m0>	→	A has 3 buffers left now
4	→	<seq = 1, data = m1>	→	A has 2 buffers left now
5	→	<seq = 2, data = m2>	• • •	Message lost but A thinks it has 1 left
6	←	<ack = 1, buf = 3>	←	B acknowledges 0 and 1, permits 2-4
7	→	<seq = 3, data = m3>	→	A has 1 buffer left
8	→	<seq = 4, data = m4>	→	A has 0 buffers left, and must stop
9	→	<seq = 2, data = m2>	→	A times out and retransmits
10	←	<ack = 4, buf = 0>	←	Everything acknowledged, but A still blocked
11	←	<ack = 4, buf = 1>	←	A may now send 5
12	←	<ack = 4, buf = 2>	←	B found a new buffer somewhere
13	→	<seq = 5, data = m5>	→	A has 1 buffer left
14	→	<seq = 6, data = m6>	→	A is now blocked again
15	←	<ack = 6, buf = 0>	←	A is still blocked
16	• • •	<ack = 6, buf = 4>	←	Potential deadlock

Retransmission:

TCP is relegated to rely mostly upon implicit signals it learns from the network and remote host. TCP must make an educated guess as to the state of the network and trust the information from the remote host in order to control the rate of data flow. This may seem like an awfully tricky problem, but in most cases TCP handles it in a seemingly simple and straightforward way.

A sender's implicit knowledge of network conditions may be achieved through the use of a **timer**. For each TCP segment sent the sender expects to receive an acknowledgement within some period of time otherwise an error in the form of a timer expiring signals that that something is wrong.

Somewhere in the end-to-end path of a TCP connection a segment can be lost along the way. Often this is due to congestion in network routers where excess packets must be dropped. TCP not only must correct for this situation, but it can also **learn** something about network conditions from it.

Whenever TCP transmits a segment the sender starts a timer which keeps track of how long it takes for an acknowledgment for that segment to return. This timer is known as the **retransmission timer**. If an acknowledgement is returned before the timer expires,
which by default is often initialized to 1.5 seconds, the timer is reset with no consequence. If however an acknowledgement for the segment does not return within the timeout period, the sender would retransmit the segment and double the retransmission timer value for each consecutive timeout up to a maximum of about 64 seconds. If there are serious network problems, segments may take a few minutes to be successfully transmitted before the sender eventually times out and generates an error to the sending application.

Fundamental to the timeout and retransmission strategy of TCP is the measurement of the **round-trip time** between two communicating TCP hosts. The round-trip time may vary during the TCP connection as network traffic patterns fluctuate and as routes become available or unavailable.

A TCP option negotiated in the TCP connection establishment phase sets the number of bits by which the window is right-shifted in order to increase the value of the window. TCP keeps track of when data is sent and at what time acknowledgements covering those sent bytes are returned. TCP uses this information to calculate an estimate of round trip time. As packets are sent and acknowledged, TCP adjusts its round-trip time estimate and uses this information to come up with a reasonable timeout value for packets sent. If acknowledgements return quickly, the round-trip time is short and the retransmission timer is thus set to a lower value. This allows TCP to quickly retransmit data when network response time is good, alleviating the need for a long delay between the occasional lost segment. The converse is also true. TCP does not retransmit data too quickly during times when network response time is long.

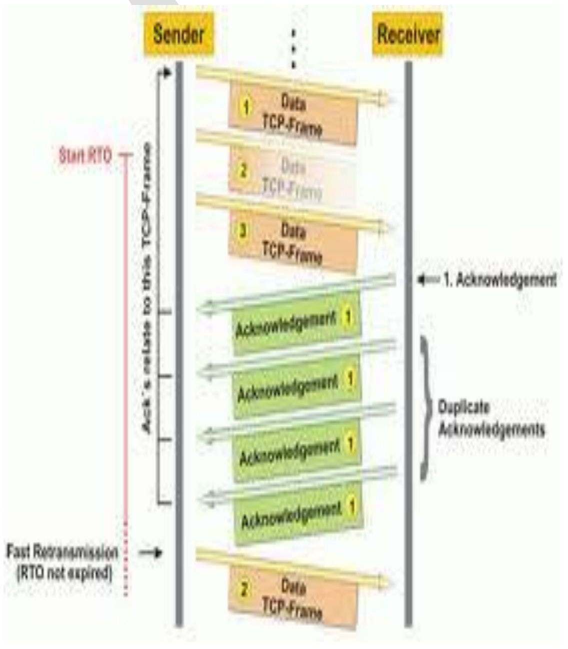
If a TCP data segment is lost in the network, a receiver will never even know it was once sent. However, the sender is waiting for an acknowledgement for that segment to return. In one case, if an acknowledgement doesn't return, the sender's retransmission timer expires which causes a retransmission of the segment. If however the sender had sent at least one additional segment after the one that was lost and that later segment is received correctly, the receiver does not send an acknowledgement for the later, out of order segment.

The receiver cannot acknowledgement out of order data; it must acknowledge the last contiguous byte it has received in the byte stream prior to the lost segment. In this case, the receiver will send an acknowledgement indicating the last contiguous byte it has received. If that last contiguous byte was already acknowledged, we call this a duplicate ACK. The reception of duplicate ACKs can implicitly tell the sender that a segment may have been lost or delayed. The sender knows this because the receiver only generates a duplicate ACK when it receives other, out of order segments. In fact, the Fast Retransmit algorithm described later uses duplicate ACKs as a way of speeding up the retransmission process.

A TCP sender uses a timer to recognize lost segments. If an acknowledgement is not received for a particular segment within a specified time (a function of the estimated Round-trip delay time), the sender will assume the segment was lost in the network, and will retransmit the segment.

Duplicate acknowledgement is the basis for the fast retransmit mechanism which works as follows: after receiving a packet (e.g. with sequence number 1), the receiver sends an acknowledgement by adding 1 to the sequence number (i.e., acknowledgement number 2) which means that the receiver receives the packet number 1 and it expects packet number 2 from the sender. Let's assume that three subsequent packets have been lost. In the meantime the receiver receives packet numbers 5 and 6. After receiving packet number 5, the receiver sends an acknowledgement, but still only for sequence number 2. When the receiver receives packet number 6, it sends yet another acknowledgement value of 2. Because the sender receives more than one acknowledgement with the same sequence number (2 in this example) this is called duplicate acknowledgement.

The fast retransmit enhancement works as follows: if a TCP sender receives a specified number of acknowledgements which is usually set to three duplicate acknowledgements with the same acknowledge number (that is, a total of four acknowledgements with the same acknowledgement number), the sender can be reasonably confident that the segment with the next higher sequence number was dropped, and will not arrive out of order. The sender will then retransmit the packet that was presumed dropped before waiting for its timeout.



TCP Congestion control:

The standard fare in TCP implementations today can be found in RFC 2581 [2]. This reference document specifies four standard congestion control algorithms that are now in common use. Each of the algorithms noted within that document was actually designed long before the standard was published [9], [11]. Their usefulness has passed the test of time.

The four algorithms, Slow Start, Congestion Avoidance, Fast Retransmit and Fast Recovery are described below.

Slow Start

Slow Start, a requirement for TCP software implementations is a mechanism used by the sender to control the transmission rate, otherwise known as sender-based flow control. This is accomplished through the return rate of acknowledgements from the receiver. Inother words, the rate of acknowledgements returned by the receiver determine the rate at which the sender can transmit data.

When a TCP connection first begins, the Slow Start algorithm initializes a **congestion window** to one segment, which is the maximum segment size (MSS) initialized by the receiver during the connection establishment phase. When acknowledgements are returned by the receiver, the congestion window increases by one segment for each acknowledgement returned. Thus, the sender can transmit the minimum of the congestion window and the advertised window of the receiver, which is simply called the **transmission window**.

Slow Start is actually not very slow when the network is not congested and network response time is good. For example, the first successful transmission and acknowledgement of a TCP segment increases the window to two segments. After successful transmission of these two segments and acknowledgements completes, the window is increased to four segments. Then eight segments, then sixteen segments and so on, doubling from there on out up to the maximum window size advertised by the receiver or until congestion finally does occur.

At some point the congestion window may become too large for the network or network conditions may change such that packets may be dropped. Packets lost will trigger a timeout at the sender. When this happens, the sender goes into congestion avoidance mode as described in the next section.

Congestion avoidance:

During the initial data transfer phase of a TCP connection the Slow Start algorithm is used. However, there may be a point during Slow Start that the network is forced to drop one or more packets due to overload or congestion. If this happens, Congestion Avoidance is used to slow the transmission rate. However, Slow Start is used in conjunction with Congestion Avoidance as the means to get the data transfer going again so it doesn't slow down and stay slow.

In the Congestion Avoidance algorithm a retransmission timer expiring or the reception of duplicate ACKs can implicitly signal the sender that a network congestion situation is occurring. The sender immediately sets its transmission window to one half of the current window size (the minimum of the congestion window and the receiver's advertised window size), but to at least two segments. If congestion was indicated by a timeout, the congestion window is reset to one segment, which automatically puts the sender into Slow Start mode. If congestion was indicated by duplicate ACKs, the Fast Retransmit and Fast Recovery algorithms are invoked (see below).

As data is received during Congestion Avoidance, the congestion window is increased. However, Slow Start is only used up to the halfway point where congestion originally occurred. This halfway point was recorded earlier as the new transmission window. After this halfway point, the congestion window is increased by one segment for all segments in the transmission window that are acknowledged. This mechanism will force the sender to more slowly grow its transmission rate, as it will approach the point where congestion had previously been detected.

DECbit:

DECbit is a technique implemented in routers to avoid congestion. Its utility is to predict possible congestion and prevent it. This protocol works with TCP.

When a router wants to signal congestion to the sender, it adds a bit in the header of packets sent. When a packet arrives at the router, the router calculates the average queue length for the last (busy + idle) period plus the current busy period. (The router is busy when it is transmitting packets, and idle otherwise). When the average queue length exceeds 1, then the router sets the congestion indication bit in the packet header of arriving packets.

When the destination replies, the corresponding ACK includes a bit of congestion. The sender receives the ACK and calculates how many packets it received with the congestion indication bit set to one. If less than half of the packets in the last window had the congestion indication bit set, then the window is increased linearly. Otherwise, the window is decreased exponentially.

This technique provides distinct advantages:

- Dynamically manages the window to avoid congestion and increasing freight if it detects congestion.
- Try to balance bandwidth with respect to the delay.

Note that this technique does not allow for effective use of the line, because it fails to take advantage of the available bandwidth. Besides, the fact that the tail has increased in size from one cycle to another does not always mean there is congestion.

DECbit (Destination Experiencing Congestion Bit) Developed for the Digital Network Architecture Basic idea One bit allocated in packet header Any router experiencing congestion sets bit Destination returns bit to source Source adjusts rate based on bits Note that responsibility is shared Routers identify congestion Hosts act to avoid congestion Key Questions:

- When should router signal congestion?
- How should end hosts react?

Congestion avoidance RED :

Random early detection (**RED**), also known as **random early discard** or **random early drop** is a queueing discipline for a network scheduler suited for congestion avoidance. In the conventional tail drop algorithm, a router or other network component buffers as many packets as it can, and simply drops the ones it cannot buffer. If buffers are constantly full, the network is congested. Tail drop distributes buffer space unfairly among traffic flows. Tail drop can also lead to TCP global synchronization as all TCP connections "hold back" simultaneously, and then step forward simultaneously. Networks become under-utilized and flooded by turns. RED addresses these issues.

Operation

RED monitors the average queue size and drops (or marks when used in conjunction with ECN) packets based on statistical probabilities. If the buffer is almost empty, all incoming packets are accepted. As the queue grows, the probability for dropping an incoming packet grows too. When the buffer is full, the probability has reached 1 and all incoming packets are dropped.

RED is more fair than tail drop, in the sense that it does not possess a bias against bursty traffic that uses only a small portion of the bandwidth. The more a host transmits, the more likely it is that its packets are dropped as the probability of a host's packet being dropped is proportional to the amount of data it has in a queue. Early detection helps avoid TCP global synchronization.

- RED provides congestion avoidance by controlling the queue size at the gateway.
- RED notifies the source before the congestion actually happens rather than wait till it actually occurs.
- RED provides a mechanism for the gateway to provide some feedback to the source on congestion status.

Advantages of RED gateways:

- Congestion Avoidance
  - If the RED gateway drops packets when avgQ reached maxQ, the avgQ will never exceed maxQ.
- Appropriate time scales
  - Source will not be notified of transient congestion.
- No Global Synchronization.
  - All connection wont back off at same time.
- Simple
- High link utilization
- Fair

QoS:

Quality of service (**QoS**) is the overall performance of a telephony or computer network, particularly the performance seen by the users of the network.To quantitatively measure quality of service, several related aspects of the network service are often considered, such as error rates, bandwidth, throughput, transmission delay, availability, jitter, etc.

Quality of service is particularly important for the transport of traffic with special requirements. In particular, much technology has been developed to allow computer networks to become as useful as telephone networks for audio conversations, as well as supporting new applications with even stricter service demands.

In the field of telephony, quality of service was defined by the ITU in 1994. Quality of service comprises requirements on all the aspects of a connection, such as service response time, loss, signal-to-noise ratio, crosstalk, echo, interrupts, frequency response, loudness levels, and so on. A subset of telephony QoS is grade of service (GoS) requirements, which comprises aspects of a connection relating to capacity and coverage of a network, for example guaranteed maximum blocking probability and outage probability.

In the field of computer networking and other packet-switched telecommunication networks, the traffic engineering term refers to resource reservation control mechanisms rather than the achieved service quality. Quality of service is the ability to provide different priority to different applications, users, or data flows, or to guarantee a certain level of performance to a data flow.

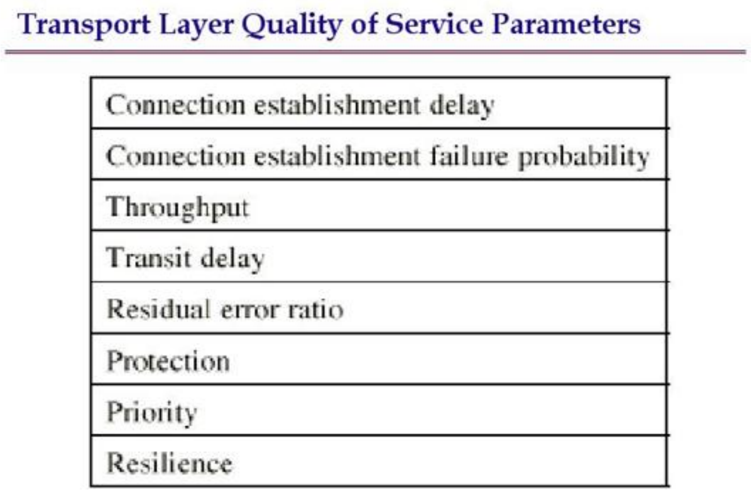
For example, a required bit rate, delay, jitter, packet dropping probability and/or bit error rate may be guaranteed. Quality of service guarantees are important if the network capacity is insufficient, especially for real-time streaming multimedia applications such as voice over IP, online games and IP-TV, since these often require fixed bit rate and are delay sensitive, and in networks where the capacity is a limited resource, for example in cellular data communication.

A network or protocol that supports QoS may agree on a traffic contract with the application software and reserve capacity in the network nodes, for example during a session establishment phase. During the session it may monitor the achieved level of performance, for example the data rate and delay, and dynamically control scheduling priorities in the network nodes. It may release the reserved capacity during a tear down phase.

A best-effort network or service does not support quality of service. An alternative to complex QoS control mechanisms is to provide high quality communication over a best-effort network by over-provisioning the capacity so that it is sufficient for the expected peak traffic load. The resulting absence of network congestion eliminates the need for QoS mechanisms.

QoS is sometimes used as a quality measure, with many alternative definitions, rather than referring to the ability to reserve resources. Quality of service sometimes refers to the level of quality of service, i.e. the guaranteed service quality.[3] High QoS is often confused with a high level of performance or achieved service quality, for example high bit rate, low latency and low bit error probability.

An alternative and disputable definition of QoS, used especially in application layer services such as telephony and streaming video, is requirements on a metric that reflects or predicts the subjectively experienced quality. In this context, QoS is the acceptable cumulative effect on subscriber satisfaction of all imperfections affecting the service. Other terms with similar meaning are the quality of experience (QoE) subjective business concept, the required "user perceived performance",[4] the required "degree of satisfaction of the user" or the targeted "number of happy customers". Examples of measures and measurement methods are mean opinion score (MOS), perceptual speech quality measure (PSQM) and perceptual evaluation of video quality (PEVQ). See also Subjective video quality.



Transport Layer Quality of Service Parameters

Connection establishment delay
Connection establishment failure probability
Throughput
Transit delay
Residual error ratio
Protection
Priority
Resilience

Qualities of traffic

In packet-switched networks, quality of service is affected by various factors, which can be divided into "human" and "technical" factors. Human factors include: stability of service, availability of service, delays, user information. Technical factors include: reliability, scalability, effectiveness, maintainability, grade of service, etc.

Many things can happen to packets as they travel from origin to destination, resulting in the following problems as seen from the point of view of the sender and receiver:

Low throughput

Due to varying load from disparate users sharing the same network resources, the bit rate (the maximum throughput) that can be provided to a certain data stream may be too low for realtime multimedia services if all data streams get the same scheduling priority.

Dropped packets

The routers might fail to deliver (drop) some packets if their data loads are corrupted, or the packets arrive when the router buffers are already full. The receiving application may ask for this information to be retransmitted, possibly causing severe delays in the overall transmission.

Errors

Sometimes packets are corrupted due to bit errors caused by noise and interference, especially in wireless communications and long copper wires. The receiver has to detect this and, just as if the packet was dropped, may ask for this information to be retransmitted.

Latency

It might take a long time for each packet to reach its destination, because it gets held up in long queues, or it takes a less direct route to avoid congestion. This is different from throughput, as the delay can build up over time, even if the throughput is almost normal. In some cases, excessive latency can render an application such as VoIP or online gaming unusable.

Jitter

Packets from the source will reach the destination with different delays. A packet's delay varies with its position in the queues of the routers along the path between source and destination and this position can vary unpredictably. This variation in delay is known as jitter and can seriously affect the quality of streaming audio and/or video.

Out-of-order delivery

When a collection of related packets is routed through a network, different packets may take different routes, each resulting in a different delay. The result is that the packets arrive in a different order than they were sent. This problem requires special additional protocols responsible for rearranging out-of-order packets to an isochronous state once they reach their destination. This is especially important for video and VoIP streams where quality is dramatically affected by both latency and lack of sequence.

Applications

A defined quality of service may be desired or required for certain types of network traffic, for example:

- Streaming media specifically
  - Internet protocol television (IPTV)
  - Audio over Ethernet
  - Audio over IP
- IP telephony also known as Voice over IP (VoIP)
- Videoconferencing
- Tele-presence
- Storage applications such as iSCSI and FCoE
- Circuit Emulation Service
- Safety-critical applications such as remote surgery where availability issues can be hazardous
- Network operations support systems either for the network itself, or for customers' business critical needs
- Online games where real-time lag can be a factor
- Industrial control systems protocols such as Ethernet/IP which are used for real-time control of machinery

These types of service are called inelastic, meaning that they require a certain minimum level of bandwidth and a certain maximum latency to function. By contrast, elastic applications can take advantage of however much or little bandwidth is available. Bulk file transfer applications that rely on TCP are generally elastic.

Application Layer

At the top of the TCP/IP protocol architecture is the Application Layer . This layer includes all processes that use the Transport Layer protocols to deliver data. There are many applications protocols. Most provide user services, and new services are always being added to this layer.

The most widely known and implemented applications protocols are:

Telnet

The Network Terminal Protocol, which provides remote login over the network.

FTP

The File Transfer Protocol, which is used for interactive file transfer.

SMTP

The Simple Mail Transfer Protocol, which delivers electronic mail.

HTTP

The Hypertext Transfer Protocol, which delivers Web pages over the network.

While HTTP, FTP, SMTP, and telnet are the most widely implemented TCP/IP applications, you will work with many others as both a user and a system administrator. Some other commonly used TCP/IP applications are:

Domain Name Service (DNS)

Also called name service , this application maps IP addresses to the names assigned to network devices. DNS is discussed in detail in this book.

Open Shortest Path First (OSPF)

Routing is central to the way TCP/IP works. OSPF is used by network devices to exchange routing information. Routing is also a major topic of this book.

Network File system (NFS)

This protocol allows files to be shared by various hosts on the network.Some protocols, such as telnet and FTP, can only be used if the user has some knowledge of the network. Other protocols, like OSPF, run without the user even knowing that they exist. As system administrator, you are aware of all these applications and all the protocols in the other TCP/IP layers.