UNIT-II

INTERNETWORKING:

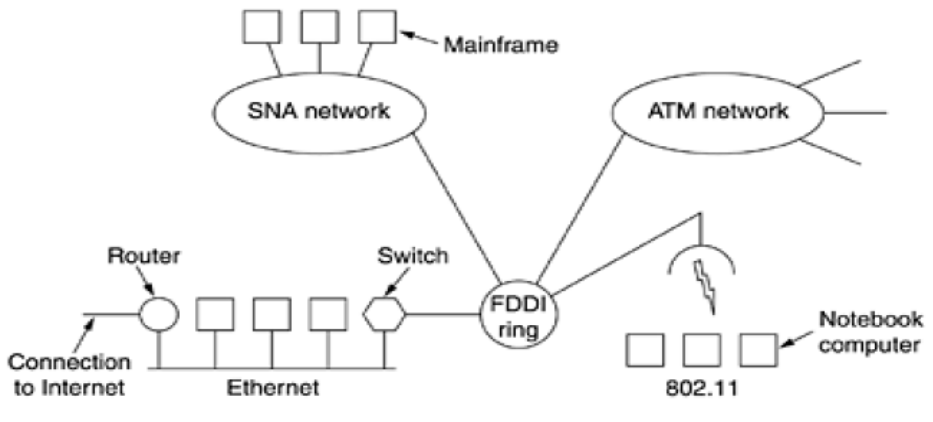
We believe that a variety of different networks will always be around, for the following reasons.

First of all, the installed base of different networks is large. Nearly all personal computers run TCP/IP. Many large businesses have mainframes running IBM's SNA. A substantial number of telephone companies operate ATM networks. Some personal computer LANs still use Novell NCP/IPX or AppleTalk. Finally, wireless is an up-and-coming area with a variety of protocols.

Second, as computers and networks get cheaper, the place where decisions get made moves downward in organizations. Many companies have a policy to the effect that purchases costing over a million dollars have to be approved by top management; purchases costing over 100,000 dollars have to be approved by middle management, but purchases under 100,000 dollars can be made by department heads without any higher approval. This can easily lead to the engineering department installing UNIX workstations running TCP/IP and the marketing department installing Macs with AppleTalk.

Third, different networks have radically different technology, so it should not be surprising that as new hardware developments occur, new software will be created to fit the new hardware.

Here we see a corporate network with multiple locations tied together by a wide area ATM network. At one of the locations, an FDDI optical backbone is used to connect an Ethernet, an 802.11 wireless LAN, and the corporate data center's SNA mainframe network.

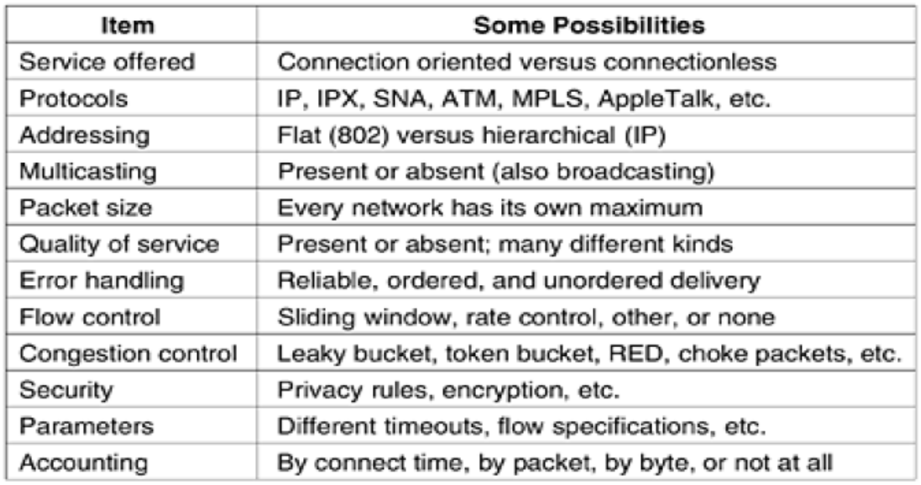


A collection of interconnected networks

The purpose of interconnecting all these networks is to allow users on any of them to communicate with users on all the other ones and also to allow users on any of them to access data on any of them.

1. How Networks Differ

Networks can differ in many ways. Some of the differences, such as different modulation techniques or frame formats, are in the physical and data link layers.



Item	Some Possibilities
Service offered	Connection oriented versus connectionless
Protocols	IP, IPX, SNA, ATM, MPLS, AppleTalk, etc.
Addressing	Flat (802) versus hierarchical (IP)
Multicasting	Present or absent (also broadcasting)
Packet size	Every network has its own maximum
Quality of service	Present or absent; many different kinds
Error handling	Reliable, ordered, and unordered delivery
Flow control	Sliding window, rate control, other, or none
Congestion control	Leaky bucket, token bucket, RED, choke packets, etc.
Security	Privacy rules, encryption, etc.
Parameters	Different timeouts, flow specifications, etc.
Accounting	By connect time, by packet, by byte, or not at all

Some of the many ways networks can differ

2. How Networks Can Be Connected

Networks can be interconnected by different devices. In the physical layer, networks can be connected by repeaters or hubs.One layer up we find bridges and switches, which operate at the data link layer. They can accept frames, examine the MAC addresses, and forward the frames to a different network while doing minor protocol translation in the process.

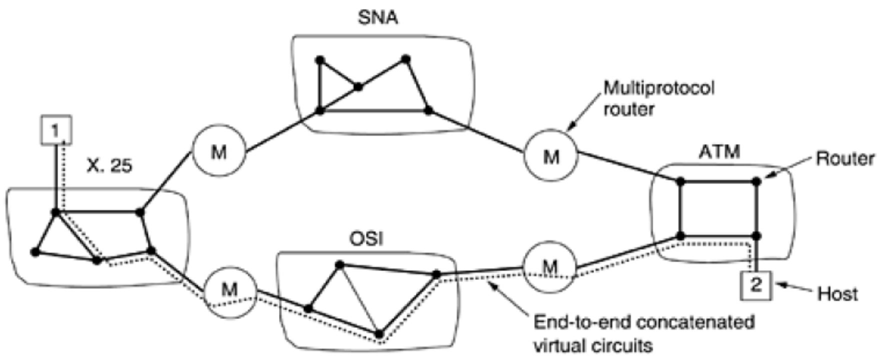
In the network layer, we have routers that can connect two networks. If two networks have dissimilar network layers, the router may be able to translate between the packet formats, although packet translation is now increasingly rare. A router that can handle multiple protocols is called a **multiprotocol router**.

In the transport layer we find transport gateways, which can interface between two transport connections. Finally, in the application layer, application gateways translate message semantics.

3. Concatenated Virtual Circuits

Two styles of internetworking are possible: a connection-oriented concatenation of virtual-circuit subnets, and a datagram internet style.

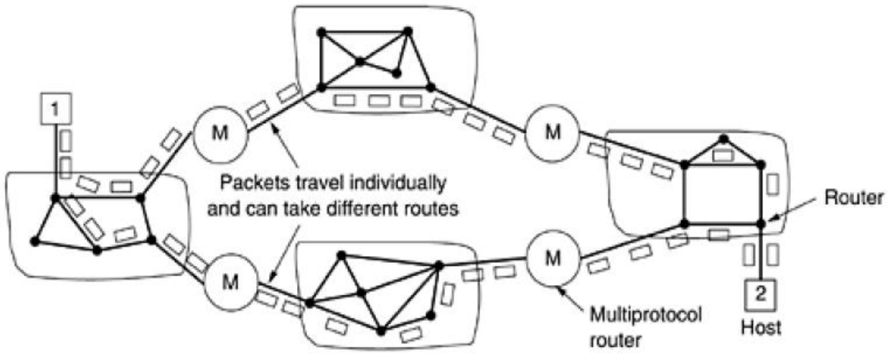
In the concatenated virtual-circuit model, shown in figure, a connection to a host in a distant network is set up in a way similar to the way connections are normally established. The subnet sees that the destination is remote and builds a virtual circuit to the router nearest the destination network. Then it constructs a virtual circuit from that router to an external **gateway** (multiprotocol router). This gateway records the existence of the virtual circuit in its tables and proceeds to build another virtual circuit to a router in the next subnet. This process continues until the destination host has been reached.



Internetworking using concatenated virtual circuits

4. Connectionless Internetworking

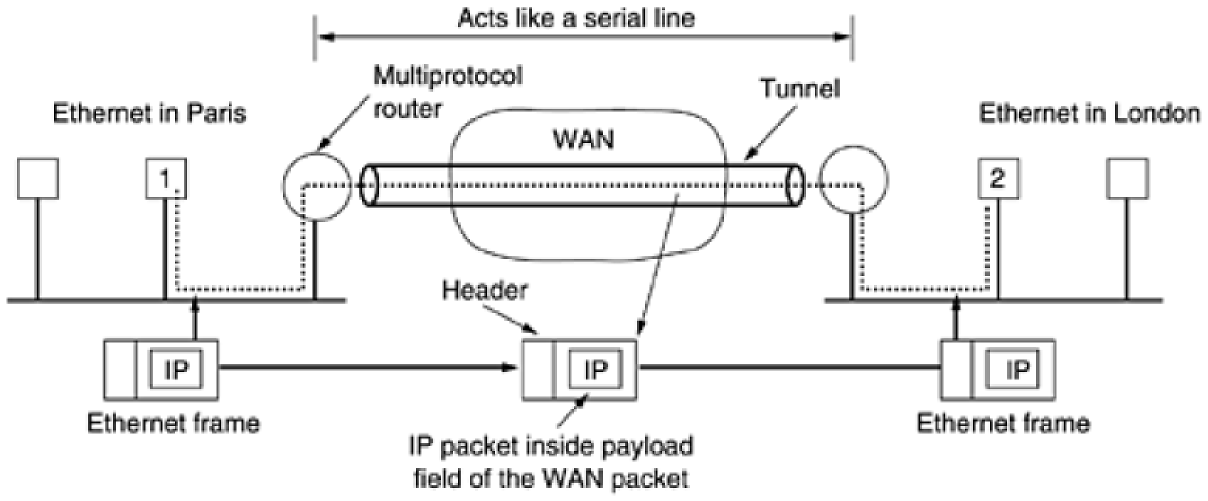
The alternative internetwork model is the datagram model. In this model, the only service the network layer offers to the transport layer is the ability to inject datagrams into the subnet and hope for the best. There is no notion of a virtual circuit at all in the network layer. This model does not require all packets belonging to one connection to traverse the same sequence of gateways. In the figure_datagrams from host 1 to host 2 are shown taking different routes through the internetwork. A routing decision is made separately for each packet, possibly depending on the traffic at the moment the packet is sent. This strategy can use multiple routes and thus achieve a higher bandwidth than the concatenated virtual-circuit model. On the other hand, there is no guarantee that the packets arrive at the destination in order, assuming that they arrive at all.



A connectionless internet

5. Tunneling

Handling the general case of making two different networks interwork is exceedingly difficult. However, there is a common special case that is manageable. This case is where the source and destination hosts are on the same type of network, but there is a different network in between. Consider the below example.

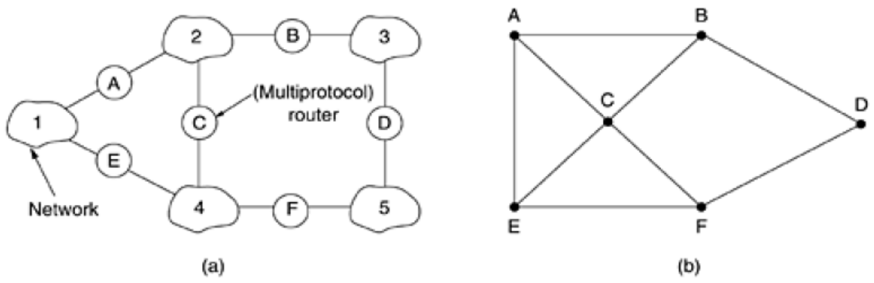


Tunneling a packet from Paris to London

The solution to this problem is a technique called **tunneling**. To send an IP packet to host 2, host 1 constructs the packet containing the IP address of host 2, inserts it into an Ethernet frame addressed to the Paris multiprotocol router, and puts it on the Ethernet. When the multiprotocol router gets the frame, it removes the IP packet, inserts it in the payload field of the WAN network layer packet, and addresses the latter to the WAN address of the London multiprotocol router. When it gets there, the London router removes the IP packet and sends it to host 2 inside an Ethernet frame.

6. Internetwork Routing

Routing through an internetwork is similar to routing within a single subnet, but with some added complications. Consider, for example, the internetwork of below figure (a) in which five networks are connected by six routers. Making a graph model of this situation is complicated by the fact that every router can directly access (i.e., send packets to) every other router connected to any network to which it is connected This leads to the graph of figure (b).



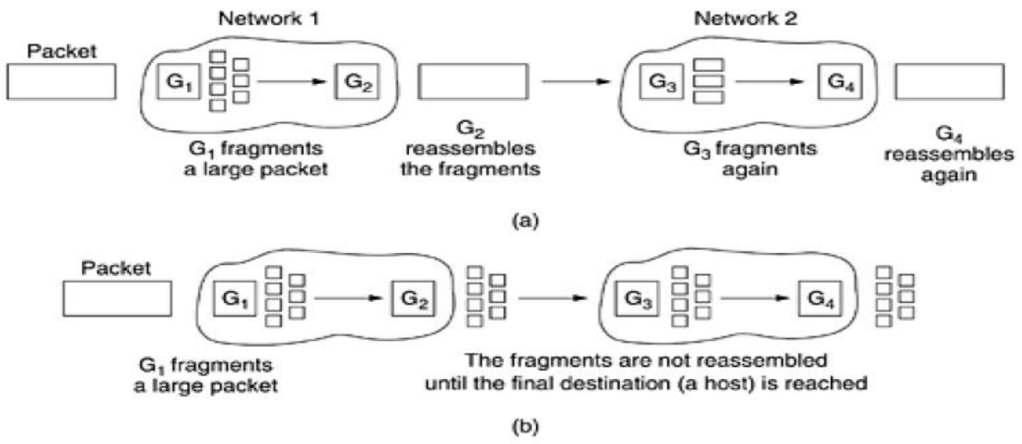
(a) An internetwork. (b) A graph of the internetwork

7. Fragmentation

Each network imposes some maximum size on its packets. These limits have various causes, among them:

1. Hardware (e.g., the size of an Ethernet frame).
2. Operating system (e.g., all buffers are 512 bytes).
3. Protocols (e.g., the number of bits in the packet length field).
4. Compliance with some (inter)national standard.
5. Desire to reduce error-induced retransmissions to some level.
6. Desire to prevent one packet from occupying the channel too long.

Basically, the only solution to the problem is to allow gateways to break up packets into **fragments**, sending each fragment as a separate internet packet. Two opposing strategies exist for recombining the fragments back into the original packet. The first strategy is to make fragmentation caused by a "small-packet" network transparent to any subsequent networks through which the packet must pass on its way to the ultimate destination. This option is shown in figure (1).

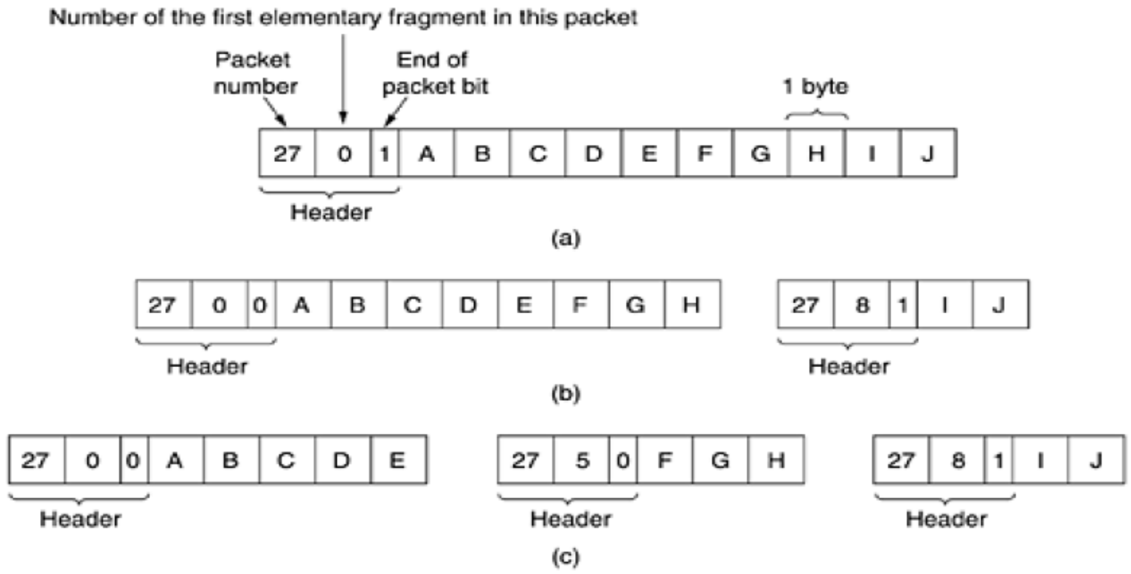

Network 1
Network 2
Packet
G1
G2
G3
G4
G1 fragments a large packet
G2 reassembles the fragments
G3 fragments again
G4 reassembles again
(a)
Packet
G1 fragments a large packet
The fragments are not reassembled until the final destination (a host) is reached
(b)

(1) Transparent fragmentation. (2) Nontransparent fragmentation

The other fragmentation strategy is to refrain from recombining fragments at any intermediate gateways. Once a packet has been fragmented, each fragment is treated as though it were an original packet. All fragments are passed through the exit gateway (or gateways), as shown in figure (2). Recombination occurs only at the destination host.

A completely different (and better) numbering system is for the internetwork protocol to define an elementary fragment size small enough that the elementary fragment can pass through every network. When a packet is fragmented, all the pieces are equal to the elementary fragment size except the last one, which may be shorter. An internet packet may contain several fragments, for efficiency reasons. The internet header must provide the original packet number and the number of the (first) elementary fragment contained in the packet. As usual, there must also be a bit indicating that the last elementary fragment contained within the internet packet is the last one of the original packet.

This approach requires two sequence fields in the internet header: the original packet number and the fragment number. There is clearly a trade-off between the size of the elementary fragment and the number of bits in the fragment number. Because the elementary fragment size is presumed to be acceptable to every network, subsequent fragmentation of an internet packet containing several fragments causes no problem. The ultimate limit here is to have the elementary fragment be a single bit or byte, with the fragment number then being the bit or byte offset within the original packet.



Fragmentation when the elementary data size is 1 byte. (a) Original packet, containing 10 data bytes. (b) Fragments after passing through a network with maximum packet size of 8 payload bytes plus header. (c) Fragments after passing through a size 5 gateway

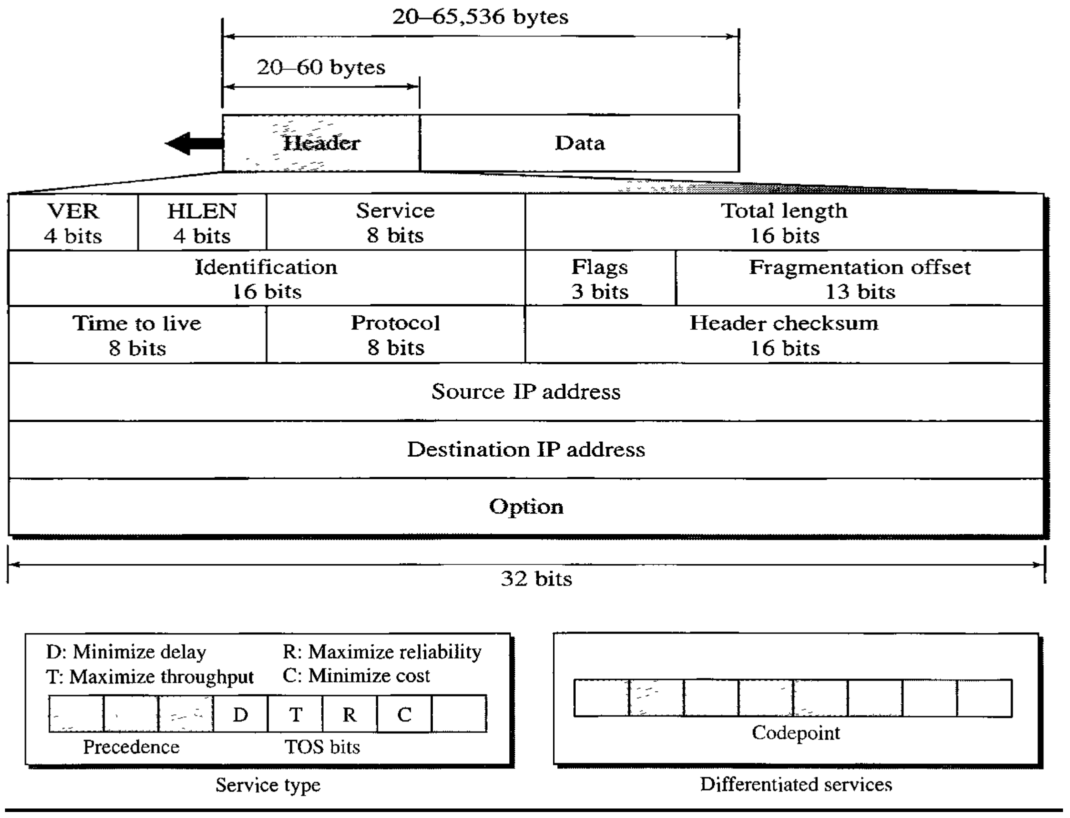
THE NETWORK LAYER IN THE INTERNET:

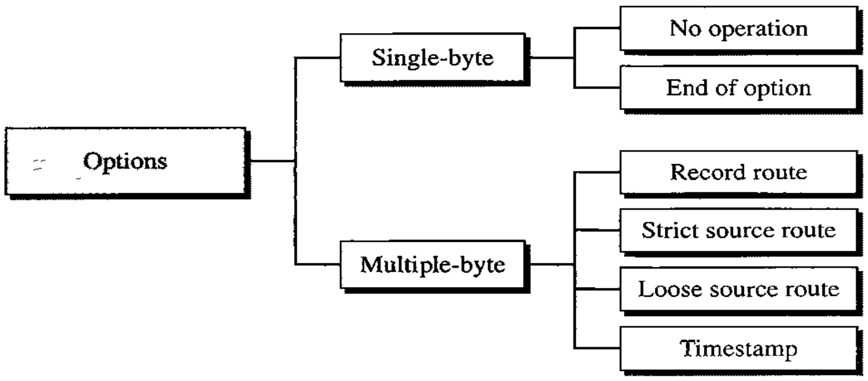
The principles that drove its design in the past and made it the success that it is today.

1. Make sure it works
2. Keep it simple
3. Make clear choices
4. Exploit modularity
5. Expect heterogeneity
6. Avoid static options and parameters
7. Look for a good design; it need not be perfect
8. Be strict when sending and tolerant when receiving
9. Think about scalability
10. Consider performance and cost

The glue that holds the whole Internet together is the network layer protocol, **IP** (**Internet Protocol**).

1. The IP Protocol





2. IP Addresses

An IPv4 address is a 32-bit address that uniquely and universally defines the connection of a device to the Internet. The IPv4 addresses are universal in the sense that the addressing system must be accepted by any host that wants to be connected to the Internet.

Address Space

An address space is the total number of addresses used by the protocol. If a protocol uses N bits to define an address, the address space is 2^N because each bit can have two different values (0 or 1) and N bits can have 2^N values. IPv4 uses 32-bit addresses, which means that the address space is 2^{32} or 4,294,967,296 (more than 4 billion).

Notations

There are two prevalent notations to show an IPv4 address: binary notation and dotted decimal notation.

Binary Notation

In binary notation, the IPv4 address is displayed as 32 bits. Each octet is often referred to as a byte. So it is common to hear an IPv4 address referred to as a 32-bit address or a 4-byte address. The following is an example of an IPv4 address in binary notation:

01110101 10010101 00011101 00000010

Dotted-Decimal Notation

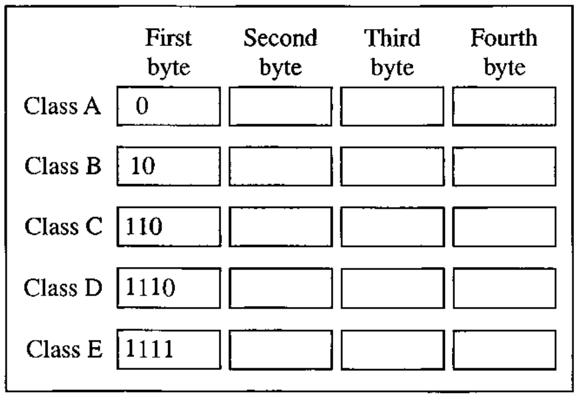
To make the IPv4 address more compact and easier to read, Internet addresses are usually written in decimal form with a decimal point (dot) separating the bytes. The following is the dotted decimal notation of the above address:

117.149.29.2

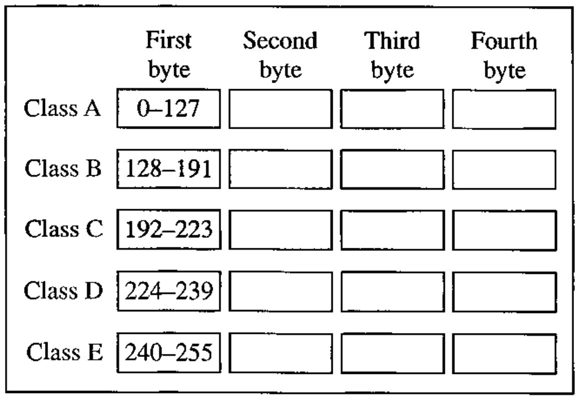
Classful Addressing

IPv4 addressing, at its inception, used the concept of classes. This architecture is called classful addressing. In classful addressing, the address space is divided into five classes: A, B, C, D, and E. Each class occupies some part of the address space.

Finding the classes in binary and dotted-decimal notation



a. Binary notation



b. Dotted-decimal notation

Classes and Blocks

One problem with classful addressing is that each class is divided into a fixed number of blocks with each block having a fixed size.

Class	*Number of Blocks*	*Block Size*	*Application*
A	128	16,777,216	Unicast
B	16,384	65,536	Unicast
C	2,097,152	256	Unicast
D	1	268,435,456	Multicast
E	1	268,435,456	Reserved

Netid and Hostid

In classful addressing, an IP address in class A, B, or C is divided into netid and hostid. These parts are of varying lengths, depending on the class of the address. In class A, one byte defines the netid and three bytes define the hostid. In class B, two bytes define the netid and two bytes define the hostid. In class C, three bytes define the netid and one byte defines the hostid.

Default masks for classful addressing

Class	*Binary*	*Dotted-Decimal*	*CIDR*
A	**11111111 00000000 00000000 00000000**	**255.0.0.0**	/8
B	**11111111 11111111 00000000 00000000**	**255.255.0.0**	/16
C	**11111111 11111111 11111111 00000000**	**255.255.255.0**	/24

The last column shows the mask in the form */n* where *n* can be 8, 16, or 24 in classful addressing. This notation is also called slash notation or Classless Interdomain Routing (CIDR) notation. The notation is used in classless addressing,

Subnetting

During the era of classful addressing, subnetting was introduced. If an organization was granted a large block in class A or B, it could divide the addresses into several contiguous groups and assign each group to smaller networks (called subnets).

Supernetting

In supernetting, an organization can combine several class C blocks to create a larger range of addresses. In other words, several networks are combined to create a supernetwork or a supemet. An organization can apply for a set of class C blocks instead of just one.

Classless Addressing

To overcome address depletion and give more organizations access to the Internet, classless addressing was designed and implemented.

Address Blocks

In classless addressing, when an entity, small or large, needs to be connected to the Internet, it is granted a block (range) of addresses. The size of the block (the number of addresses) varies based on the nature and size of the entity.

Restriction To simplify the handling of addresses, the Internet authorities impose three restrictions on classless address blocks:

1. The addresses in a block must be contiguous, one after another.

2. The number of addresses in a block must be a power of 2 (I, 2, 4, 8 ...).

3. The first address must be evenly divisible by the number of addresses.

In 1Pv4 addressing, a block of addresses can be defined as x.y.z.t/n in which x.y.z.t defines one of the addresses and the */n* defines the mask.

The first address in the block can be found by setting the rightmost 32 - *n* bits to 0s.

The last address in the block can be found by setting the rightmost 32 - *n* bits to 1s.

The number of addresses in the block can be found by using the formula 2^{32-n}

Network Addresses

The first address in a block is normally not assigned to any device; it is used as the network address that represents the organization to the rest of the world.

Each address in the block can be considered as a two-level hierarchical structure: the leftmost *n* bits (prefix) define the network; the rightmost 32 - *n* bits define the host.

Address Allocation

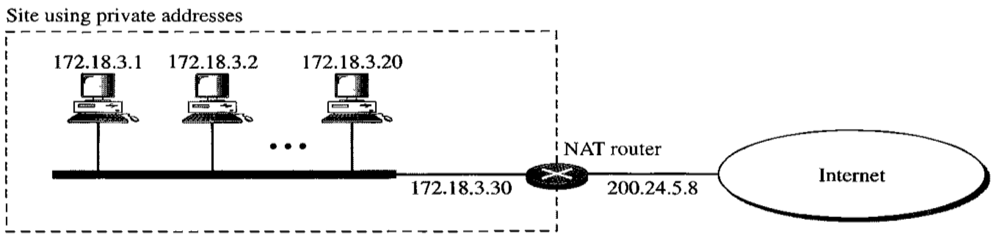
The ultimate responsibility of address allocation is given to a global authority called the ***Internet Corporation for Assigned Names and Addresses*** **(ICANN).** It assigns a large block of addresses to an ISP. Each ISP, in turn, divides its assigned block into smaller sub blocks and grants the sub blocks to its customers. This is called address aggregation: many blocks of addresses are aggregated in one block and granted to one ISP.

Network Address Translation (NAT)

NAT enables a user to have a large set of addresses internally and one address, or a small set of addresses, externally. The traffic inside can use the large set; the traffic outside, the small set. To separate the addresses used inside the home or business and the ones used for the Internet, the Internet authorities have reserved three sets of addresses as private addresses.

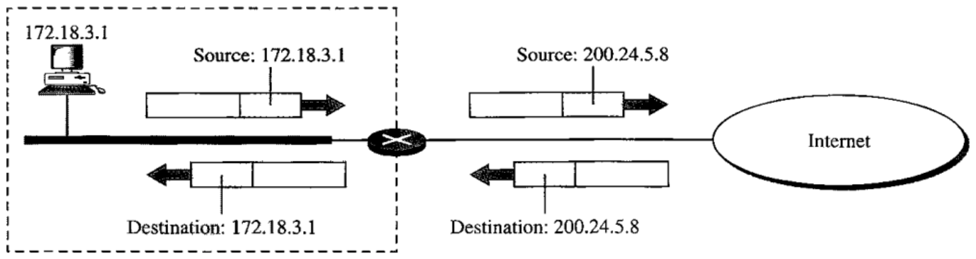
Range			*Total*
10.0.0.0	to	10.255.255.255	2^{24}
172.16.0.0	to	172.31.255.255	2^{20}
192.168.0.0	to	192.168.255.255	2^{16}

Any organization can use an address out of this set without permission from the Internet authorities. Everyone knows that these reserved addresses are for private networks. They are unique inside the organization, but they are not unique globally. The site must have only one single connection to the global Internet through a router that runs the NAT software.



Address Translation

All the outgoing packets go through the NAT router, which replaces the *source address* in the packet with the global NAT address. All incoming packets also pass through the NAT router, which replaces the *destination address* in the packet (the NAT router global address) with the appropriate private address.



Using One IP Address: In its simplest form, a translation table has only two columns: the private' address and the external address (destination address of the packet). When the router translates the source address of the outgoing packet, it also makes note of the destination address-where the packet is going. When the response comes back from the destination, the router uses the source address of the packet to find the private address of the packet.

Using a Pool of IP Addresses: Since the NAT router has only one global address, only one private network host can access the same external host. To remove this restriction, the NAT router uses a pool of global addresses. For example, instead of using only one global address (200.24.5.8), the NAT router can use four addresses (200.24.5.8, 200.24.5.9, 200.24.5.10, and 200.24.5.11). In this case, four private network hosts can communicate with the same external host at the same time because each pair of addresses defines a connection.

3. Internet Control Protocols:

ICMP

The Internet Control Message Protocol (ICMP) has been designed to compensate mechanisms for host and management queries

Types of Messages

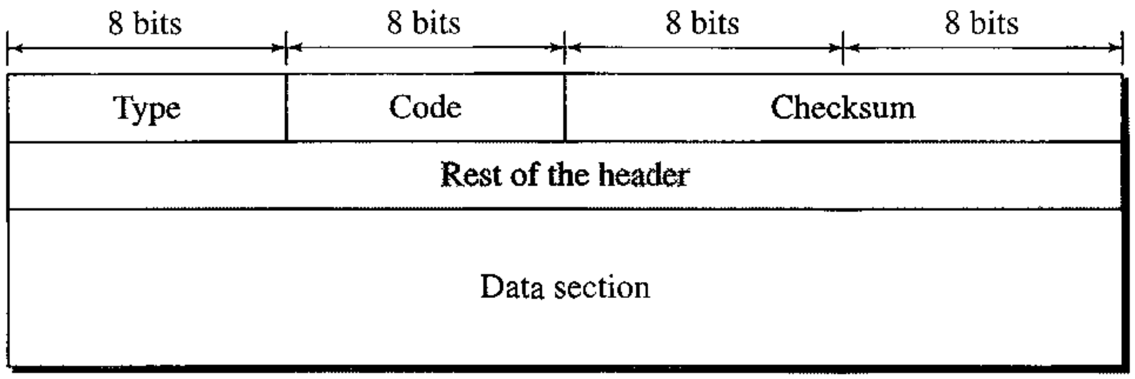
ICMP messages are divided into two broad categories: error-reporting messages and query messages.

The error-reporting messages report problems that a router or a host (destination) may encounter when it processes an IP packet.

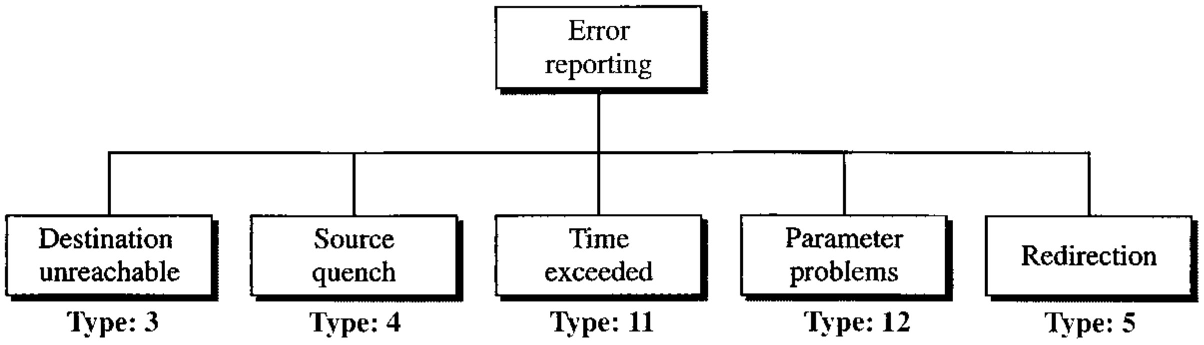
The query messages, which occur in pairs, help a host or a network manager get specific information from a router or another host. For example, nodes can discover their neighbors.

<u>Message Format</u>

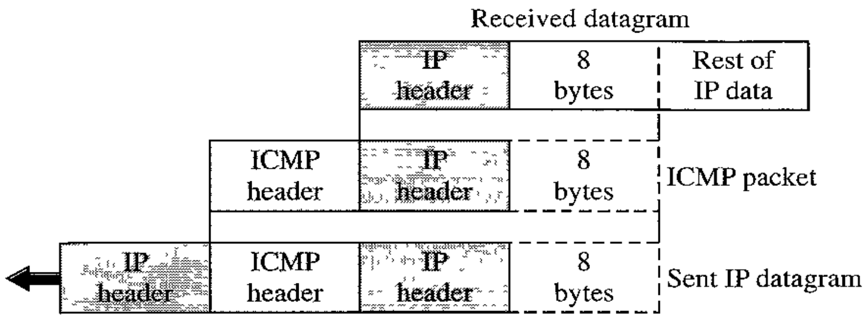
An ICMP message has an 8-byte header and a variable-size data section. Although the general format of the header is different for each message type, the first 4 bytes are common to all.



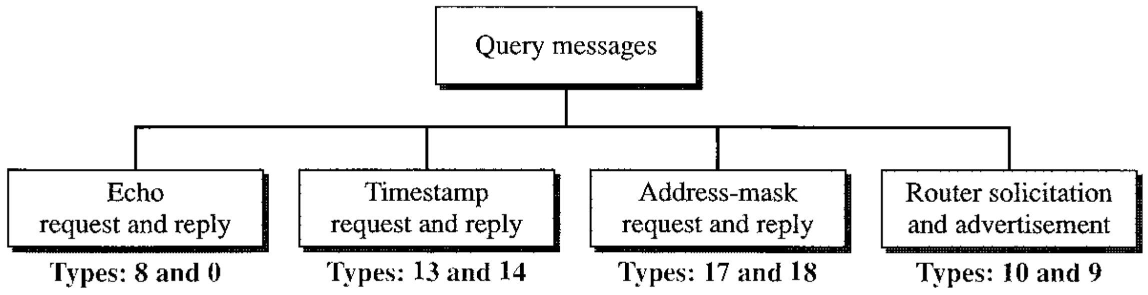
Five types of errors are handled: destination unreachable, source quench, time exceeded, parameter problems, and redirection



Contents of data field for the error messages



Query messages



Debugging Tools

There are several tools that can be used in the Internet for debugging. We can trace the route of a packet. The two tools that use ICMP for debugging: ***ping*** and ***traceroute.***

ADDRESS RESOLUTION PROTOCOL:

The host or the router sends an ARP query packet. The packet includes the physical and IP addresses of the sender and the IP address of the receiver. Because the sender does not know the physical address of the receiver, the query is broadcast over the network

Every host or router on the network receives and processes the ARP query packet, but only the intended recipient recognizes its IP address and sends back an ARP response packet. The response packet contains the recipient's IP and physical addresses. The packet is unicast directly to the inquirer by using the physical address received in the query packet.

Operation:

These are the steps involved in an ARP process:

1. The sender knows the IP address of the target.

2. IP asks ARP to create an ARP request message, filling in the sender physical address, the sender IP address, and the target IP address. The target physical address field is filled with 0s.

3. The message is passed to the data link layer where it is encapsulated in a frame by using the physical address of the sender as the source address and the physical broadcast address as the destination address.

4. Every host or router receives the frame. Because the frame contains a broadcast destination address, all stations remove the message and pass it to ARP. All machines except the one targeted drop the packet. The target machine recognizes its IP address.

5. The target machine replies with an ARP reply message that contains its physical address. The message is unicast.

6. The sender receives the reply message. It now knows the physical address of the target machine.

7. The IP datagram, which carries data for the target machine, is now encapsulated in a frame and is unicast to the destination.

<u>Mapping Physical to Logical Address: RARP, BOOTP, and DHCP</u>

There are occasions in which a host knows its physical address, but needs to know its logical address. This may happen in two cases:

1. A diskless station is just booted. The station can find its physical address by checking its interface, but it does not know its IP address.

2. An organization does not have enough IP addresses to assign to each station; it needs to assign IP addresses on demand. The station can send its physical address and ask for a short time lease.

Reverse Address Resolution Protocol (RARP) finds the logical address for a machine that knows only its physical address. A RARP request is created and broadcast on the local network. Another machine on the local network that knows all the IP addresses will respond with a RARP reply. The requesting machine must be running a RARP client program; the responding machine must be running a RARP server program.

BOOTP

The Bootstrap Protocol (BOOTP) is a client/server protocol designed to provide physical address to logical address mapping. BOOTP is an application layer protocol. The administrator may put the client and the server on the same network or on different networks. BOOTP messages are encapsulated in a UDP packet, and the UDP packet itself is encapsulated in an IP packet.

DHCP

The Dynamic Host Configuration Protocol (DHCP) has been devised to provide static and dynamic address allocation that can be manual or automatic.

Static Address Allocation In this capacity DHCP acts as BOOTP does. It is backward compatible with BOOTP, which means a host running the BOOTP client can request a static address from a DHCP server. A DHCP server has a database that statically binds physical addresses to IP addresses.

Dynamic Address Allocation DHCP has a second database with a pool of available IP addresses. This second database makes DHCP dynamic. When a DHCP client requests a temporary IP address, the DHCP server goes to the pool of available (unused) IP addresses and assigns an IP address for a negotiable period of time.

4. OSPF—the Interior Gateway Routing Protocol:

The Internet is made up of a large number of autonomous systems. Each AS is operated by a different organization and can use its own routing algorithm inside. A routing algorithm within an AS is called an **interior gateway protocol**; an algorithm for routing between AS is called an **exterior gateway protocol**.

The original Internet interior gateway protocol was a distance vector protocol (RIP) based on the Bellman-Ford algorithm inherited from the ARPANET. It worked well in small systems, but less well as AS got larger. It also suffered from the count-to-infinity problem and generally slows convergence, so it was replaced by a link state protocol. In 1988, the Internet Engineering Task Force began work on a successor. That successor, called **OSPF** (**Open Shortest Path First**), became a standard in 1990.

Given the long experience with other routing protocols, the group designing the new protocol had a long list of requirements that had to be met.

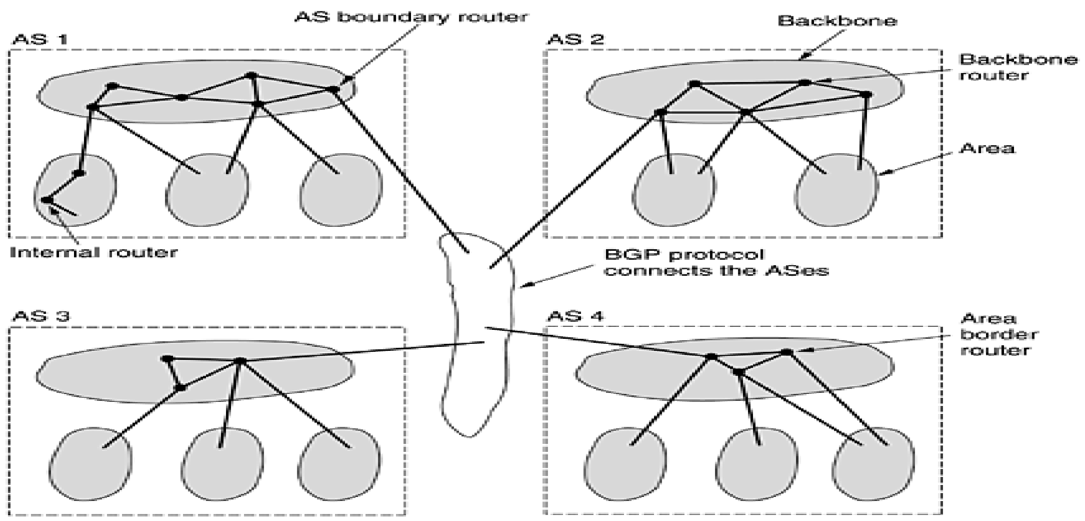
- The algorithm had to be published in the open literature
- Second, the new protocol had to support a variety of distance metrics, including physical distance, delay, and so on.
- Third, it had to be a dynamic algorithm, one that adapted to changes in the topology automatically and quickly.
- Fourth, and new for OSPF, it had to support routing based on type of service.
- Fifth, and related to the above, the new protocol had to do load balancing, splitting the load over multiple lines.
- Sixth, support for hierarchical systems was needed.
- Seventh, some modicum of security was required to prevent fun-loving students from spoofing routers by sending them false routing information.

OSPF supports three kinds of connections and networks:

1. Point-to-point lines between exactly two routers.

2. Multi-access networks with broadcasting (e.g., most LANs).

3. Multi-access networks without broadcasting (e.g., most packet-switched WANs).



The relation between AS, backbones, and areas in OSPF

OSPF distinguishes four classes of routers:

1. Internal routers are wholly within one area.

2. Area border routers connect two or more areas.

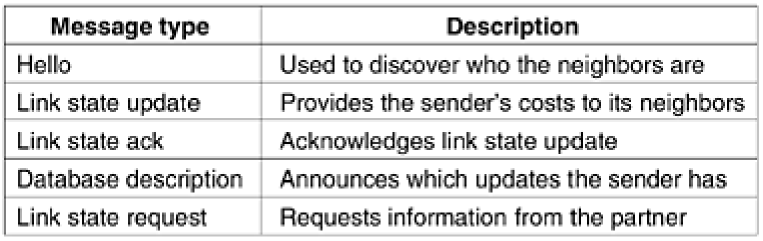
3. Backbone routers are on the backbone.

4. AS boundary routers talk to routers in other ASes.

OSPF works by exchanging information between adjacent routers, which is not the same as between neighboring routers. In particular, it is inefficient to have every router on a LAN talk to every other router on the LAN. To avoid this situation, one router is elected as the **designated router**. It is said to be **adjacent** to all the other routers on its LAN, and exchanges information with them. Neighboring routers that are not adjacent do not exchange information with each other. A backup designated router is always kept up to date to ease the transition should the primary designated router crash and need to replaced immediately.

During normal operation, each router periodically floods LINK STATE UPDATE messages to each of its adjacent routers. This message gives its state and provides the costs used in the topological database. The flooding messages are acknowledged, to make them reliable. Each message has a sequence number, so a router can see whether an incoming LINK STATE UPDATE is older or newer than what it currently has. Routers also send these messages when a line goes up or down or its cost changes.

The five kinds of messages are summarized below:



Message type	Description
Hello	Used to discover who the neighbors are
Link state update	Provides the sender's costs to its neighbors
Link state ack	Acknowledges link state update
Database description	Announces which updates the sender has
Link state request	Requests information from the partner

5. BGP—the Exterior Gateway Routing Protocol

Between AS, a different protocol, **BGP** (**Border Gateway Protocol**), is used. Exterior gateway protocols in general, and BGP in particular, have been designed to allow many kinds of routing policies to be enforced in the interAS traffic.

Typical policies involve political, security, or economic considerations. A few examples of routing constraints are:

1. No transit traffic through certain AS.

2. Never put Iraq on a route starting at the Pentagon.

3. Do not use the United States to get from British Columbia to Ontario.

4. Only transit Albania if there is no alternative to the destination.

5. Traffic starting or ending at IBM should not transit Microsoft.

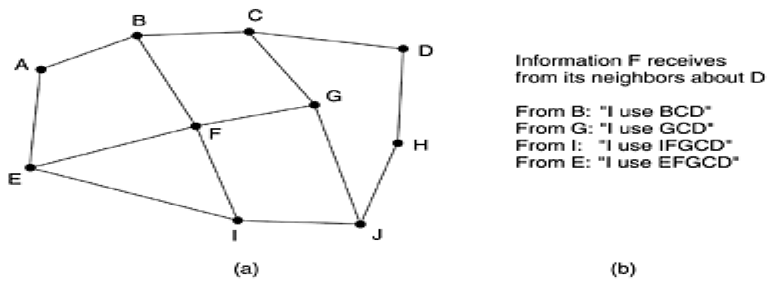
Policies are typically manually configured into each BGP router (or included using some kind of script). They are not part of the protocol itself.

From the point of view of a BGP router, the world consists of AS and the lines connecting them. Two AS are considered connected if there is a line between border routers in each one. Given BGP's special interest in transit traffic, networks are grouped into one of three categories. The first category is the **stub networks**, which have only one connection to the BGP graph. These cannot be used for transit traffic because there is no one on the other side. Then come the **multiconnected networks**. These could be used for transit traffic, except that they refuse.

Finally, there are the **transit networks**, such as backbones, which are willing to handle third-party packets, possibly with some restrictions, and usually for pay.

Instead of maintaining just the cost to each destination, each BGP router keeps track of the path used. Similarly, instead of periodically giving each neighbor its estimated cost to each possible destination, each BGP router tells its neighbors the exact path it is using.

As an example, consider the BGP routers shown in figure (a). In particular, consider *F*'s routing table. Suppose that it uses the path *FGCD* to get to *D*. When the neighbors give it routing information, they provide their complete paths, as shown in figure (b)



Information F receives
from its neighbors about D

From B: "I use BCD"
From G: "I use GCD"
From I: "I use IFGCD"
From E: "I use EFGCD"

***(a) A set of BGP routers. (b) Information sent to* F.**

After all the paths come in from the neighbors, *F* examines them to see which is the best. Any route violating a policy constraint automatically gets a score of infinity. The router then adopts the route with the shortest distance. The scoring function is not part of the BGP protocol and can be any function the system managers want.

8. IPv6

In 1990, IETF started work on a new version of IP, one which would never run out of addresses, would solve a variety of other problems, and be more flexible and efficient as well. Its major goals were:

1. Support billions of hosts, even with inefficient address space allocation.

2. Reduce the size of the routing tables.

3. Simplify the protocol, to allow routers to process packets faster.

4. Provide better security (authentication and privacy) than current IP.

5. Pay more attention to type of service, particularly for real-time data.

6. Aid multicasting by allowing scopes to be specified.

7. Make it possible for a host to roam without changing its address.

8. Allow the protocol to evolve in the future.

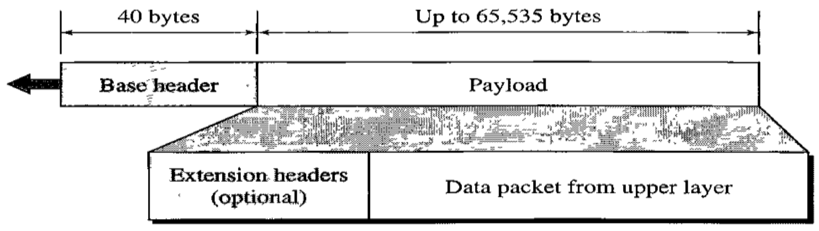
9. Permit the old and new protocols to coexist for years.

Advantages

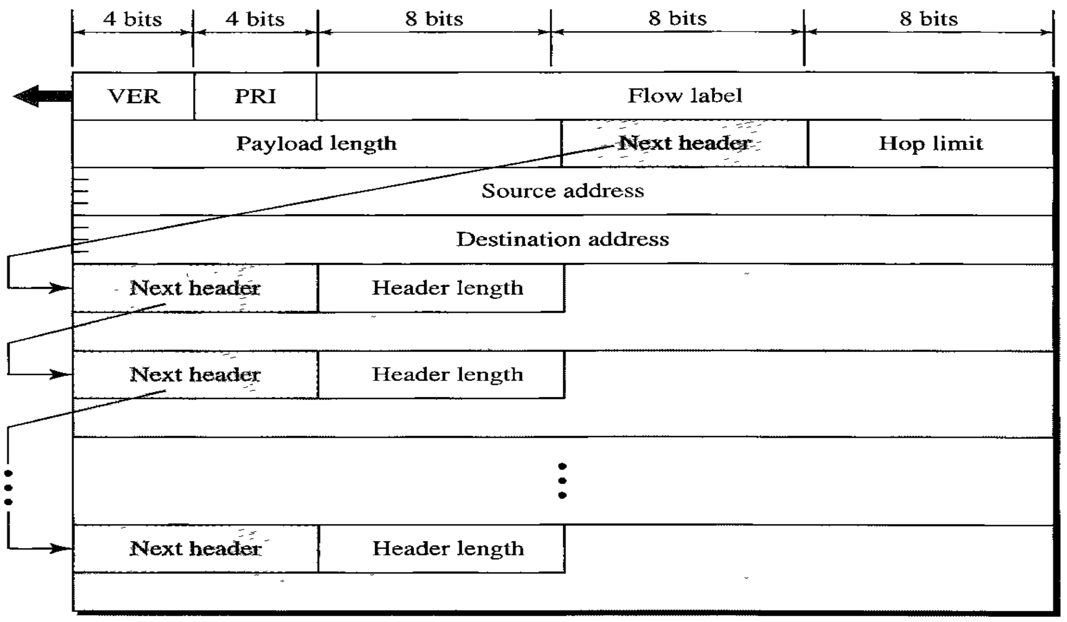
The next-generation IP, or IPv6, has some advantages over IPv4 that can be summarized as follows:

- Larger address space
- Better header format
- Allowance for extension
- Support for resource allocation
- Support for more security

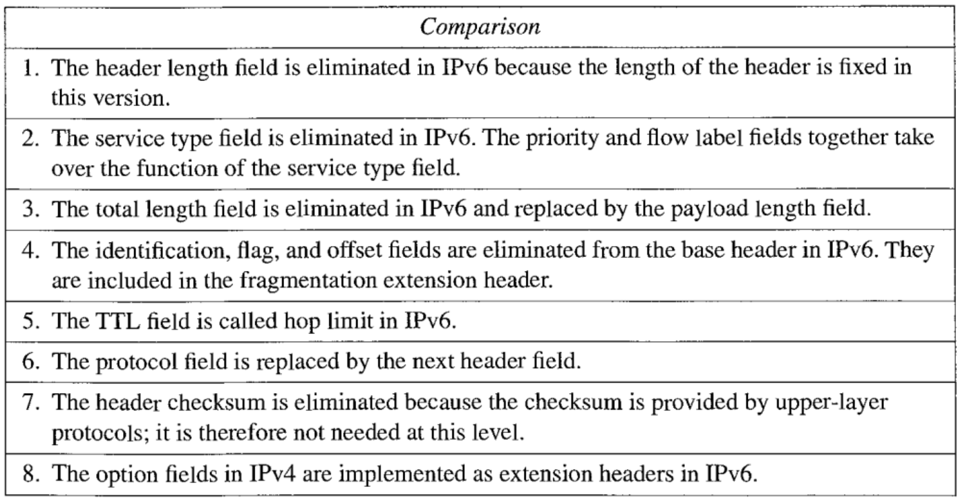
<u>Packet Format</u>



IPv6 datagram header and payload

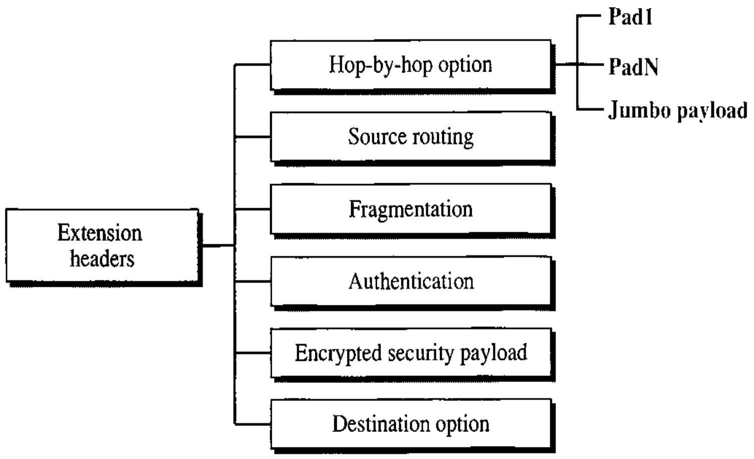


Comparison between IPv4 and IPv6 Headers



Comparison
1. The header length field is eliminated in IPv6 because the length of the header is fixed in this version.
2. The service type field is eliminated in IPv6. The priority and flow label fields together take over the function of the service type field.
3. The total length field is eliminated in IPv6 and replaced by the payload length field.
4. The identification, flag, and offset fields are eliminated from the base header in IPv6. They are included in the fragmentation extension header.
5. The TTL field is called hop limit in IPv6.
6. The protocol field is replaced by the next header field.
7. The header checksum is eliminated because the checksum is provided by upper-layer protocols; it is therefore not needed at this level.
8. The option fields in IPv4 are implemented as extension headers in IPv6.

Extension Headers



Comparison between IPv4 options and IPv6 extension headers

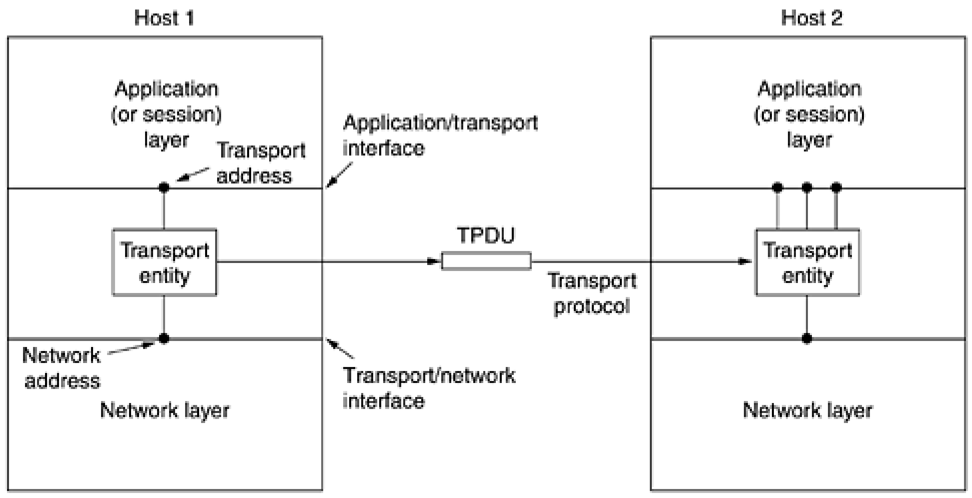
Comparison
1. The no-operation and end-of-option options in IPv4 are replaced by Pad1 and PadN options in IPv6.
2. The record route option is not implemented in IPv6 because it was not used.
3. The timestamp option is not implemented because it was not used.
4. The source route option is called the source route extension header in IPv6.
5. The fragmentation fields in the base header section of IPv4 have moved to the fragmentation extension header in IPv6.
6. The authentication extension header is new in IPv6.
7. The encrypted security payload extension header is new in IPv6.

THE TRANSPORT LAYER

THE TRANSPORT SERVICE

1. Services Provided to the Upper Layers

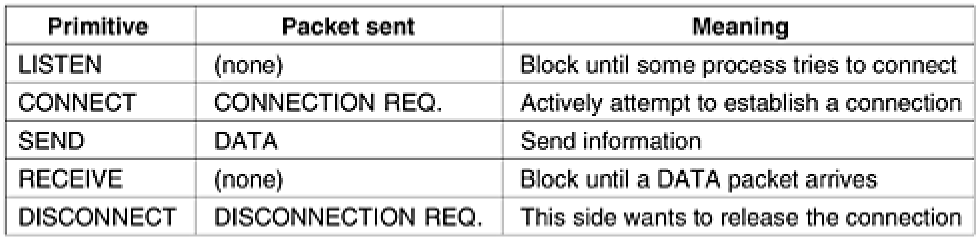
The ultimate goal of the transport layer is to provide efficient, reliable, and cost-effective service to its users, normally processes in the application layer. To achieve this goal, the transport layer makes use of the services provided by the network layer. The hardware and/or software within the transport layer that does the work is called the **transport entity**. The transport entity can be located in the operating system kernel, in a separate user process, in a library package bound into network applications, or conceivably on the network interface card.



The network, transport, and application layers

The bottom four layers can be seen as the **transport service provider**, whereas the upper layer(s) are the **transport service user**. This distinction of provider versus user has a considerable impact on the design of the layers and puts the transport layer in a key position.

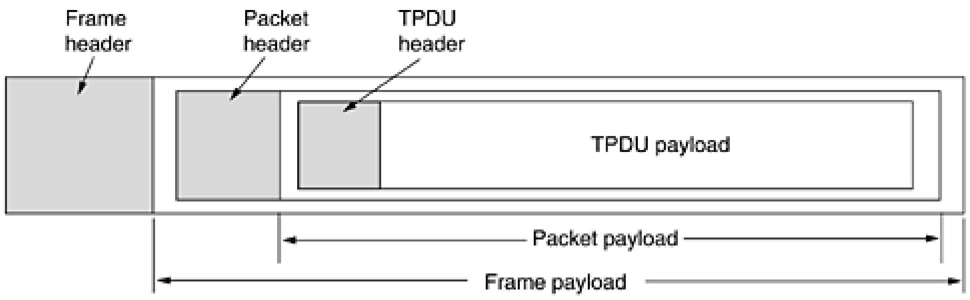
2. Transport Service Primitives



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

The primitives for a simple transport service

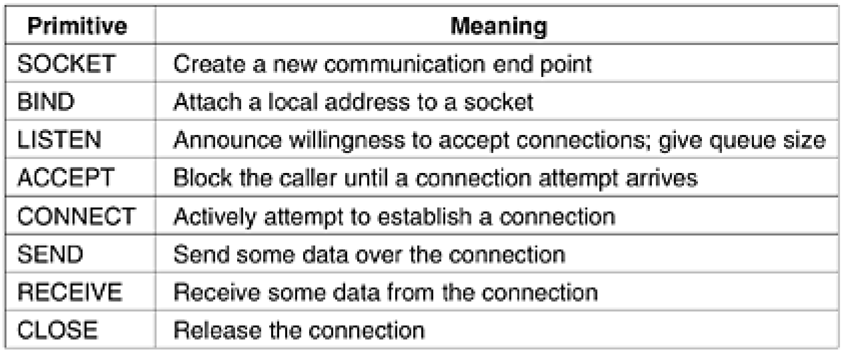
We reluctantly use the acronym **TPDU** (**Transport Protocol Data Unit**) for messages sent from transport entity to transport entity. Thus, TPDUs are contained in packets. In turn, packets are contained in frames. When a frame arrives, the data link layer processes the frame header and passes the contents of the frame payload field up to the network entity. The network entity processes the packet header and passes the contents of the packet payload up to the transport entity. This nesting is illustrated in figure below



Nesting of TPDUs, packets, and frames

3. Berkeley Sockets

Another set of transport primitives used are the socket primitives used in Berkeley UNIX for TCP. These primitives are widely used for Internet programming.



Primitive	Meaning
SOCKET	Create a new communication end point
BIND	Attach a local address to a socket
LISTEN	Announce willingness to accept connections; give queue size
ACCEPT	Block the caller until a connection attempt arrives
CONNECT	Actively attempt to establish a connection
SEND	Send some data over the connection
RECEIVE	Receive some data from the connection
CLOSE	Release the connection

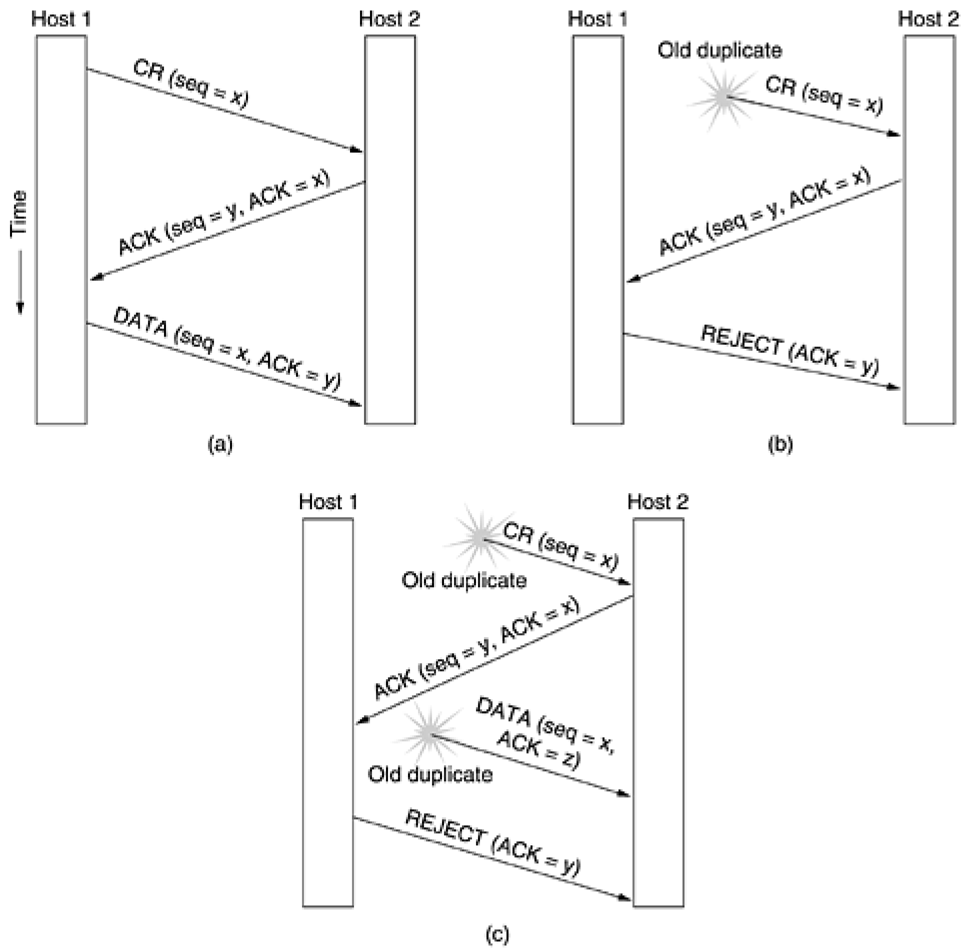
ELEMENTS OF TRANSPORT PROTOCOLS:

The transport service is implemented by a **transport protocol** used between the two transport entities. However, significant differences between the two also exist. At the data link layer, two routers communicate directly via a physical channel, whereas at the transport layer, this physical channel is replaced by the entire subnet.

1. Addressing

2. Connection Establishment

To solve the problems in transport layer we introduced the **three-way handshake**. This establishment protocol does not require both sides to begin sending with the same sequence number, so it can be used with synchronization methods other than the global clock method. The normal setup procedure when host 1 initiates is shown in figure (a). Host 1 chooses a sequence number, *x*, and sends a CONNECTION REQUEST TPDU containing it to host 2. Host 2 replies with an ACK TPDU acknowledging *x* and announcing its own initial sequence number, *y*. Finally, host 1 acknowledges host 2's choice of an initial sequence number in the first data TPDU that it sends.

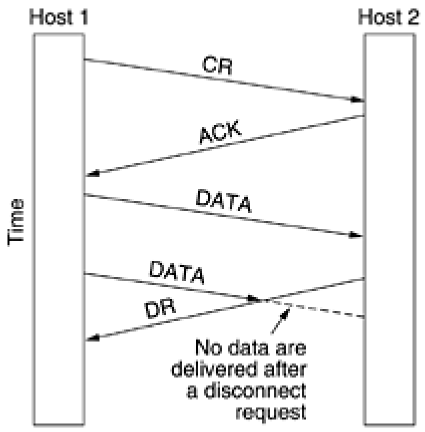


(a) Normal operation. (b) Old duplicate CONNECTION REQUEST appearing out of nowhere. (c) Duplicate CONNECTION REQUEST and duplicate ACK.

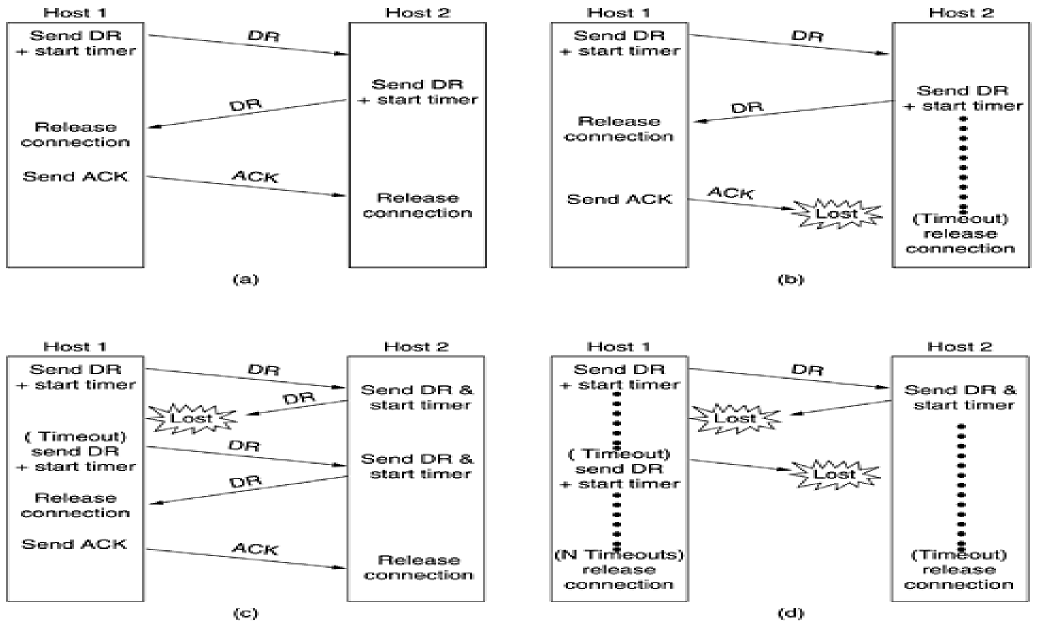
3. Connection Release

There are two styles of terminating a connection: asymmetric release and symmetric release. Asymmetric release is the way the telephone system works: when one party hangs up, the connection is broken. Symmetric release treats the connection as two separate unidirectional connections and requires each one to be released separately.

Asymmetric release is abrupt and may result in data loss. Consider the scenario of below figure. After the connection is established, host 1 sends a TPDU that arrives properly at host 2. Then host 1 sends another TPDU. Unfortunately, host 2 issues a DISCONNECT before the second TPDU arrives. The result is that the connection is released and data are lost.



Abrupt disconnection with loss of data

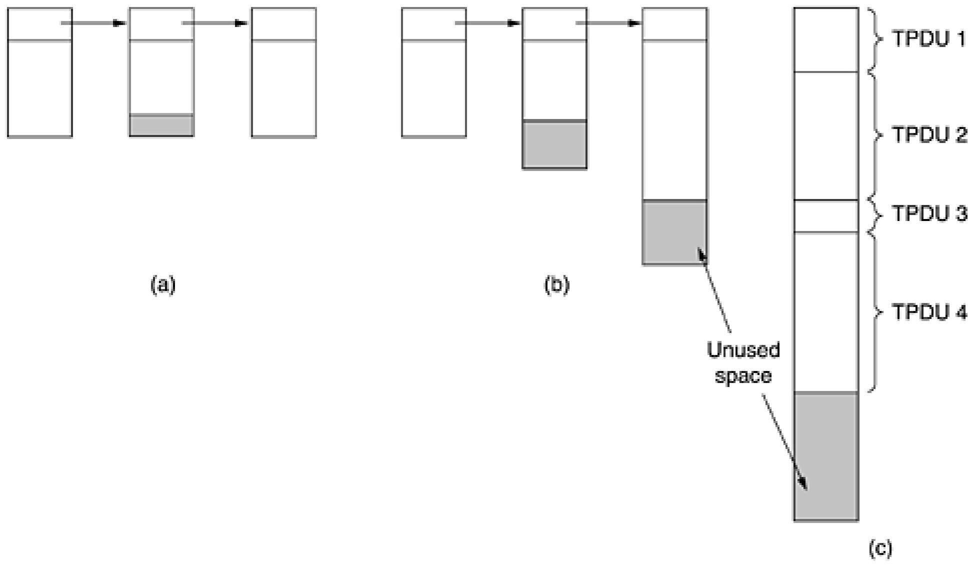


(a) Normal case of three-way handshake. (b) Final ACK lost. (c) Response lost. (d) Response lost and subsequent DRs lost

4. Flow Control and Buffering

In some ways the flow control problem in the transport layer is the same as in the data link layer, but in other ways it is different. The basic similarity is that in both layers a sliding window or other scheme is needed on each connection to keep a fast transmitter from overrunning a slow receiver. The main difference is that a router usually has relatively few lines, whereas a host may have numerous connections. This difference makes it impractical to implement the data link buffering strategy in the transport layer.

Even if the receiver has agreed to do the buffering, there still remains the question of the buffer size. If most TPDUs are nearly the same size, it is natural to organize the buffers as a pool of identically-sized buffers, with one TPDU per buffer, as in figure (a).



(a) Chained fixed-size buffers. (b) Chained variable-sized buffers. (c) One large circular buffer per connection

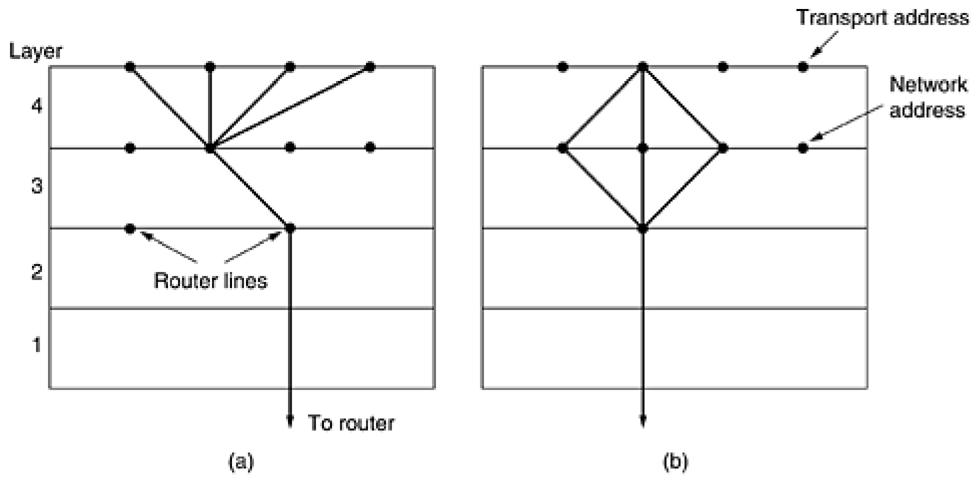
Another approach to the buffer size problem is to use variable-sized buffers, as in figure (b). The advantage here is better memory utilization, at the price of more complicated buffer management. A third possibility is to dedicate a single large circular buffer per connection, as in figure (c). This system also makes good use of memory, provided that all connections are heavily loaded, but is poor if some connections are lightly loaded.

The flow control mechanism must be applied at the sender to prevent it from having too many unacknowledged TPDUs outstanding at once. If the network can handle *c* TPDUs/sec and the cycle time is *r*, then the sender's window should be *cr*. With a window of this size the sender normally operates with the pipeline full. Any small decrease in network performance will cause it to block.

5. Multiplexing

In the transport layer the need for multiplexing can arise in a number of ways. When a TPDU comes in, some way is needed to tell which process to give it to. This situation, called

upward multiplexing. In this figure, four distinct transport connections all use the same network connection to the remote host.



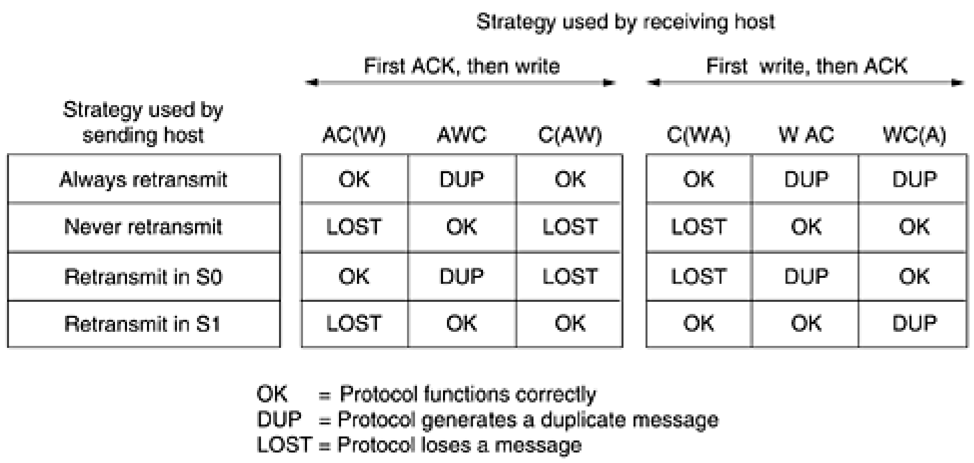
(a) Upward multiplexing. (b) Downward multiplexing

6. Crash Recovery

If hosts and routers are subject to crashes, recovery from these crashes becomes an issue. If the transport entity is entirely within the hosts, recovery from network and router crashes is straightforward. A more troublesome problem is how to recover from host crashes. In particular, it may be desirable for clients to be able to continue working when servers crash and then quickly reboot. In an attempt to recover its previous status, the server might send a broadcast TPDU to all other hosts, announcing that it had just crashed and requesting that its clients inform it of the status of all open connections. Each client can be in one of two states: one TPDU outstanding, *S1*, or no TPDUs outstanding, *S0*. Based on only this state information, the client must decide whether to retransmit the most recent TPDU.

At first glance it would seem obvious: the client should retransmit only if and only if it has an unacknowledged TPDU outstanding when it learns of the crash.

Three events are possible at the server: sending an acknowledgement writing to the output process (*W*), and crashing (*C*). The three events can occur in six different orderings: *AC*(*W*), *AWC*, *C*(*AW*), *C*(*WA*), *WAC*, and *WC*(*A*), where the parentheses are used to indicate that neither *A* nor *W* can follow *C*. Below table shows all eight combinations of client and server strategy and the valid event sequences for each one.



	Strategy used by receiving host					
	First ACK, then write			First write, then ACK		
Strategy used by sending host	AC(W)	AWC	C(AW)	C(WA)	W AC	WC(A)
Always retransmit	OK	DUP	OK	OK	DUP	DUP
Never retransmit	LOST	OK	LOST	LOST	OK	OK
Retransmit in S0	OK	DUP	LOST	LOST	DUP	OK
Retransmit in S1	LOST	OK	OK	OK	OK	DUP

OK = Protocol functions correctly
DUP = Protocol generates a duplicate message
LOST = Protocol loses a message

Different combinations of client and server strategy

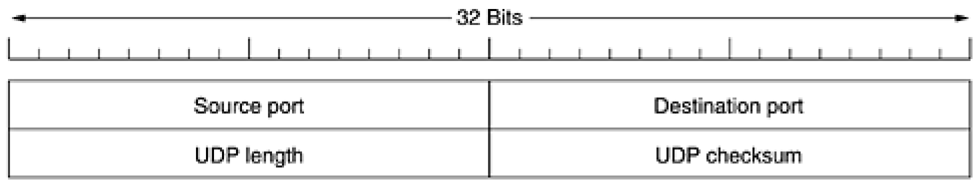
THE INTERNET TRANSPORT PROTOCOLS: UDP

The Internet has two main protocols in the transport layer, a connectionless protocol and a connection-oriented one. The connectionless protocol is UDP. The connection-oriented protocol is TCP. Because UDP is basically just IP with a short header added.

1. 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 payload. 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. This attachment occurs when BIND primitive or something similar is used.



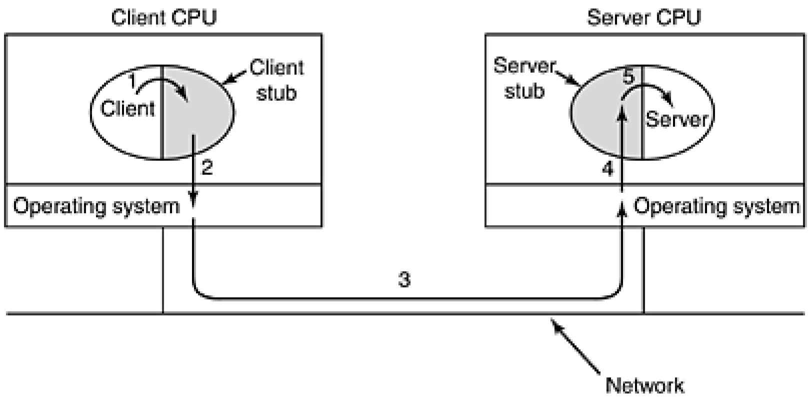
The UDP header

2. Remote Procedure Call

In a certain sense, sending a message to a remote host and getting a reply back is a lot like making a function call in a programming language. In both cases you start with one or more parameters and you get back a result. This observation has led people to try to arrange request-reply interactions on networks to be cast in the form of procedure calls. Such an arrangement makes network applications much easier to program and more familiar to deal with.

When a process on machine 1 calls a procedure on machine 2, the calling process on 1 is suspended and execution of the called procedure takes place on 2. Information can be transported from the caller to the callee in the parameters and can come back in the procedure result. No message passing is visible to the programmer. This technique is known as **RPC** (**Remote Procedure Call**) and has become the basis for many networking applications. Traditionally, the calling procedure is known as the client and the called procedure is known as the server.

The idea behind RPC is to make a remote procedure call look as much as possible like a local one. In the simplest form, to call a remote procedure, the client program must be bound with a small library procedure, called the **client stub** that represents the server procedure in the client's address space. Similarly, the server is bound with a procedure called the **server stub**. These procedures hide the fact that the procedure call from the client to the server is not local.

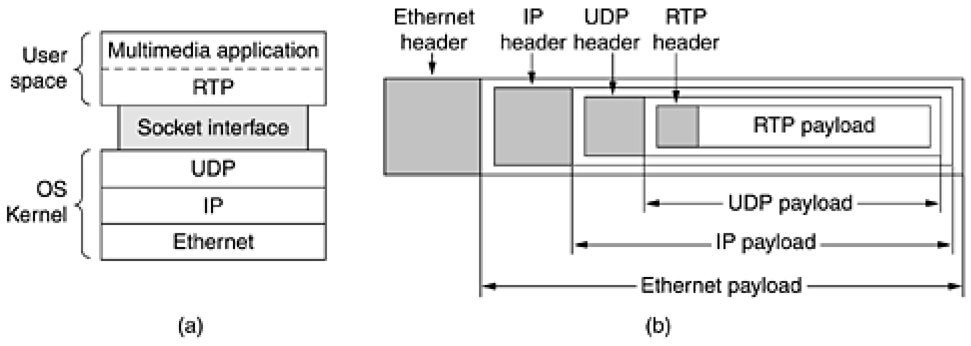


Steps in making a remote procedure call. The stubs are shaded

3. The Real-Time Transport Protocol

Client-server RPC is one area in which UDP is widely used. Another one is real-time multimedia applications. It gradually became clear that having a generic real-time transport protocol for multiple applications would be a good idea. Thus was **RTP** (**Real-time Transport Protocol**) born.

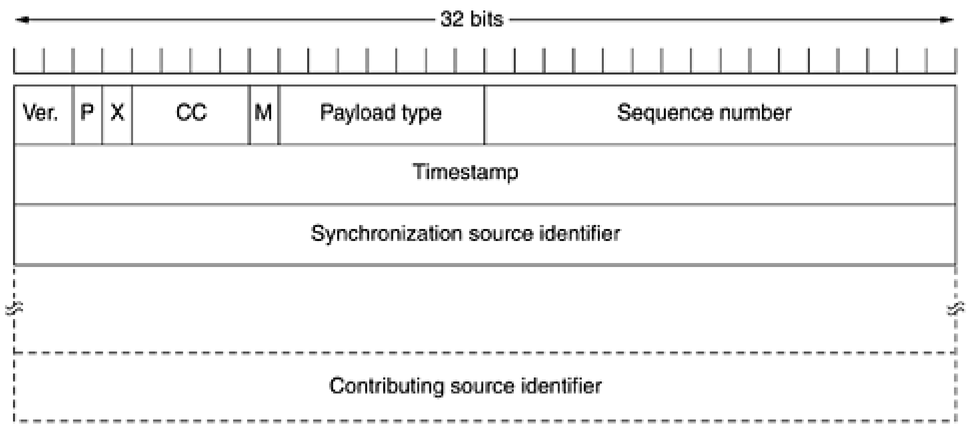
The position of RTP in the protocol stack is somewhat strange. It was decided to put RTP in user space and have it run over UDP. It operates as follows. The multimedia application consists of multiple audio, video, text, and possibly other streams. These are fed into the RTP library, which is in user space along with the application. This library then multiplexes the streams and encodes them in RTP packets, which it then stuffs into a socket. At the other end of the socket, UDP packets are generated and embedded in IP packets. If the computer is on an Ethernet, the IP packets are then put in Ethernet frames for transmission.



(a) The position of RTP in the protocol stack. (b) Packet nesting

The basic function of RTP is to multiplex several real-time data streams onto a single stream of UDP packets. The UDP stream can be sent to a single destination or to multiple destinations. Because RTP just uses normal UDP, its packets are not treated specially by the routers unless some normal IP quality-of-service features are enabled. RTP has no flow control, no error control, no acknowledgements, and no mechanism to request retransmissions.

The RTP header is illustrated in figure below. It consists of three 32-bit words and potentially some extensions. The first word contains the *Version* field, which is already at 2.


32 bits
Ver.
P
X
CC
M
Payload type
Sequence number
Timestamp
Synchronization source identifier
Contributing source identifier

RTP has a little sister protocol called **RTCP** (**Real-time Transport Control Protocol**). It handles feedback, synchronization, and the user interface but does not transport any data. The first function can be used to provide feedback on delay, jitter, bandwidth, congestion, and other network properties to the sources. This information can be used by the encoding process to increase the data rate when the network is functioning well and to cut back the data rate when there is trouble in the network. By providing continuous feedback, the encoding algorithms can be continuously adapted to provide the best quality possible under the current circumstances. RTCP also handles interstream synchronization. The problem is that different streams may use different clocks, with different granularities and different drift rates. RTCP can be used to keep them in sync.

Finally, RTCP provides a way for naming the various sources. This information can be displayed on the receiver's screen to indicate who is talking at the moment.

THE INTERNET TRANSPORT PROTOCOLS: TCP

1. Introduction to TCP

TCP (**Transmission Control Protocol**) was specifically designed to provide a reliable end-to-end byte stream over an unreliable internetwork. TCP was designed to dynamically adapt to properties of the internetwork and to be robust in the face of many kinds of failures.

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 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 reconstructs the original byte streams.

2. The TCP Service Model

TCP service is obtained by both the sender and receiver creating end points, called sockets. Each socket has a socket number (address) consisting of the IP address of the host and a 16-bit number local to that host, called a **port**. A port is the TCP name for a TSAP. Connections are identified by the socket identifiers at both ends that is, (*socket1*, *socket2*). No virtual circuit numbers or other identifiers are used. A TCP connection is a byte stream, not a message stream. Message boundaries are not preserved end to end.

Some early applications used the PUSH flag as a kind of marker to delineate messages boundaries. While this trick sometimes works, it sometimes fails since not all implementations of

TCP pass the PUSH flag to the application on the receiving side. Furthermore, if additional PUSHes come in before the first one has been transmitted (e.g., because the output line is busy), TCP is free to collect all the PUSHed data into a single IP datagram, with no separation between the various pieces.

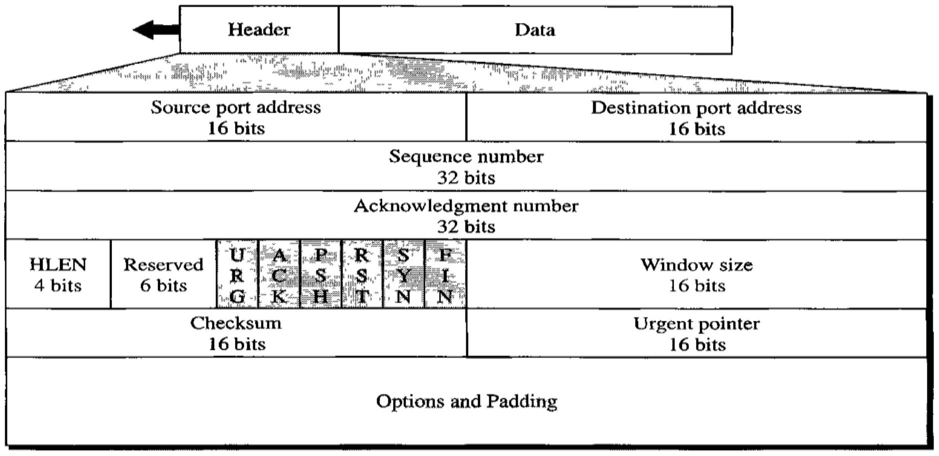
One last feature of the TCP service that is worth mentioning here is **urgent data**. When an interactive user hits the DEL or CTRL-C key to break off a remote computation that has already begun, the sending application puts some control information in the data stream and gives it to TCP along with the URGENT flag. This event causes TCP to stop accumulating data and transmit everything it has for that connection immediately.

3. The TCP Protocol

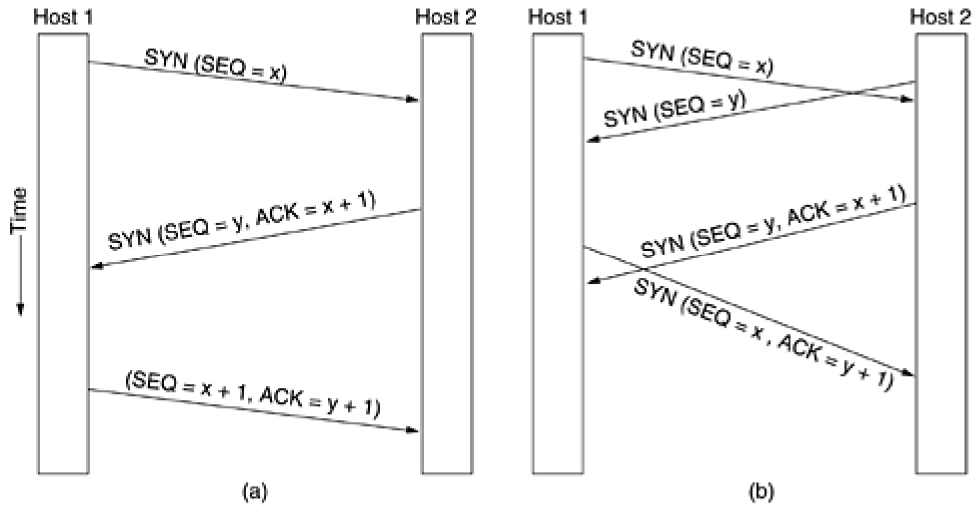
A key feature of TCP, and one which dominates the protocol design, is that every byte on a TCP connection has its own 32-bit sequence number. The sending and receiving TCP entities exchange data in the form of segments. A **TCP segment** consists of a fixed 20-byte header (plus an optional part) followed by zero or more data bytes. 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.

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 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.

4. The TCP Segment Header



5. TCP Connection Establishment


Host 1
Host 2
SYN (SEQ = x)
Time
SYN (SEQ = y, ACK = x + 1)
(SEQ = x + 1, ACK = y + 1)
(a)
Host 1
Host 2
SYN (SEQ = x)
SYN (SEQ = y)
SYN (SEQ = y, ACK = x + 1)
SYN (SEQ = x , ACK = y + 1)
(b)

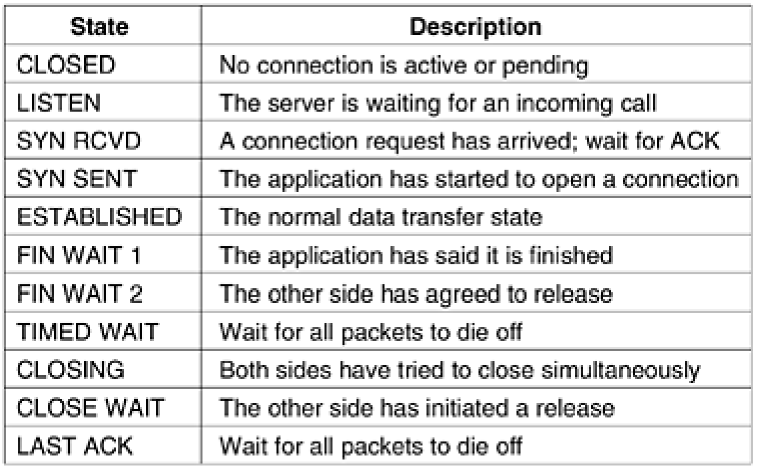
(a)TCP connection establishment in the normal case (b) Call collision

6. TCP Connection Release

To release a connection, either party can send a TCP segment with the *FIN* bit set, which means that it has no more data to transmit. When the *FIN* is acknowledged, that direction is shut down for new data. Data may continue to flow indefinitely in the other direction. When both directions have been shut down, the connection is released. Normally, four TCP segments are needed to release a connection, one *FIN* and one *ACK* for each direction.

7. TCP Connection Management Modeling

The steps required to establish and release connections can be represented in a finite state machine with the 11 states listed. In each state, certain events are legal. When a legal event happens, some action may be taken. If some other event happens, an error is reported.

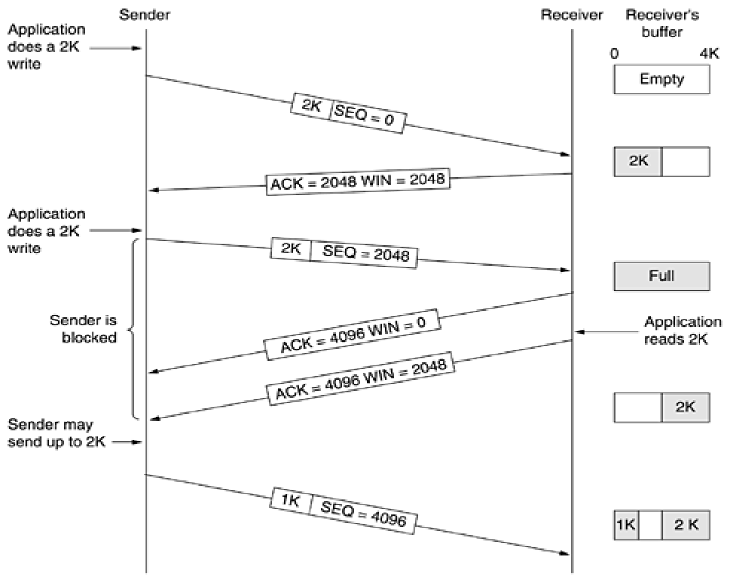


State	Description
CLOSED	No connection is active or pending
LISTEN	The server is waiting for an incoming call
SYN RCVD	A connection request has arrived; wait for ACK
SYN SENT	The application has started to open a connection
ESTABLISHED	The normal data transfer state
FIN WAIT 1	The application has said it is finished
FIN WAIT 2	The other side has agreed to release
TIMED WAIT	Wait for all packets to die off
CLOSING	Both sides have tried to close simultaneously
CLOSE WAIT	The other side has initiated a release
LAST ACK	Wait for all packets to die off

The states used in the TCP connection management finite state machine

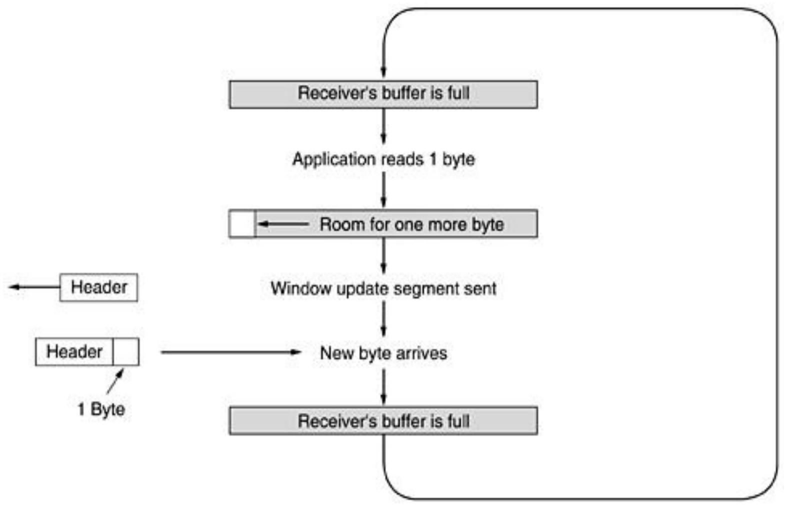
8. TCP Transmission Policy

Window management in TCP is not directly tied to acknowledgements as it is in most data link protocols.



Window management in TCP

Another problem that can degrade TCP performance is the **silly window syndrome**. This problem occurs when data are passed to the sending TCP entity in large blocks, but an interactive application on the receiving side reads data 1 byte at a time.



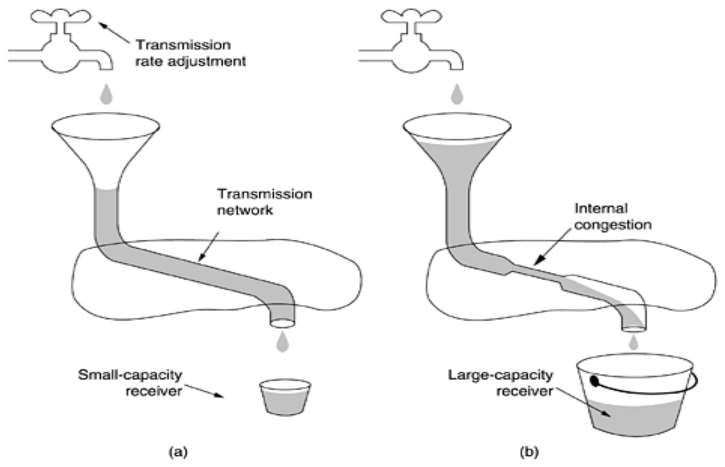
Silly window syndrome

9. TCP Congestion Control

When the load offered to any network is more than it can handle, congestion builds up. The Internet is no exception. In theory, congestion can be dealt with by employing a principle borrowed from physics: the law of conservation of packets. The idea is to refrain from injecting a new packet into the network until an old one leaves. TCP attempts to achieve this goal by dynamically manipulating the window size.

The first step in managing congestion is detecting it. A timeout caused by a lost packet could have been caused by either (1) noise on a transmission line or (2) packet discard at a congested router. Nowadays, packet loss due to transmission errors is relatively rare because most long-haul trunks are fiber. Consequently, most transmission timeouts on the Internet are due to congestion. All the Internet TCP algorithms assume that timeouts are caused by congestion and monitor timeouts for signs of trouble the way miners watch their canaries.

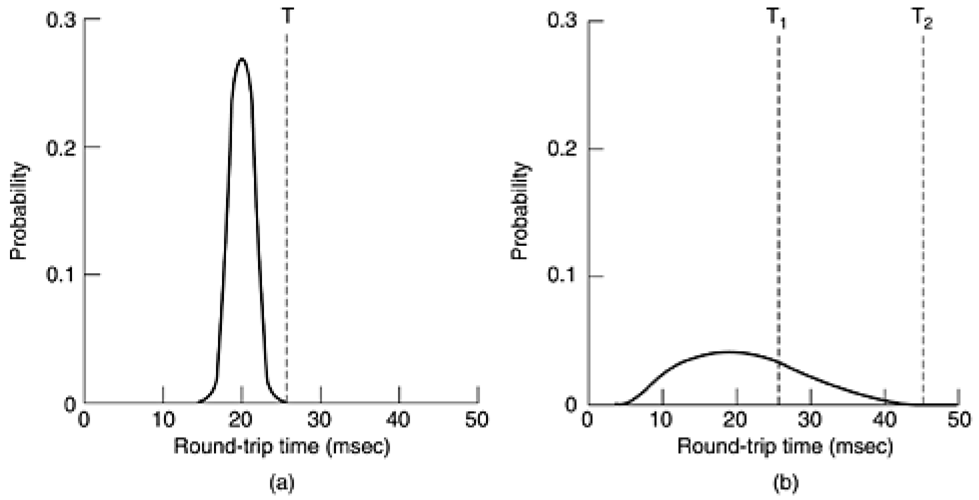
In figure (a), we see a thick pipe leading to a small-capacity receiver. As long as the sender does not send more water than the bucket can contain, no water will be lost. In figure (b), the limiting factor is not the bucket capacity, but the internal carrying capacity of the network. If too much water comes in too fast, it will back up and some will be lost.



(a) A fast network feeding a low-capacity receiver. (b) A slow network feeding a high-capacity receiver

10. TCP Timer Management

TCP uses multiple timers to do its work. The most important of these is the **retransmission timer**. When a segment is sent, a retransmission timer is started. If the segment is acknowledged before the timer expires, the timer is stopped. If, on the other hand, the timer goes off before the acknowledgement comes in, the segment is retransmitted.



(a) Probability density of acknowledgement arrival times in the data link layer (b) Probability density of acknowledgement arrival times for TCP

TCP is faced with a radically different environment. The probability density function for the time it takes for a TCP acknowledgement to come back looks more like figure (b) than figure (a). Determining the round-trip time to the destination is tricky. Even when it is known, deciding on the timeout interval is also difficult. If the timeout is set too short, say, T_1, unnecessary retransmissions will occur, clogging the Internet with useless packets. If it is set too long, performance will suffer due to the long retransmission delay whenever a packet is lost.

The solution is to use a highly dynamic algorithm that constantly adjusts the timeout interval, based on continuous measurements of network performance. For each connection, TCP maintains a variable, *RTT*, that is the best current estimate of the round-trip time to the destination in question. When a segment is sent, a timer is started, both to see how long the acknowledgement takes and to trigger a retransmission if it takes too long. If the acknowledgement gets back before the timer expires, TCP measures how long the acknowledgement took, say, *M*. It then updates *RTT* according to the formula

$$RTT = \alpha RTT + (1 - \alpha)M$$

where α is a smoothing factor that determines how much weight is given to the old value. Typically α = 7/8.

Even given a good value of *RTT*, choosing a suitable retransmission timeout is a nontrivial matter. Normally, TCP uses β*RTT*, but the trick is choosing β. In the initial implementations, β was always 2, but experience showed that a constant value was inflexible because it failed to respond when the variance went up.

A algorithm requires keeping track of another smoothed variable, *D*, the deviation. Whenever an acknowledgement comes in, the difference between the expected and observed values, | *RTT* - *M* |, is computed. A smoothed value of this is maintained in *D* by the formula

$$D = \alpha D + (1 - \alpha)\ |RTT - M|$$

where α may or may not be the same value used to smooth *RTT*. While *D* is not exactly the same as the standard deviation. Most TCP implementations now use this algorithm and set the timeout interval to

$$\text{Timeout} = RTT + 4 \times D$$

The choice of the factor 4 is somewhat arbitrary, but it has two advantages. First, multiplication by 4 can be done with a single shift. Second, it minimizes unnecessary timeouts and retransmissions because less than 1 percent of all packets come in more than four standard deviations late.

One problem that occurs with the dynamic estimation of *RTT* is what to do when a segment times out and is sent again. When the acknowledgement comes in, it is unclear whether the acknowledgement refers to the first transmission or a later one. Guessing wrong can seriously contaminate the estimate of *RTT*. One scientist made a simple proposal: do not update *RTT* on any segments that have been retransmitted. Instead, the timeout is doubled on each failure until the segments get through the first time. This fix is called **Karn's algorithm**.

The retransmission timer is not the only timer TCP uses. A second timer is the **persistence timer**. It is designed to prevent the following deadlock. The receiver sends an acknowledgement with a window size of 0, telling the sender to wait. Later, the receiver updates the window, but the packet with the update is lost. Now both the sender and the receiver are waiting for each other to do something. When the persistence timer goes off, the sender transmits a probe to the receiver. The response to the probe gives the window size. If it is still zero, the persistence timer is set again and the cycle repeats. If it is nonzero, data can now be sent.

A third timer that some implementations use is the **keep-alive timer**. When a connection has been idle for a long time, the keep-alive timer may go off to cause one side to check whether the other side is still there. If it fails to respond, the connection is terminated. This feature is controversial because it adds overhead and may terminate an otherwise healthy connection due to a transient network partition.

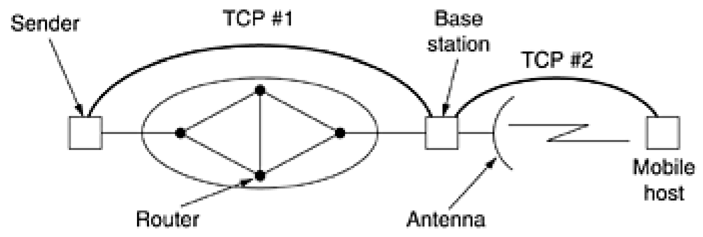
The last timer used on each TCP connection is the one used in the *TIMED WAIT* state while closing. It runs for twice the maximum packet lifetime to make sure that when a connection is closed; all packets created by it have died off.

11. Wireless TCP and UDP

Ignoring the properties of wireless transmission can lead to a TCP implementation that is logically correct but has horrendous performance.

The principal problem is the congestion control algorithm. Nearly all TCP implementations nowadays assume that timeouts are caused by congestion, not by lost packets. Unfortunately, wireless transmission links are highly unreliable. They lose packets all the time. The proper approach to dealing with lost packets is to send them again, and as quickly as possible. Slowing down just makes matters worse. In effect, when a packet is lost on a wired network, the sender should slow down. When one is lost on a wireless network, the sender should try harder. When the sender does not know what the network is, it is difficult to make the correct decision.

Frequently, the path from sender to receiver is heterogeneous. The first 1000 km might be over a wired network, but the last 1 km might be wireless. Now making the correct decision on a timeout is even harder, since it matters where the problem occurred. A solution is **indirect TCP**, is to split the TCP connection into two separate connections. The first connection goes from the sender to the base station. The second one goes from the base station to the receiver. The base station simply copies packets between the connections in both directions.

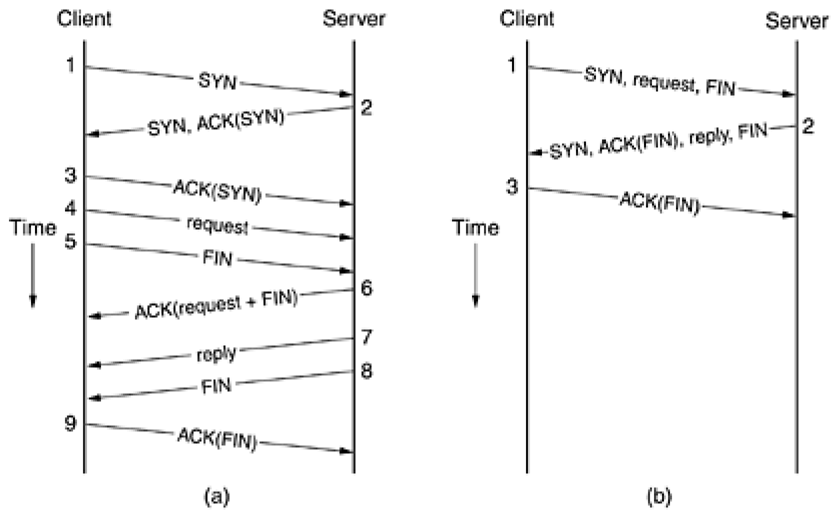


Splitting a TCP connection into two connections

12. Transactional TCP

If both the request and reply are small enough to fit into single packets and the operation is idempotent, UDP can simply be used, if these conditions are not met, using UDP is less attractive.

Clearly, that is unattractive, but using TCP itself is also unattractive. The problem is the efficiency. The normal sequence of packets for doing an RPC over TCP is shown in below figure. Nine packets are required in the best case.



(a) RPC using normal TCP. (b) RPC using T/TCP

The nine packets are as follows:

- 1. The client sends a *SYN* packet to establish a connection.
- 2. The server sends an *ACK* packet to acknowledge the SYN packet.
- 3. The client completes the three-way handshake.
- 4. The client sends the actual request.
- 5. The client sends a *FIN* packet to indicate that it is done sending.
- 6. The server acknowledges the request and the *FIN*.
- 7. The server sends the reply back to the client.
- 8. The server sends a FIN packet to indicate that it is also done.
- 9. The client acknowledges the server's *FIN*.

The question quickly arises of whether there is some way to combine the efficiency of RPC using UDP with the reliability of TCP. The answer is: Almost. It can be done with an experimental TCP variant called **T/TCP** (**Transactional TCP**).

The central idea here is to modify the standard connection setup sequence slightly to allow the transfer of data during setup. The T/TCP protocol is illustrated in the figure (b). The client's first packet contains the *SYN* bit, the request itself, and the *FIN*. In effect it says: I want to establish a connection, here is the data, and I am done.

When the server gets the request, it looks up or computes the reply, and chooses how to respond. If the reply fits in one packet, it gives the reply of figure (b) which says: I acknowledge your *FIN*, here is the answer, and I am done. The client then acknowledges the server's *FIN* and the protocol terminates in three messages.

If the result is larger than 1 packet, the server also has the option of not turning on the *FIN* bit, in which case it can send multiple packets before closing its direction.

It is probably worth mentioning that T/TCP is not the only proposed improvement to TCP. Another proposal is **SCTP** (**Stream Control Transmission Protocol**). Its features include message boundary preservation, multiple delivery modes, multihoming (backup destinations), and selective acknowledgements.