INTERNATIONAL ATOMIC ENERGY AGENCY
UNITED NATIONS EDUCATIONAL, SCIENTIFIC AND CULTURAL ORGANIZATION
INTERNATIONAL CENTRE FOR THEORETICAL PHYSICS
I.C.T.P., P.O. BOX 586, 34100 TRIESTE, ITALY, CABLE: CENTRATOM TRIESTE

H4.SMR/585-17

FIRST INTERNATIONAL SCHOOL ON COMPUTER NETWORK ANALYSIS AND MANAGEMENT

(3 - 14 December 1990)

TCP / IP
Transmission Control Protocol / Internet Protocol

B. M. Segal

CERN
Computing & Networks Division
1211 Geneva 23
Switzerland

TCP / IP

Transmission Control Protocol / Internet Protocol

-- Developed for ARPANET: starting in 1969 !!

-- Designed to connect HETEROGENEOUS systems
across HETEROGENEOUS networks.

-- Networks can be COMPOSITE ("CATENET" or "INTERNET")

-- NETWORK EXAMPLES:

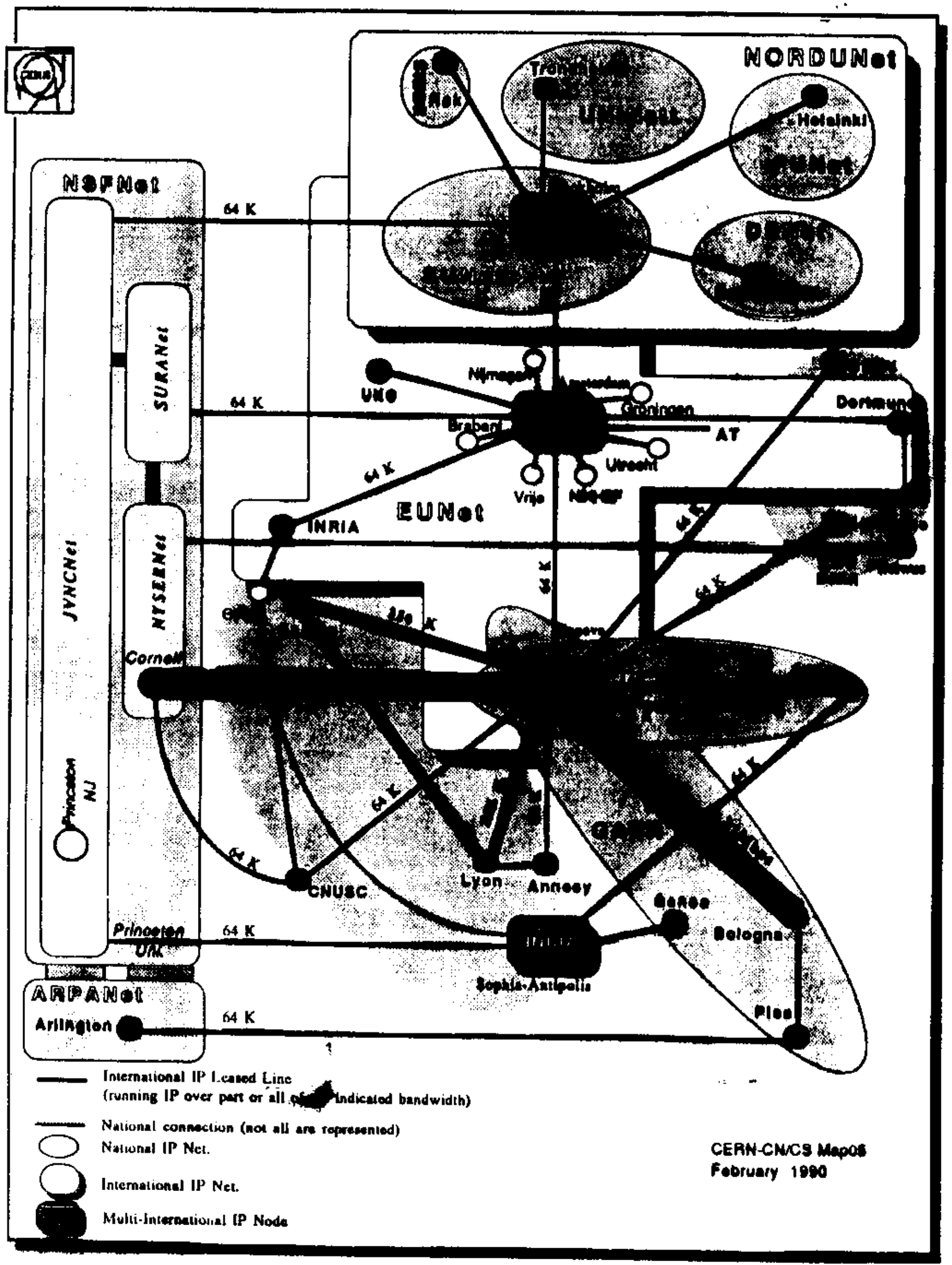
Wide-area Terrestial

Wide-area Satellite / Radio

Local-area (ETHERNET, Token, etc.)

Etc.....

Main International Leased Lines using IP in Europe (incl. ordered lines)



(Prepared for CERN internal use from available information. Can be obtained from F.G.de Bilio CERN-CN/CS)

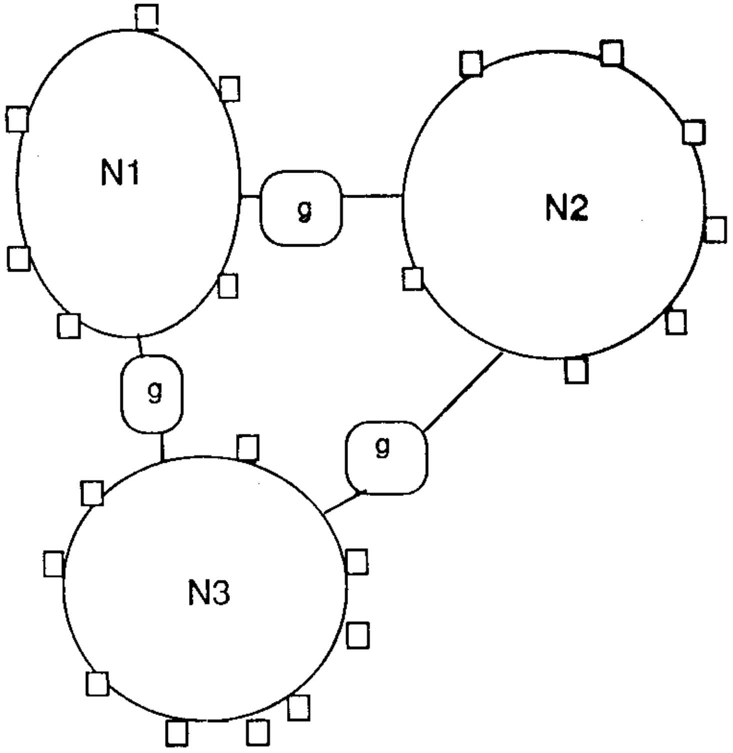
PROTOCOL LAYERS

TCP/IP	ISO / OSI
4. APPLICATION	7. APPLICATION
--	6. PRESENTATION
3. SERVICE	5. SESSION
--	4. TRANSPORT
2. INTERNET	3. NETWORK
--	2. LINK
1. NETWORK	1. PHYSICAL
--	

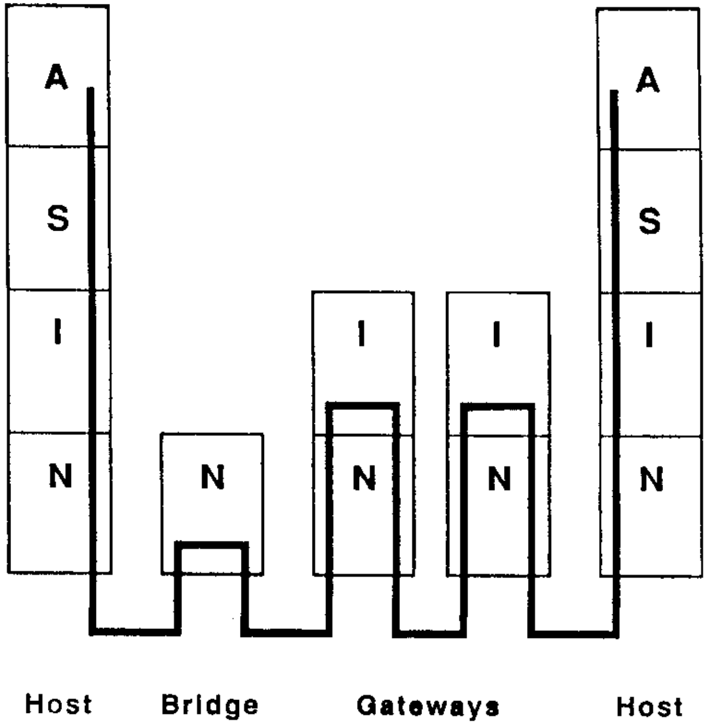
QUOTE: If you know what you are doing,
3 layers are enough...

If you don't...
even 17 layers won't help.

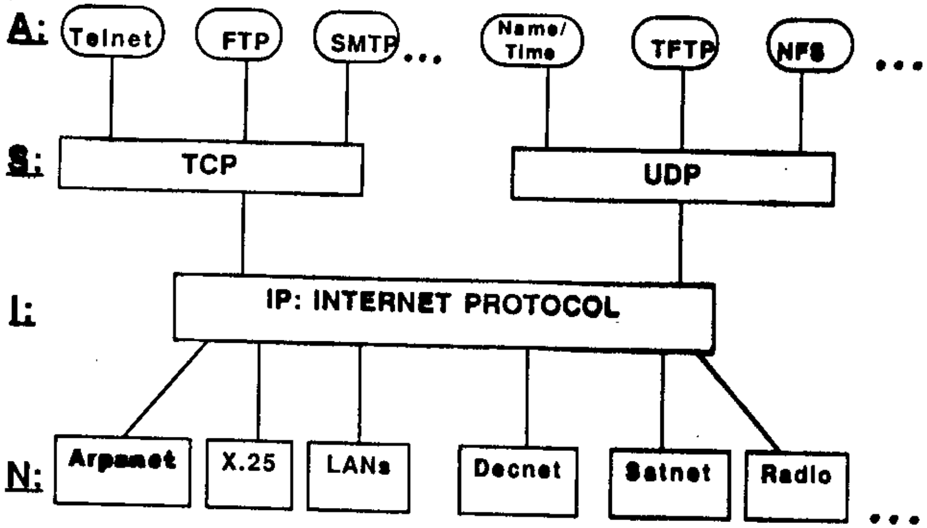
INTERNET ARCHITECTURE



INTERNET COMMUNICATION



TCP/IP PROTOCOL SUITE



What Are the Services You Get?

-- Remote login to/from systems (telnet/rlogin).

-- File transfer (ftp or rcp protocols).

-- Distributed file system (NFS).

-- Distributed window system (X).

-- Remote procedure call (Apollo NCS, Sun RPC...)

-- Task-task programming interface (BSD sockets).

-- Remote command execution (rsh protocol).

-- Remote printing (lpr protocol).

-- Electronic mail/conferencing/news (sendmail).

-- Remote file backup.

-- .... etc... etc...

The following command script illustrates how one might use FTP to retrieve RFCs under 4.3 BSD UNIX:

```
--------------------------------------------------------------------
#! /bin/sh
#
# rfc - 4.3 BSD UNIX (Bourne) shell script to obtain copies of RFCs,
#       keeping a local cache for subsequent requests.
#
# use: rfc number [number...]
#
PATH=/bin:/usr/bin:/usr/ucb
PUB=/usr/pub/RFC
for i
do      if test ! -r $PUB/$i -o $i = "-index"
        then echo Retrieving RFC $i from SRI-NIC.ARPA >&2
#
# invoke FTP under 4.3 BSD UNIX and feed it retrieval commands as input.
#
                ftp -n SRI-NIC.ARPA  >/dev/null 2>&1 <<!
user anonymous guest
get <rfc>rfc$i.txt $PUB/$i
quit
!
#
# Have obtained file; give copy to user if retrieval was successful.
#
        if test -r $PUB/$i
        then cat $PUB/$i
        else echo Could not retrieve RFC $i 1>&2
        fi
done
--------------------------------------------------------------------
```

The script shown above does more than use FTP to retrieve an RFC. It leaves a copy of the RFC in directory */usr/pub/RFC*. The advantage of keeping a local copy of an RFC is that subsequent requests are much faster than the first because they do not use FTP nor do they pass information across the Internet. If the script finds one of the requested RFCs in the cache, it merely presents the user with a copy. Note that the script does not look in the cache when retrieving the special file *-index* because the index contains a list of all RFCs that changes as new RFCs appear.

Conceptually, each address is a pair (*netid*, *hostid*), where *netid* identifies a network, and *hostid* identifies a host on that network. In practice, Internet addresses have three† primary forms, as Figure 4.1 shows. Given an Internet address, its class can be determined from the three high-order bits, with two bits being sufficient to distinguish among the primary classes. Class *A* addresses, which are used for the handful of networks that have more than 2^{16} (i.e., 65,536) hosts, devote 7 bits to netid and 24 bits to hostid. Class *B* addresses, which are used for intermediate size networks that have between 2^8 (i.e., 256) and 2^{16} hosts, allocate 14 bits to the netid and 16 bits to the hostid. Finally, class *C* networks, which have less than 2^8 hosts, allocate 22 bits to the netid and only 8 bits to the hostid. Note that the Internet address has been defined in such a way that it is possible to extract the hostid or netid portions in constant time. Gateways, which base routing on the netid, depend on such efficient extraction.

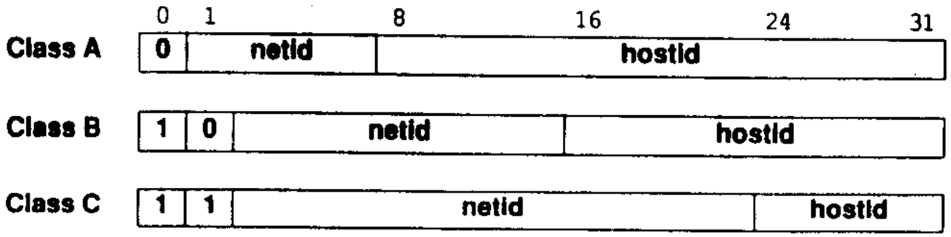


Figure 4.1 The three primary forms of Internet addresses.

Datagram Format

Now that the basic datagram content has been described, we will look at the fields in more detail. Figure 7.4 shows the arrangement of fields in a datagram:

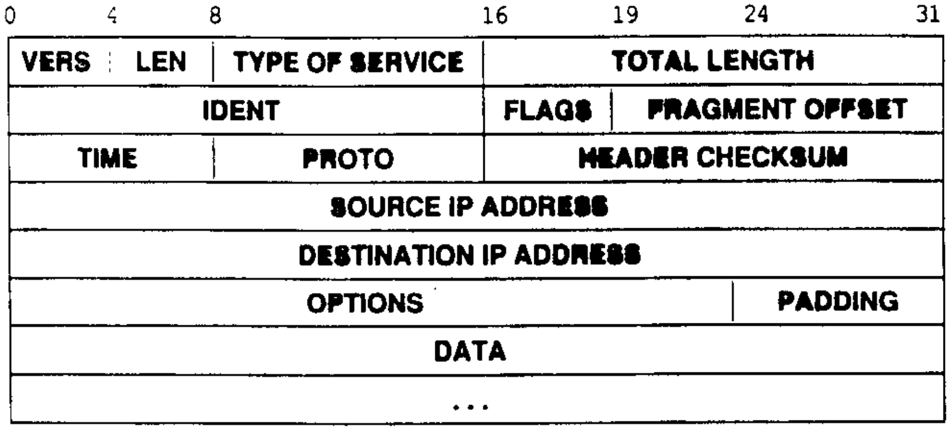


Figure 7.4 Format of an Internet datagram, the basic unit of transfer on the Internet.

The ICMP *TYPE* field defines the meaning of the message and the format of the rest of the packet. The types include:

Type Field	ICMP Message Type
0	Echo Reply
3	Destination Unreachable
4	Source Quench
5	Redirect (change a route)
8	Echo Request
11	Time Exceeded for a Datagram
12	Parameter Problem on a Datagram
13	Timestamp Request
14	Timestamp Reply
15	Information Request
16	Information Reply
17	Address Mask Request
18	Address Mask Reply

The *CODE* field in a destination unreachable message contains an integer that further describes the problem. Possible values are:

Code Value	Meaning
0	Network Unreachable
1	Host Unreachable
2	Protocol Unreachable
3	Port Unreachable
4	Fragmentation Needed and DF set
5	Source Route Failed

A gateway sends network or host unreachable messages when it cannot route or deliver datagrams. Destinations may be unreachable because hardware is temporarily out of service, because the sender specified a nonexistent destination address, or (in rare circumstances) because the gateway does not have a route to the destination network. ICMP includes a short prefix of the datagram that caused the problem so protocol software at the original source can know exactly which datagram caused the problem. The meaning of protocol and port unreachable messages will become clear when we study how higher level protocols use abstract destination points called "ports".

ARP Protocol Format

Unlike most protocols, the data in ARP packets does not have a fixed-format header. Instead, the message is designed to be useful with a variety of network technologies, so early header fields contain counts that specify lengths of succeeding fields. In fact, ARP can be used with arbitrary physical addresses and arbitrary protocol addresses. The example in Figure 5.3 below shows the 28-octet ARP message format used on Ethernet hardware (where physical addresses are 48-bits or 6 octets long), when resolving DARPA Internet protocol addresses (4 octets long). Unlike most of the Internet protocols, the variable-length fields in ARP packets do not align on 32-bit boundaries, making the diagram difficult to read. For example, the sender's hardware address, labeled *SENDER HA*, occupies 6 contiguous octets, so it spans two lines in the diagram. Nevertheless, we have chosen this format because it is standard throughout the Internet literature.

0 8 16 31

HARDWARE		PROTOCOL
HLEN	PLEN	OPERATION
SENDER HA (octets 0-3)		
SENDER HA(octets 4-5)		SENDER IA (octets 0-1)
SENDER IA (octets 2-3)		TARGET HA (octets 0-1)
TARGET HA (octets 2-5)		
TARGET IA (octets 0-4)		

Figure 5.3 The format of ARP/RARP messages used for Internet-to-Ethernet address resolution.

Format Of UDP Messages

Each UDP message is called a *user datagram* and consists of two parts as Figure 11.1 shows: a UDP header and UDP data area.

UDP header	UDP data area

Figure 11.1 The two components of a UDP message. Such messages are called *user datagrams*.

The user datagram header is divided into four 16-bit fields that specify the port from which the message was sent, the port to which the message is destined, the message length, and a UDP checksum. Figure 11.2 gives the details, showing a UDP datagram in 32-bit segments:

0 16 31

SOURCE PORT	DESTINATION PORT
LENGTH	UDP CHECKSUM

Figure 11.2 The format of fields in the UDP datagram header.

The pseudo header used in the UDP checksum computation consists of 12 octets arranged as Figure 11.3 shows:

0 8 16 31

SOURCE IP ADDRESS		
DESTINATION IP ADDRESS		
ZERO	PROTO	UDP LENGTH

Figure 11.3 The 12 octets of the pseudo header used during UDP checksum computation.

Figure 12.1 shows how the simplest positive acknowledgement protocol transfers data.

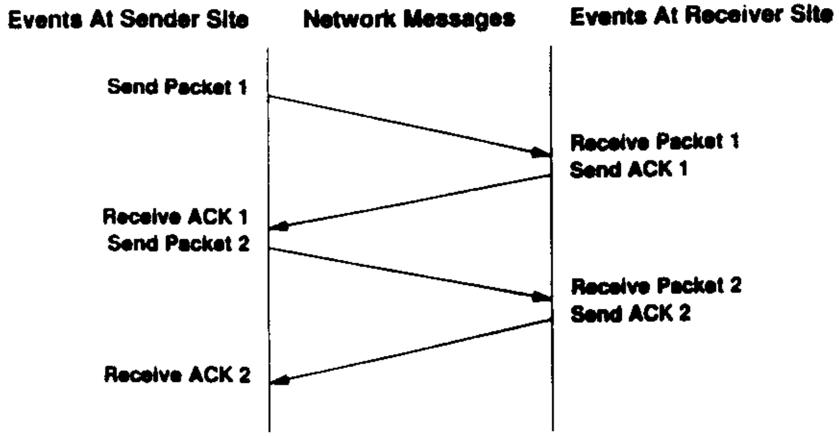


Figure 12.1 A protocol using positive acknowledgement with retransmission in which the sender awaits an acknowledgement for each packet sent. Vertical distance down the figure represents increasing time and diagonal lines across the middle represent network packet transmission.

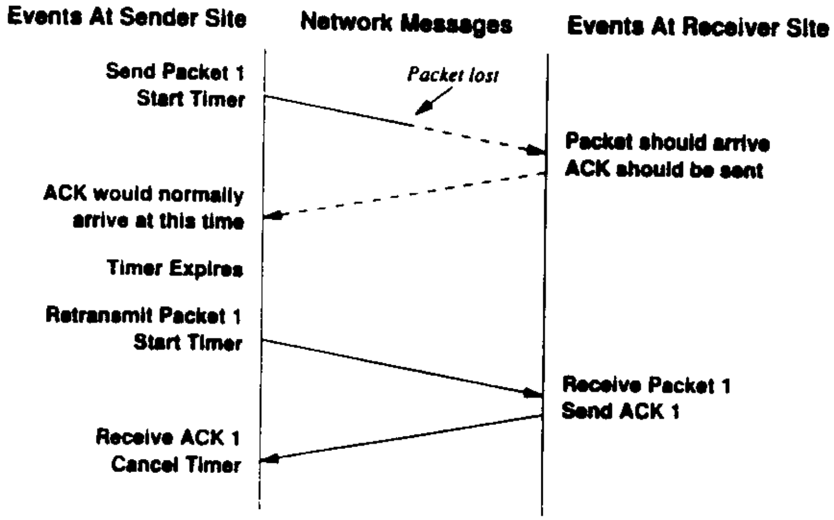


Figure 12.2 Timeout and retransmission that occurs when a packet is lost. The dotted lines show the time that would be taken by the transmission of a packet and its acknowledgement, if the packet were not lost.

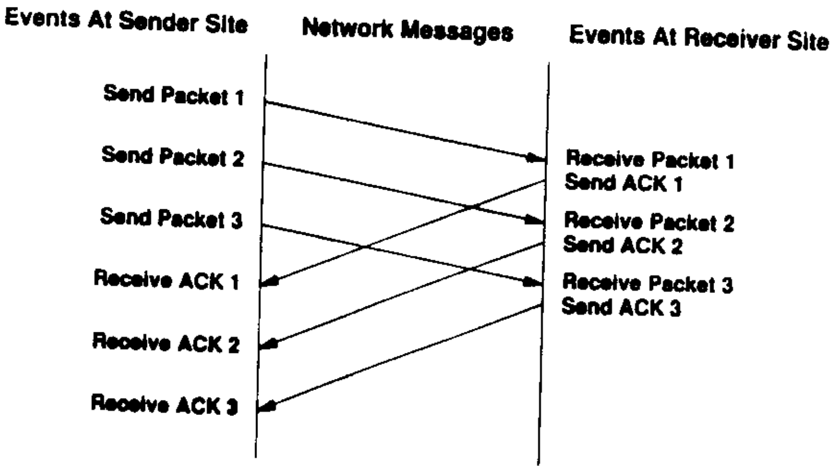


Figure 12.4 An example of three packets transmitted using a sliding window protocol. The key concept is that the sender can transmit all packets in the window without waiting for an acknowledgement.

The Idea Behind Sliding Windows

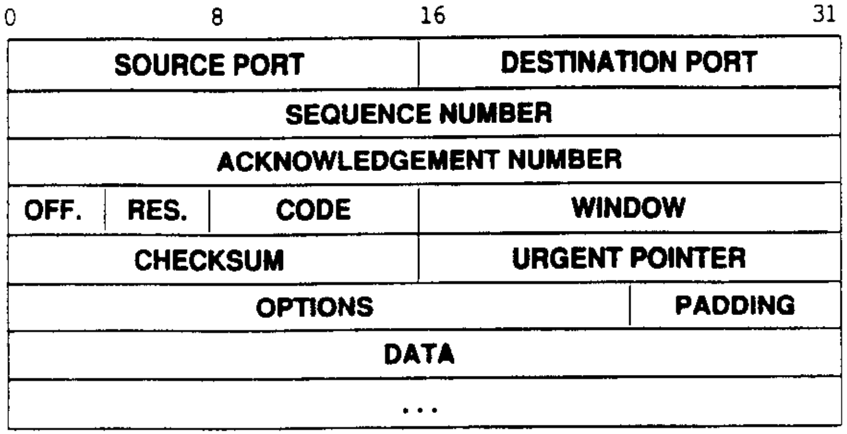


Figure 12.7 The format of a TCP segment with a TCP header followed by data. Segments are used to establish connections as well as to carry data and acknowledgements.

Bit (left to right)	Meaning
URG	Urgent pointer field is valid
ACK	Acknowledgement field is valid
PSH	This segment requests a push
RST	Reset the connection
SYN	Synchronize sequence numbers
FIN	Sender has reached end of its byte stream

TCP software advertises how much data it is willing to accept every time it sends a segment by specifying its buffer size in the *WINDOW* field. Window advertisements provide another example of piggybacking because they accompany all segments including those carrying data as well as those carrying only an acknowledgement.

TCP allows the sender to specify that some data is *urgent*, meaning that it should be delivered as quickly as possible. The protocol specifies that when urgent data is found, the receiving TCP should notify whatever application program is associated with the connection to go into "urgent" mode. After all urgent data has been received, TCP tells the application program to return to normal operation. Typically, urgent data contains messages other than normal data. For example, urgent traffic might include keyboard interrupt signals. Such traffic is often referred to as *out of band* traffic.

Protocol Dependencies

The chart in Figure 19.4 shows dependencies among the major protocols we have discussed. Each enclosed polygon corresponds to one protocol and resides directly above the polygons representing protocols that it uses. For example, the mail protocol, SMTP, depends on TCP, which depends on IP. Both ARP and RARP appear in the diagram, even though not all machines or network technologies use them. In particular, RARP is seldom used except for diskless machines.

On most systems, application programs are limited. They can access any of the protocols that form the top level in Figure 19.4, but nothing below the TCP/UDP level. However, some systems provide special purpose mechanisms that allow an application program to interact with lower protocol layers. For example, having access to ICMP echo request and reply service is especially helpful to programmers building Internet software or network managers responsible for Internet operation and maintenance.

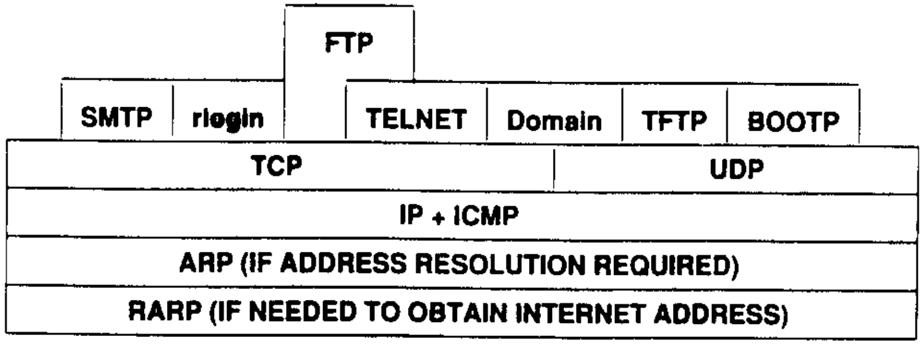

FTP
SMTP
rlogin
TELNET
Domain
TFTP
BOOTP
TCP
UDP
IP + ICMP
ARP (IF ADDRESS RESOLUTION REQUIRED)
RARP (IF NEEDED TO OBTAIN INTERNET ADDRESS)

Figure 19.4 Dependencies among higher level Internet protocols. A protocol uses those protocols that lie directly below it. Application programs can use all protocols above IP.

Decimal	Keyword	Description
0		Reserved
1-4		Unassigned
5	RJE	Remote Job Entry
7	ECHO	Echo
9	DISCARD	Discard
11	USERS	Active Users
13	DAYTIME	Daytime
15	NETSTAT	Who is up or NETSTAT
17	QUOTE	Quote of the Day
19	CHARGEN	Character Generator
20	FTP-DATA	File Transfer Protocol (data)
21	FTP	File Transfer Protocol
23	TELNET	Terminal connection
25	SMTP	Simple Mail Transport Protocol
37	TIME	Time
39	RLP	Resource Location Protocol
42	NAMESERVER	Host Name Server
43	NICNAME	Who Is
53	DOMAIN	Domain Name Server
67	BOOTPS	Bootstrap Protocol Server
68	BOOTPC	Bootstrap Protocol Client
69	TFTP	Trivial File Transfer
75		any private dial out service
77		any private RJE service
79	FINGER	Finger
95	SUPDUP	SUPDUP Protocol
101	HOSTNAME	NIC Host Name Server
102	ISO-TSAP	ISO-TSAP
113	AUTH	Authentication Service
117	UUCP-PATH	UUCP Path Service
123	NTP	Network Time Protocol
133-159	Unassigned	
160-223	Reserved	
224-241	Unassigned	
247-255	Unassigned	

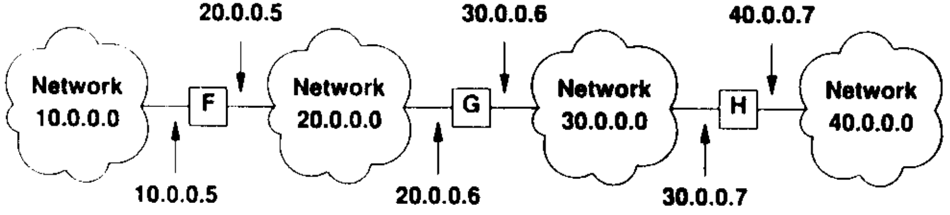
Figure 12.11 Examples of currently assigned TCP port numbers. To the extent possible, protocols like UDP use the same numbers.

An example will make the SMTP exchange clear. Suppose user Smith at host Alpha.EDU sends a message to users Jones, Green, and Brown at host Beta.GOV. The SMTP client software on host Alpha.EDU contacts the SMTP server software on host Beta.GOV and begins the following exchange shown in Figure 19.3.

```
R: 220 Beta.GOV Simple Mail Transfer Service Ready
S: HELO Alpha.EDU
R: 250 Beta.GOV

S: MAIL FROM:<Smith@Alpha.EDU>
R: 250 OK

S: RCPT TO:<Jones@Beta.GOV>
R: 250 OK

S: RCPT TO:<Green@Beta.GOV>
R: 550 No such user here

S: RCPT TO:<Brown@Beta.GOV>
R: 250 OK

S: DATA
R: 354 Start mail input; end with <CR><LF>.<CR><LF>
S: ...sends body of mail message...
S: ...continues for as many lines as message contains
S: <CR><LF>.<CR><LF>
R: 250 OK

S: QUIT
R: 221 Beta.GOV Service closing transmission channel
```

Figure 19.3 Example of SMTP transfer from Alpha.EDU to Beta.GOV. Lines that begin with "S:" are transmitted by the sender (Alpha), while lines that begin "R:" are transmitted by the receiver. In the example, machine Beta.GOV does not recognize intended recipient Green.

Figure 8.1 shows an example Internet that consists of 4 networks and 3 gateways. In the figure, the routing table gives the routes that gateway *G* uses. Because *G* connects directly to networks 20.0.0.0 and 30.0.0.0, it can reach any host on those networks directly (possibly using ARP to find physical addresses). Given a datagram destined for a host on network 40.0.0.0, *G* routes it to address 30.0.0.7, the address of gateway *H*. *H* will then deliver the datagram directly. *G* can reach address 30.0.0.7 because it attaches directly to network 30.0.0.0.



(a)

TO REACH HOSTS ON NETWORK	ROUTE TO THIS ADDRESS
20.0.0.0	DELIVER DIRECT
30.0.0.0	DELIVER DIRECT
10.0.0.0	20.0.0.5
40.0.0.0	30.0.0.7

(b)

Figure 8.1 (a) An example Internet with 4 networks and 3 gateways, and (b) the routing table for gateway *G*.

Subnet Addresses

The third technique used to allow a single network address to span multiple physical networks is called ***subnet addressing***, or ***subnet routing***. Subnetting is the most widely used of the three techniques because it is the most general and because it has been standardized.

The easiest way to understand subnet addressing is to imagine that a site has a single class *B* IP network address assigned to it, but it has two or more physical networks. Only local gateways know that there are two physical nets and how to route traffic among them; the core gateways route all traffic as if there is a single network. Figure 16.3 shows an example.

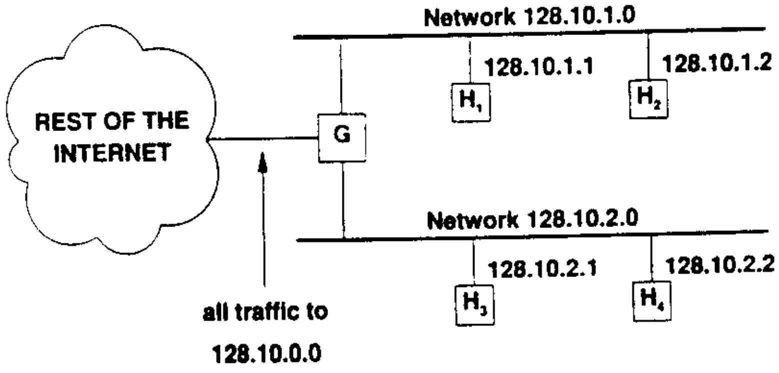


Figure 16.3 A site with two physical networks using subnet addressing to span them with a single class *B* network address. Gateway *G* accepts all traffic for net 128.10.0.0 and chooses a physical network based on the third octet of the address.

The Final Algorithm

Taking into account everything we have said, the IP routing algorithm becomes:

```
                         Algorithm:

        Route_IP_Datagram (datagram, routing_table)

    Extract destination IP address, $I_D$, from datagram
    Compute IP address of destination network, $I_N$
    If $I_N$ matches any direct connected network address
            send datagram to destination over that network;
            (This involves resolving $I_D$ to a physical address,
            encapsulating datagram, and sending the frame.)
    else if $I_D$ appears as host-specific route
            route datagram as specified in the table;
    else if $I_N$ appears in routing table
            route datagram as specified in the table;
    else if a default route has been specified
            route datagram to the default gateway;
    else declare a routing error;
```

Figure 8.2 The IP routing algorithm. Given an IP datagram and a routing table, this algorithm selects a next machine to which the datagram should be sent. Routing tables always specify a next machine that lies on a directly connected network.

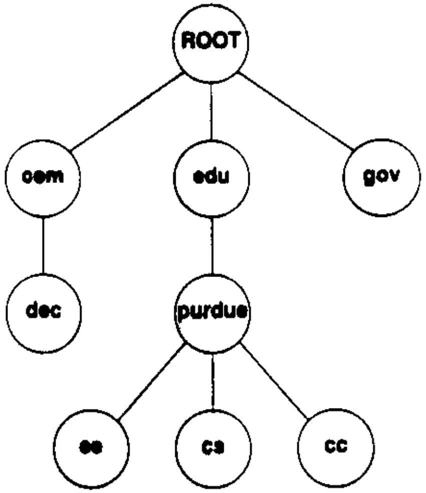


Figure 18.2 The conceptual arrangement of domain name servers in a tree that corresponds to the hierarchy of name authority. Each node represents a name server that handles names for a single subdomain.

ULTRIX DECnet-Internet Gateway

COMMANDS IMPLEMENTED:

From DECnet (VMS) side:

COPY, APPEND, DELETE,
DIRECTORY, TYPE,
SET HOST, MAIL

From Internet (ULTRIX) side:

FTP, TELNET, MAIL

ULTRIX DECnet-Internet Gateway

COMMAND DETAILS:

From DECnet (VMS) side:

```
COPY  loc_file  GATE"user@inet pass"::"rem_file"

COPY  *.c  GATE"user@inet pass"::

DIR   GATE"user@inet pass"::

DELE  GATE"inet!user pass"::"*.h"


MAIL  GATE::"user@host.domain"


SET HOST GATE

    Gate login:  inet!
    -----------

    inet: login:  ...
    ------------
```

ULTRIX DECnet-Internet Gateway

COMMAND DETAILS:

From Internet (ULTRIX) side:

```
MAIL:    mail user%node.dnet@gate


FTP:     ftp gate

         Gate Name:  node::username
         ----------
         Password:      ...
         ---------


TELNET:  telnet gate

         Gate login:  node::
         -----------
         node: login:  ...
         ------------
```

TCP/IP: Recent Developments

-- Influence of UNIX (4.2BSD):
rsh, rcp, rlogin, ... NFS ...

-- Defined the "SOCKET INTERFACE", also valid for XNS, ISO ...

-- Intelligent Interface Boards:
- sockets and utilities appear in LIBRARY
- protocols actually run on the INTERFACE

-- IMPLEMENTATIONS EXIST FOR:
- All Unix 4.2/3, including Ultrix.
- Many SYS V Unix, including Cray UNICOS.
- VMS (native and Intelligent-Boards)
- IBM (VM/CMS and MVS)
- IBM PC
- Apollo, HP, most other workstations.

Etc....

"BSD SOCKET INTERFACE"

- Generic: NOT restricted to TCP/IP !!
  Typical support also for DECnet, XNS, ISO ...

- Found on VMS, VM & MVS, IBM-PC/DOS, Mac ...

- "Almost" makes a network look like files.

- Primitives are:

  socket(), bind(), connect(), listen(), accept()

  send(), recv(), sendto(), recvfrom()

  select(), shutdown()

  read(), write(), close()

SOCKET INTERFACE: CLIENT DETAILS

```
/* Create the client socket. */

    s = socket(AF_INET, SOCK_STREAM, 0);
or:
    s = socket(AF_ISO,  SOCK_DGRAM, 0);


/* Bind the client address to the socket. */
 * (optional - otherwise system will do). */

    bind(s, &myaddr_in, sizeof(myaddr_in));


/* Try to connect to the remote server at
 * the address built into srvaddr. */

    connect(s, &srvaddr_in, sizeof(srvaddr_in));


/* Now, shutdown connection for further sends.
 * This causes the server to receive an EOF
 * condition after it has received all data
 * have been sent so far, indicating that we
 * will not be sending any further data. */

    shutdown(s, 1);
```

SOCKET INTERFACE: SERVER DETAILS

```
/* Create the listen socket. */

   ls = socket (AF_INET, SOCK_STREAM, 0);
or:
   ls = socket (AF_ISO,  SOCK_DGRAM,  0);


/* Bind the listen address to the socket. */

   bind(ls, &myaddr_in, sizeof(myaddr_in));


/* Initiate the listen on the socket.
 * The listen backlog is set to 5, which
 * is the largest currently supported. */

   listen(ls, 5);


/* The accept call will block until a new
 * connection arrives.  Then, it will
 * return the address of the connecting
 * peer, and a new socket descriptor, s,
 * for that connection. */

   s = accept(ls, &peeraddr_in, &addrlen);
```

SOCKET INTERFACE: SELECT CALL

```
/* Set time limit for select call */

   timelim.tv_sec = (long)10;
   timelim.tv_usec = 0;

/* Select on socket s */
   readfs = (1<<s);

/* Select also on stdin (fd bit 0) */
   readfs |= 1;


/* Do the wait on combined input...     */
   i = select(s+1, &readfs, 0, 0, timelp);

   if (i==0) {
        fprintf(stderr, "Select timed out!\n");
        exit(0);
   }

   if (i<0) {
        fprintf(stderr, "Select error %d\n", i);
        exit(1);
   }
   fprintf(stderr, "Select OK...\n");

   if (readfs & 1)  ... /* Handle TTY input */

   if (readfs & !1)... /* Handle net input */
```