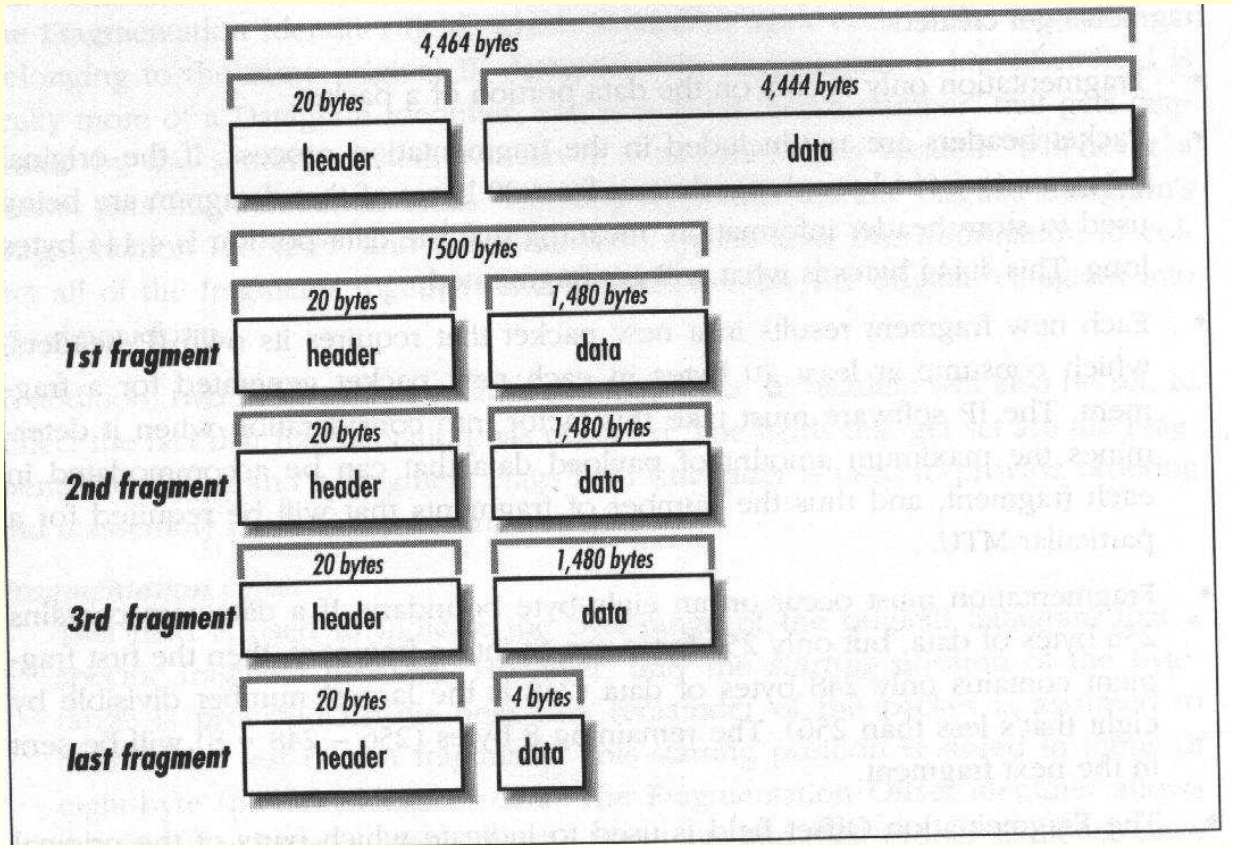
Mathematics of Fragmentation

split 4464-byte packet into fragments for Ethernet (MTU 1500)



exercise: write C–program for fragment offset computation

exercise: reassembling algorithm?

Type of Service (TOS)

Value	Service	Description
0	Normal	When all of the Type-of-Service flags are off, the IP datagram is to be treated as a normal datagram, and is not to be given any special handling. Almost all IP datagrams are marked with all zeroes in the Type-of-Service field.
1	Minimize Delay	The Delay flag is used to request that IP route this packet over a network that provides lower latency than normal. This may be useful for an application such as Telnet, where the user would want to see their keystrokes echoed back to them quickly. The Delay flag may be set to either 0 (normal) or 1 (low delay).
2	Maximize Through-put	The Throughput flag is used to request that IP route this packet over a network that provides higher throughput than normal. This may be useful for an application such as FTP, where the user would want to download a lot of data very quickly. The Throughput flag may be set to 0 (normal) or 1 (high throughput).

Type of Service (TOS)

Value	Service	Description
4	Maximize Reliability	The Reliability flag is used to request that IP route this packet over a network that provides the most reliable service (perhaps as indicated by overall up-time, or by the number of secondary routes). This may be useful for an application such as NFS, where the user would want to be able to open a database on a remote server without worrying about a network failure. The Reliability flag may be set to 0 (normal) or 1 (high reliability).
8	Minimize Cost	The Cost flag was added by RFC 1349 and was not defined in RFC 791. For this reason, many systems do not recognize or use it. The Cost flag is used to request that IP route this packet over the least expensive route available. This may be useful for an application such as NNTP news, where the user would not need data very quickly. The Cost flag may be set to 0 (normal) or 1 (low cost).
15	Maximize Security	RFC 1455—an experimental specification for data-link layer security—states that this flag is used to request that IP route this packet over the most secure path possible. This may be useful with applications that exchange sensitive data over the open Internet. Since RFC 1455 is experimental, most vendors do not support this setting.

IP Remaining Fields

4–bit version (currently $4 = (0100)_2$)

4–bit header length in 32–bit multiples$\rightsquigarrow$need padding

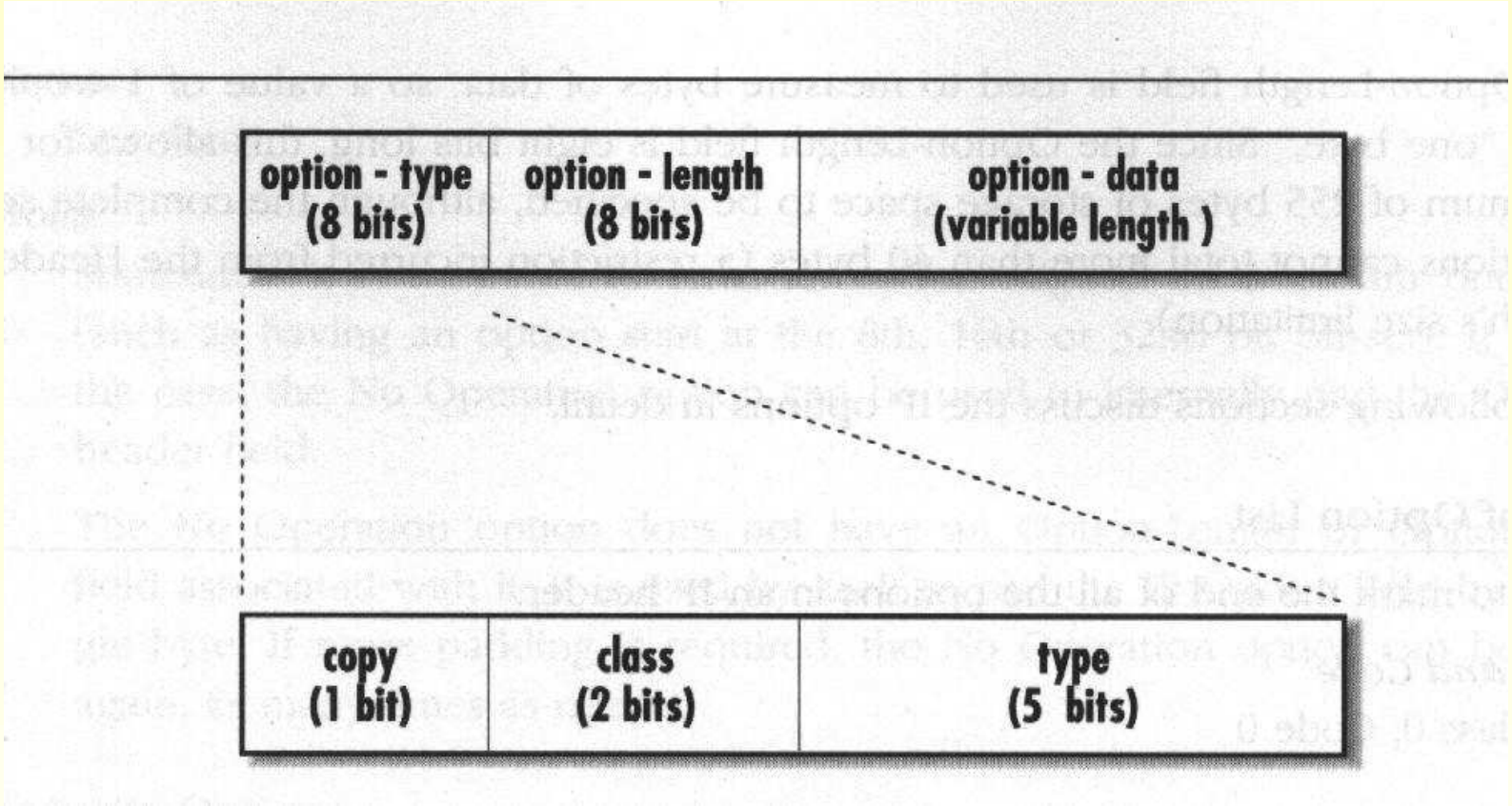
8–bit protocol identifier (ICMP,TCP,UDP,...)

32–bit Source IP

32–bit Destination IP

0–320 bit Options, mostly none

IP Options


option - type
(8 bits)
option - length
(8 bits)
option - data
(variable length)
copy
(1 bit)
class
(2 bits)
type
(5 bits)

Integer Representation

high endian

$$x = a_k \cdot 256^k + a_{k-1}256^{k-1} + \ldots + a_1 \cdot 256 + a_0$$

a_3	a_2	a_1	a_0

little endian

$$x = a_0 + a_1 \cdot 256 + \ldots + a_{k-1}256^{k-1} + a_k \cdot 256^k$$

a_0	a_1	a_2	a_3

network byte order (is high endian)

host byte order (is high or little endian)

Internet Control Message Protocol (ICMP)

RFC 792

messages for semi–permanent IP delivery errors

uses IP

destination IP = source IP of faulty packet

transient errors may be ignored:

- eventually the sender notices his error and retransmits
- next packet probably okay
- sender would use a reliable protocol if needed

Examples for Semi–Permanent Errors

(fundamental problems with the network)

network/host/port unreachable

TTL dropped to 0

fragmentation required but DON'T FRAGMENT flag set

fragment reassembly time exceeded

source quench

ICMP Message Suppressed. . .

. . . on receiving . . .

- a faulty ICMP Message
- a broadcast or multicast message
- an unspecified source address
- a fragment that is not the first of a packet

ICMP Queries (1)

additional functionality of ICMP: network probing

ECHO request / ECHO reply (RFC 1122: mandatory)

IS THAT HOST ALIVE?

```
$ ping powercrypt
PING powercrypt (192.168.1.1) : 56(84) bytes of data.
From buddy (192.168.1.5): icmp_seq=1 Destination Host Unreac
$ ping localhost
PING localhost (127.0.0.1) : 56(84) bytes of data.
64 bytes from localhost : icmp_seq=1 ttl=64 time=62 usec
```

exercise: options of ping

ICMP Queries (2)

timestamp request / response (RFC 1122: optional)

network timings

use UTC (universal time coordinated)

Application: Traceroute

send IP packets increasing TTL values

record sources of ICMP time exceeded messages

average latency by 3 messages, different UDP ports

maybe blocked $\leadsto$no ICMP response generated

use UDP or IP for this?

no final conclusion

Application: Path MTU Discovery

goal: find optimal MTU

RFC 1191

idea:

- send IP packets with decreasing size with DF flag
- await ICMP message fragentation required

extension:

- fill ICMP message with next–hop MTU
- next–hop MTU can be used for next query

Application: Path MTU Discovery

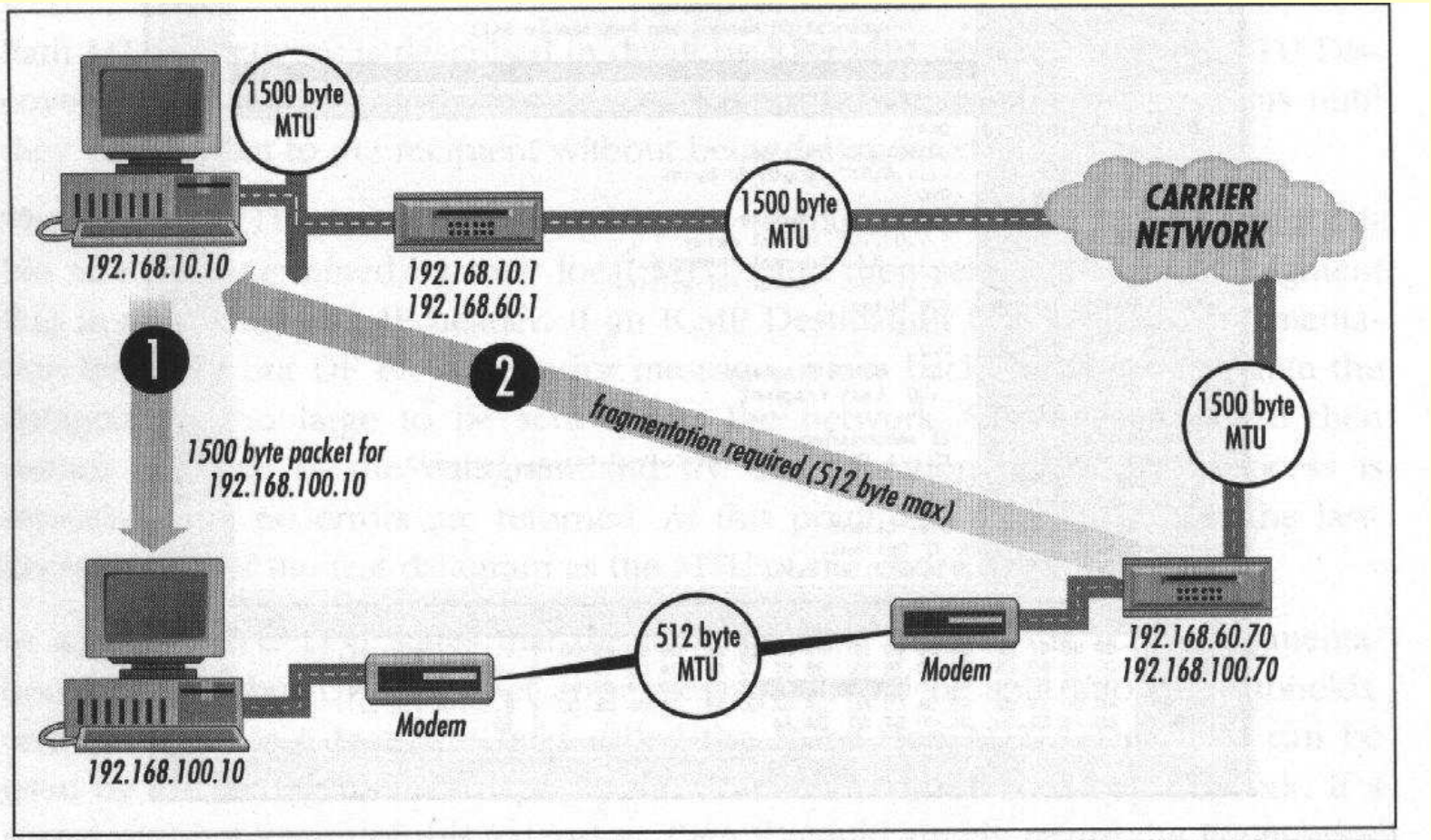


Figure 5-42. An overview of Path MTU Discovery

ICMP Message Format

predefined error messages with codes

Field	Size	Usage Notes
Message Type	1 byte	Indicates the specific ICMP error message.
Message Code	1 byte	Indicates the specific subclass of the specific ICMP error message.
Checksum	2 bytes	Used to validate the ICMP message's contents.
Message Data	4 bytes	Used for message-specific needs.
Original Headers	20–60 bytes	The full header of the IP datagram that failed.
Original Data	8 bytes	The first 64 bits of the IP datagram's data. This data will contain the Source and Destination Port fields for the transport protocol used by the sender, allowing the transport protocols to determine which specific application generated the failing datagram.

ICMP Message Types

Table 5-1. Message Types and Their Usage

Type	Message Description	Message Family	Defined In
0	Echo Reply	Query (Reply)	RFC 792
3	Destination Unreachable	Error	RFC 1122
4	Source Quench	Error	RFC 792
5	Redirect	Error	RFC 792
8	Echo Request	Query (Request)	RFC 792
9	Router Advertisement	Query (Reply)	RFC 1256
10	Router Solicitation	Query (Request)	RFC 1256
11	Time Exceeded	Error	RFC 1122
12	Parameter Problem	Error	RFC 792
13	Timestamp Request	Query (Request)	RFC 792
14	Timestamp Reply	Query (Reply)	RFC 792
17	Address Mask Request	Query (Request)	RFC 950
18	Address Mask Reply	Query (Reply)	RFC 950

ICMP Message Codes

message type "destination unreachable"

Code	Meaning
0	Network Unreachable
1	Host Unreachable
2	Protocol Unreachable
3	Port Unreachable
4	Fragmentation Required but DF Bit Is Set
5	Source Route Failed
6	Destination Network Unknown
7	Destination Host Unknown
8	Source Host Isolated (obsolete)
9	Destination Network Administratively Prohibited (obsolete)
10	Destination Host Administratively Prohibited (obsolete)
11	Destination Network Unreachable for Type-of-Service
12	Destination Host Unreachable for Type-of-Service
13	Communication Administratively Prohibited
14	Host Precedence Violation
15	Precedence Cutoff in Effect

message code network/host/port unreachable

Common Problems

- firewalls block ICMP (maybe both sides)
- first ping packet fails
  - ARP cache empty
  - Routing cache empty
  - DNS cache empty

User Datagram Protocol (UDP) – Layer 4

RFC 768

jokingly unreliable data protocol

connectionless (datagram–oriented)

analogy: postcard

basically a user interface to IP

advantages:

- performance
- ability to broadcast

local unreliability mainly negligible

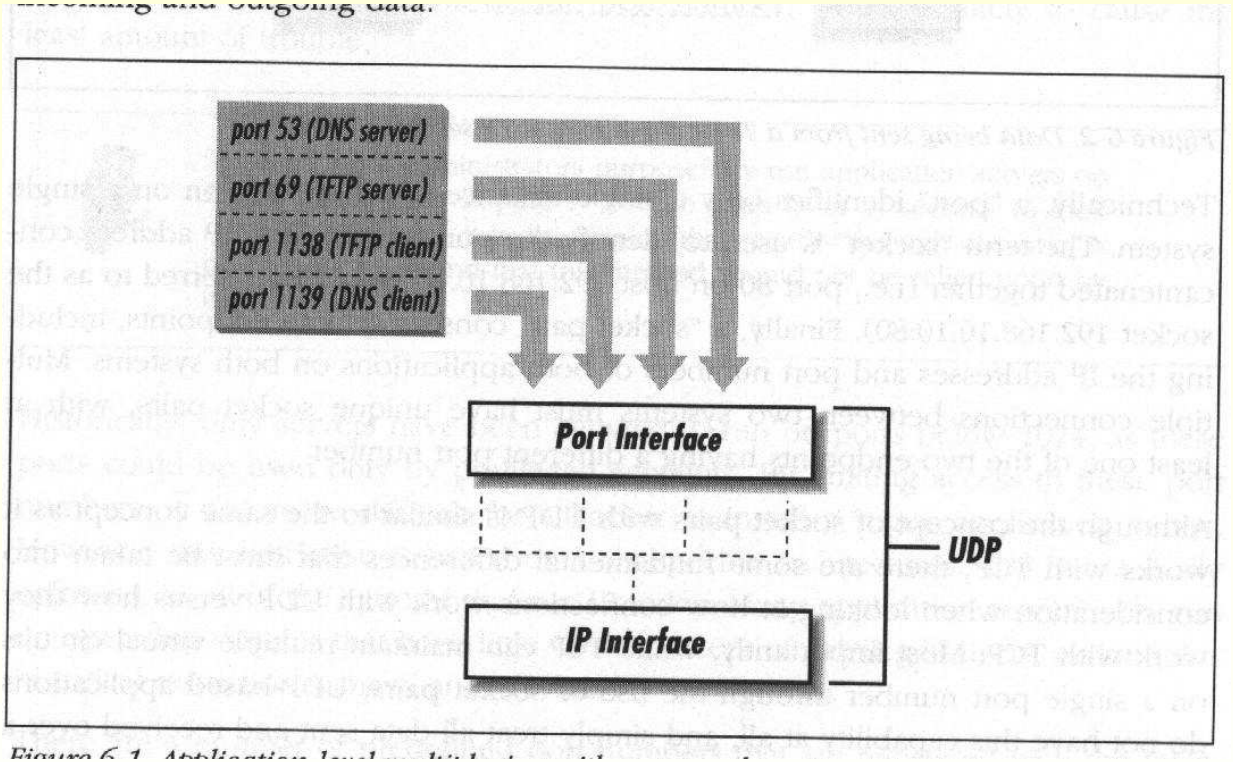
optional UDP checksum

UDP Port

interface to application program (comm endpoint)

16 bit number, notation: `host:port`

multiplexing service



Important UDP Services

- DNS (port 53)
- TFTP (port 69)
- NetBIOS (port 137)
- SNMP (port 161)
- NFS (port 2049)

see `/etc/services`

predefined port numbers $\rightsquigarrow$ IANA

`http://www.iana.org/assignments/port-numbers`

UDP Message Format

Table 6-2. The Fields in a UDP Message

Field	Bytes	Usage Notes
Source Port	2	Identifies the 16-bit port number in use by the application that is sending the data
Destination Port	2	Identifies the 16-bit target port number of the application that is to receive this data
Length	2	Specifies the size of the total UDP message, including both the header and data segments
Checksum	2	Used to store a checksum of the entire UDP message
Data	varies	The data portion of the UDP message

source port only needed for reply

max message length 65,535 bytes

max data length 65507 (20 IP-Header, 8 UDP Header)

UDP Errors

wrong checksum $\rightsquigarrow$packet

port unreachable (ICMP)

UDP traffic blocked by firewall

Common Data Structures: hostent

```
struct hostent
{
  char *h_name;                 /* Official name of host.  *
  char **h_aliases;             /* Alias list.  */
  int h_addrtype;               /* Host address type.  */
  int h_length;                 /* Length of address.  */
  char **h_addr_list;           /* List of addresses */
  #define h_addr  h_addr_list[0]  /* First address */
};

functions:
struct hostent *gethostbyname(char *host); /* NULL = invalid
struct hostent *gethostbyaddr(...); /* NULL = invalid */

file: /etc/hosts
```

... and DNS (domain name service) via network

Common Data Structures: protoent

```
struct protoent
{
  char *p_name;                 /* Official protocol name.
  char **p_aliases;             /* Alias list.  */
  int p_proto;                  /* Protocol number.  */
};
```

```
function:
struct protoent *getprotobyname(char *);
```

```
file: /etc/protocols
ip              0       IP
icmp            1       ICMP
tcp             6       TCP
udp             17      UDP
```

Common Data Structures: socket

a socket is an integer

the integer "points to" a data structure containing

- the domain, also called protocol family

  (`PF_INET, PF_INET6, PF_IPX, PF_X25...`)

- the socket type

  (`SOCK_STREAM, SOCK_DGRAM, SOCK_RAW...`)

- the protocol number

  (`ip, icmp, tcp, udp,...`) ⟵ protoent

`function:`

```
int socket(int domain, int type, int protocol);
/* error: -1 */
```

Common Data Structures: sockaddr

a socket address describes a communication endpoint (= process)

the generic sockaddr is not protocol specific:

```
struct sockaddr {
  sa_family_t     sa_family;      /* address family, AF_xxx
  char            sa_data[14];    /* generic protocol addres
};
```

with TCP/UDP we have the concept of ports

```
struct sockaddr_in {
  sa_family_t         sin_family;  /* Address family   */
  unsigned short int  sin_port;    /* Port number      */
  struct in_addr      sin_addr;    /* Internet address */
};
```

so we can cast `struct sockaddr_in *` to `struct sockaddr *`

UDP Client

1. fill sockaddr_in structure
2. create socket
3. `bind()` $\longleftarrow$ if the client receives response(s)
4. `sendto()` for sending a request to the server
5. `recvfrom()` for receiving a response
6. `close()` socket for terminating the communication

UDP Server

1. fill sockaddr_in structure
2. create socket
3. `bind()` for initialising an endpoint
4. `recvfrom()` for receiving a message
5. `sendto()` for sending a response
6. `close()` socket when terminating the server

Binding a Socket to an Endpoint

```
int bind(int s, const struct sockaddr *name, int namelen);
```

Linux: `man 2 bind`

Solaris: `man -s 3socket bind`

Sending and Receiving Data

1. **sending**

```
int sendto(int s, const void *msg, size_t len, int flag
           const struct sockaddr *to, socklen_t tolen)
```

flags: 0, MSG_DONTWAIT (<-non-blocking)

2. **receiving**

```
int recvfrom(int  s,  void  *buf,  size_t len, int flag
             struct sockaddr *from, socklen_t *fromlen
```

flags: 0, MSG_PEEK, MSG_DONTWAIT (<-non-blocking)

Analogy with File Operations

Network	C	C++
socket()	fopen()	fstream::open()
bind()	–	–
sendto()	fwrite()	fstream::write()
recvfrom()	fread()	fstream::read()
close()	fclose()	fstream::close()

Transmission Control Protocol (TCP) – Layer 4

RFC 793

TCP data entity is called segment

connection–oriented (full duplex)

analogy: phone call

advantages:

- reliability
- flow control
- congestion control

two connection endpoints $\rightsquigarrow$no broadcasts

Reliability

RFC 793:

recover from data that is damaged, lost, duplicated or out of order

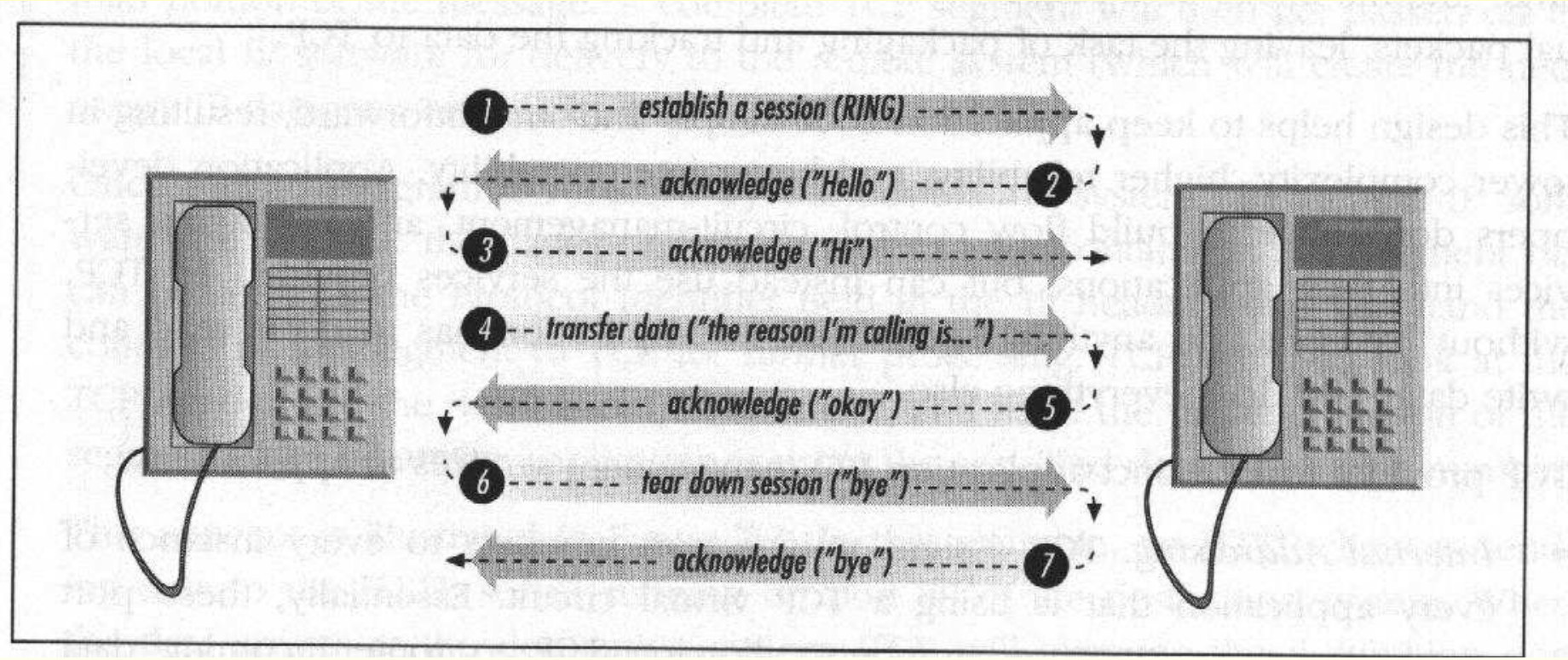
- order of sent data same as order of received data
- no data corruption
- no data loss
- no duplication of data

but use unreliable IP for this

Main Ideas for Reliability

- sequence numbers
- data checksum
- acknowledgements
- timer

Connection Establishment and Termination



Properties

stream service

- open connection
- read/write data
- close connection

analogy: open–[read/write]–close file

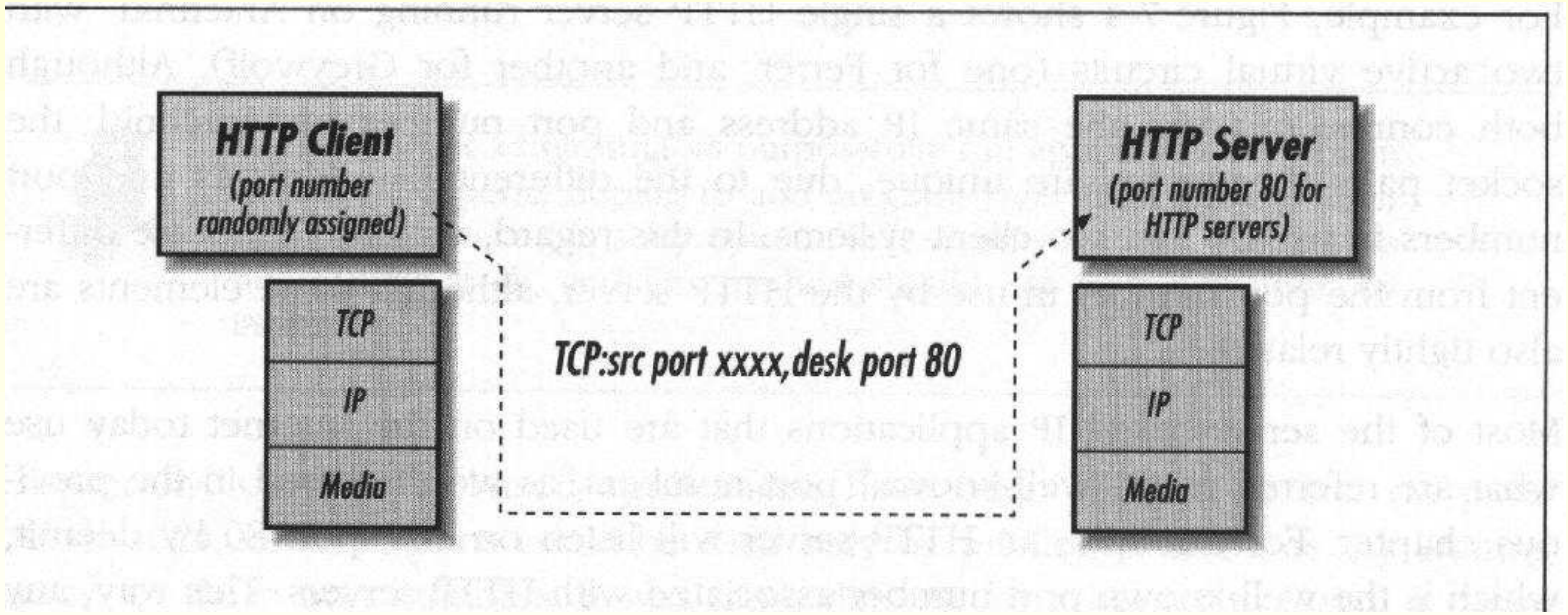
problem: end of requests/replies

TCP Port

interface to application program

16 bit number

multiplexing service



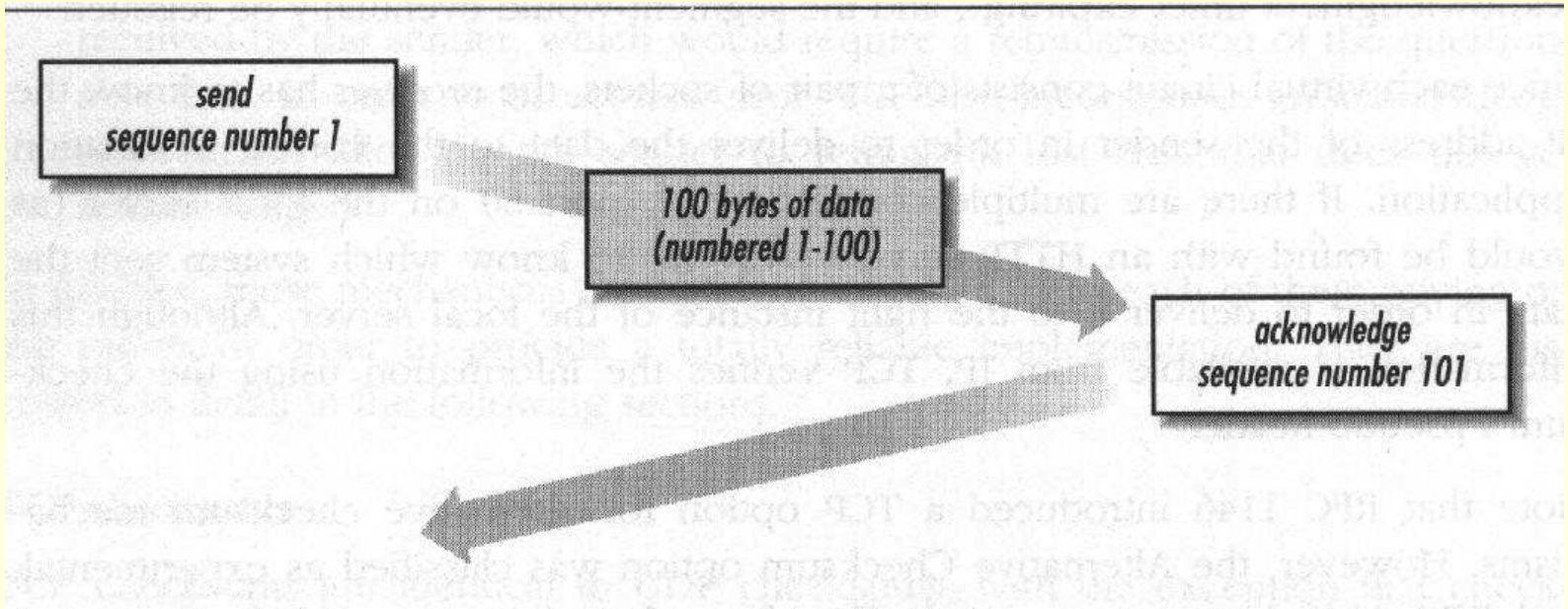
connection: pair of (host, port)

e.g. ((host1,port1),(host2,port2))

Important TCP Services

- FTP control (port 21)
- FTP data (port 20)
- TELNET (port 23)
- SMTP (port 25)
- HTTP (port 80)
- POP3 (port 110)
- NNTP (port 119)

Sequence Numbers (Idea)



Sequence Numbers

32 bit number

position in data stream

initial value randomly on each side (RFC 1122)

- avoid overlap
- security

ACK number = next expected sequence number

Acknowledgement Numbers

ACK number = next expected sequence number

Rules:

Event	Action of receiving TCP
segment arrives in order	delayed ACK (500 ms)
every two segments	send cumulative ACK immediate
segment out of order	duplicate ACK
segment fills gap partially	ACK (if lower bound changed)

Timer

measure RTT = Round Trip Time

must be adaptable to

- fast and slow connections
- changes over time

for each new packet with round trip time RTT

modify $\overline{RTT}$, the estimated RTT

$$\overline{RTT} \longleftarrow (1-x) \cdot \overline{RTT} + x \cdot RTT$$

$\rightsquigarrow$weighted average, typical value $x = \frac{1}{8}$

Connection Establishment

- passive open (server)
- active open (client)

three–way–handshake

- $A \longrightarrow B$: SYN, seq r
- $B \longrightarrow A$: SYN+ACK, seq r', ack $r+1$
- $A \longrightarrow B$: ACK, ack $r'+1$

API:

```
int connect(int sockfd, struct sockaddr *addr, int len);
```

Connection Termination

- $A \longrightarrow B$: FIN
- $B \longrightarrow A$: ACK
- $B \longrightarrow A$: . . . maybe some more data . . .
- $B \longrightarrow A$: FIN
- $A \longrightarrow B$: ACK

think of FIN as end–of–file

API:

```
int shutdown(int s, int how);
```

The TCP Finite State Machine