

ELE3305 Computer Systems & Communications Protocols

Module 6: Internetworking

Overview

- Introduction & Historical Perspective
- Standards Documents
- Protocol Layering, Sliding Windows, Client/Server Model
- Applications
 - Electronic Mail (email)
 - File Transfer Protocol (ftp)
 - Remote Terminal (telnet)
 - Hypertext (http & html)

Overview

- Protocols
 - Transmission Control Protocol (TCP)
 - Internet Protocol (IP)
 - Internet Control Message Protocol (ICMP)
 - Serial Line Internet Protocol (SLIP) and Point-to-Point Protocol (PPP)
- Addressing
 - Internet address format
 - Address to Name mapping
 - The Domain Name System (DNS)
 - Address Resolution Protocol (ARP)

Overview

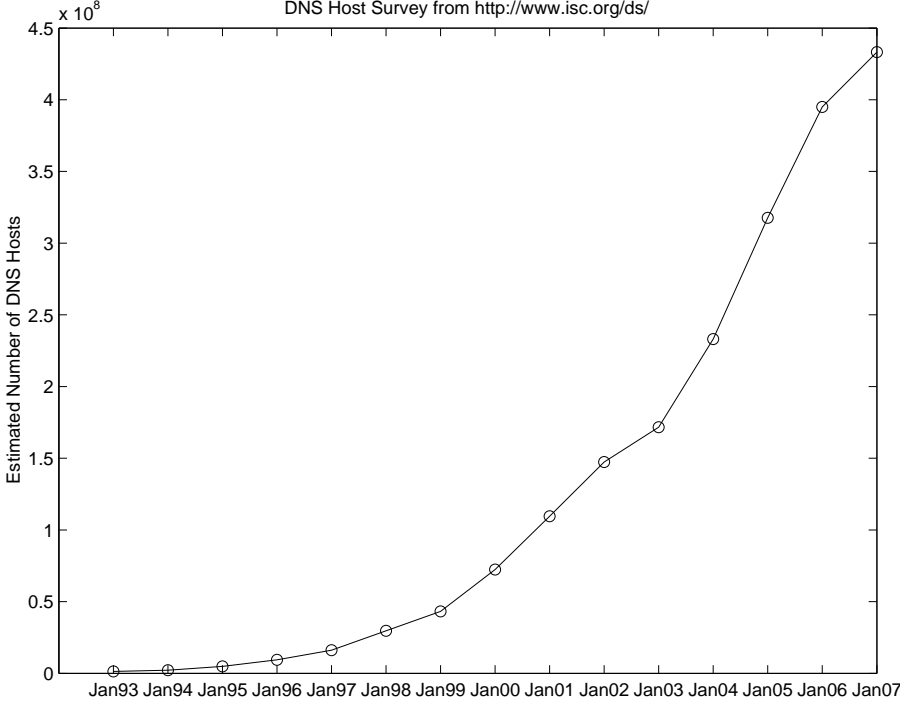
- Network Setup & Diagnostic Tools
 - Configuration Files
 - ipconfig/arp/ping/traceroute
- Network Programming – Windows & Unix
 - Berkeley Sockets Application Programming Interface (API)
 - Example client/server application
 - Windows ‘winsock’ extensions

History of the Internet

- Originally a Military Network
- DARPA (Defense Advanced Projects Research Agency)
- Designed for distributed, fault tolerant communications
- First run December 1969 with four nodes
- Honeywell minicomputers, 12K of 16-bit memory
- 56 kbps lines

Source: "Computer Networks", Tanenbaum

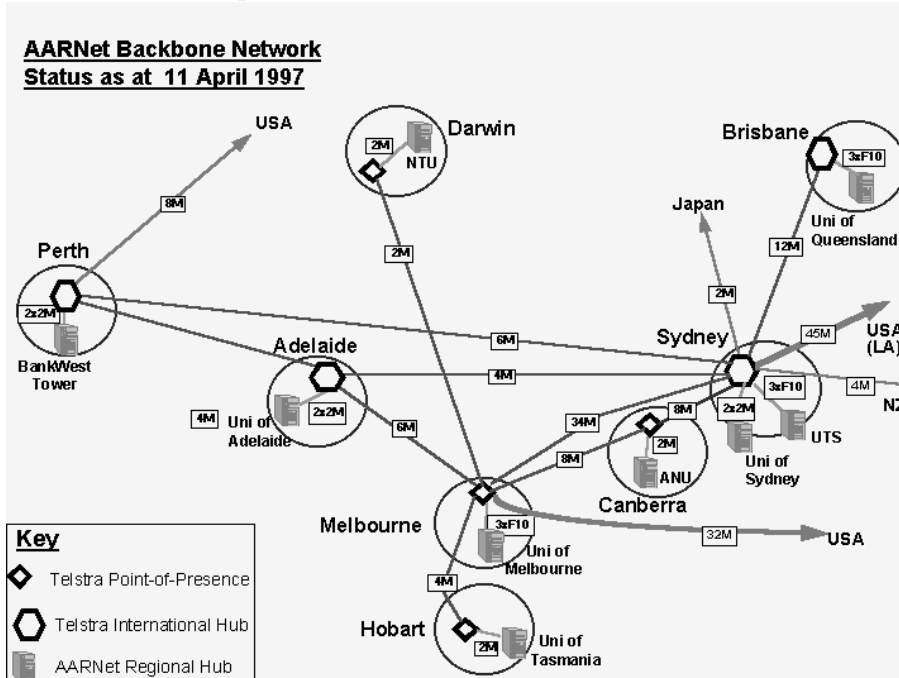
Growth of the Internet



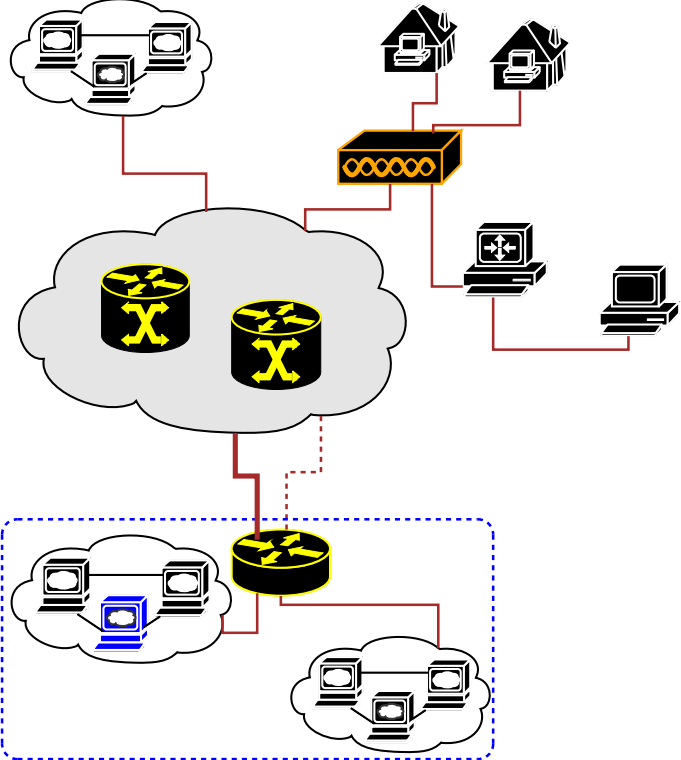
Raw data from ISC: <http://www.isc.org/ds/>

AARNet

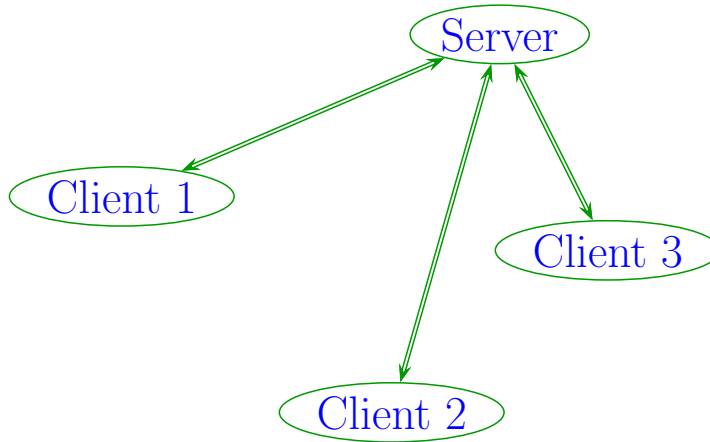
<http://www.avcc.edu.au/avcc/aarnet/>



Internet Concept

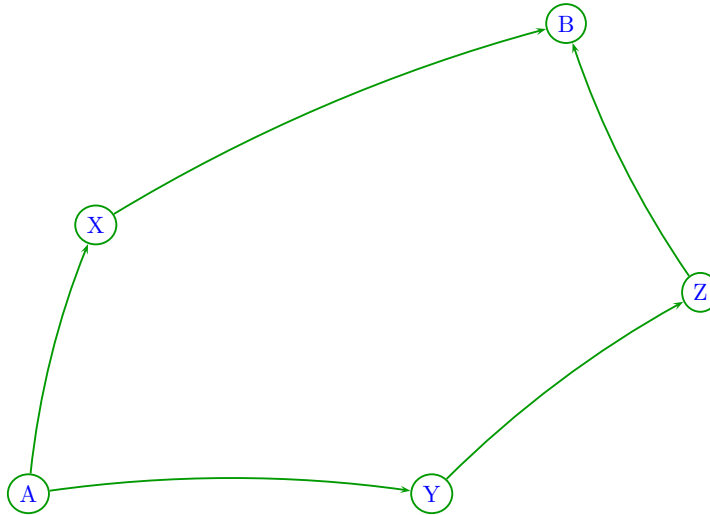


Client-Server Transaction



- Server handles many clients
- Server continually running

Routing

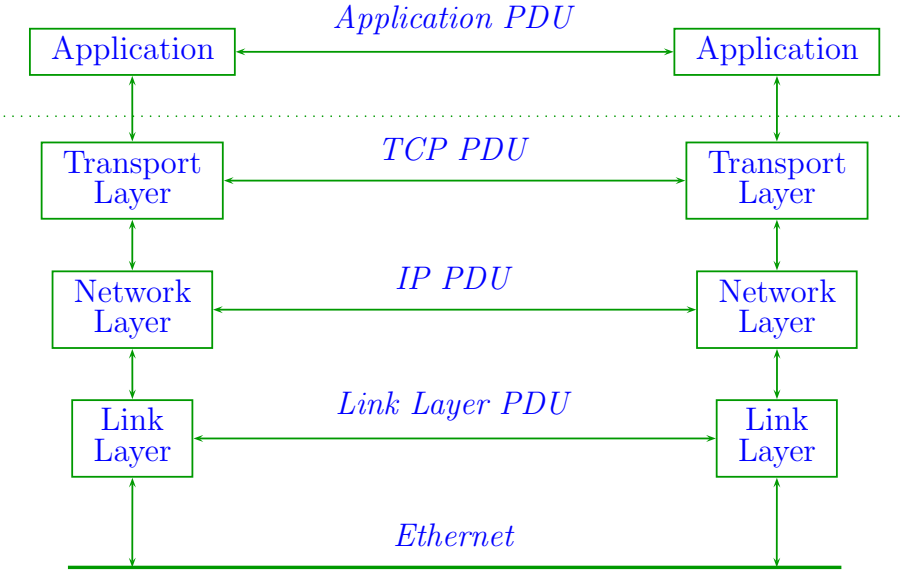


- Possibly multiple routes
- Distributed routing tables (not centralized)

Standards Documents

- Web: W3C <http://www.w3.org/>
- Internet: IETF <http://www.ietf.org/>
 - RFC = “Request For Comment”
 - Example: rfc1521.txt = standard for email encoding
- Internet Addresses: ICANN <http://www.icann.org/>
- CCITT-ITU Standards eg X.25 for Packet Switching
- Proprietary eg Novell

Protocol Stack



Protocol Issues

Some things to think about...

- How to split large chunks of data (eg files)
- What if the data gets corrupted (error checking)
- Can you rely on the other side (timeouts?)

Protocol Segments

Split data into “chunks” (Protocol Data Units or PDUs), variously called Frames, Datagrams, Segments or Packets depending on the protocol.



Exact content depends on the specific protocol

Protocol Segments

Need information such as (depends on the protocol)

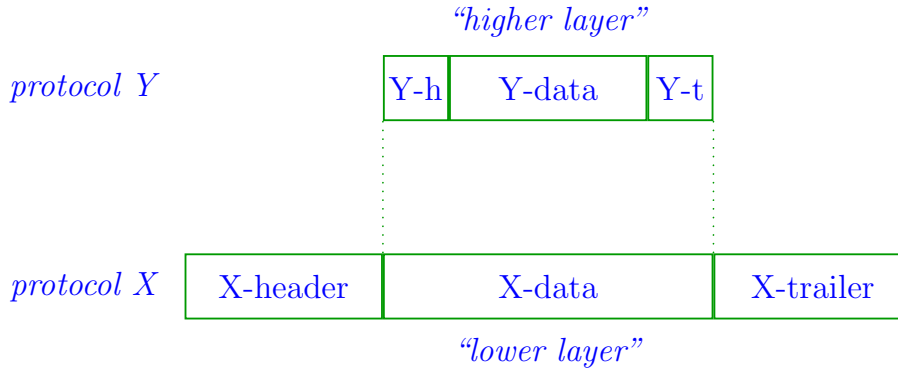
- Frame type, such as data, (negative) acknowledge, control
- Source address, Destination address
- Length of Data portion
- Error checking information

Information at the beginning (up to the DATA portion) is called the **header**.

Information after the DATA portion is called the **trailer**.

Protocol Encapsulation

Higher-layer protocols become the data payload for lower-layer protocols.
Example: TCP segments sent on IP datagrams which are sent on Ethernet frames.



Specific protocols later

An Application: email

- SMTP (Simple Mail Transfer Protocol) RFC 821
- Addresses of the form “leis@usq.edu.au”
- User agent – for example, Outlook for Windows
- sendmail daemon process
- POP - Post Office Protocol (PC environment, machines not guaranteed to be on continuously)

email continued

- A problem: links not guaranteed to be “8-bit clean”. 7-bit only, with *no* control characters (non-printable)
- How to send binary files, eg compressed, executables, bitmapped pictures etc ?
- *uuencode/uudecode* - Unix standard (old but still used)
- *BinHex* - Mac standard (old but still used)
- *base64* - MIME (Multipurpose Internet Mail Extensions)

Base64 Encoding

- Standard RFC1521
- Three, 8-bit bytes split up into four, 6-bit numbers
- 6-bit numbers encoded using a lookup table (printable characters)
- Mail application needs to be “mime-aware”

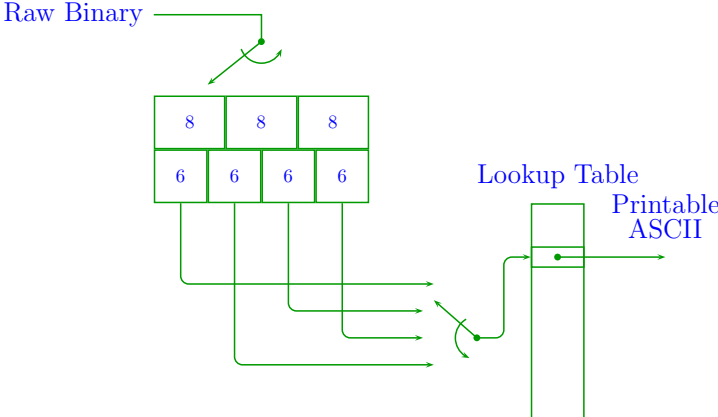
Header of message:

Content-Type: application/octet-stream; name=temp.wri

Content-Description: This is a WRITE document.

Content-Transfer-Encoding: base64

Base64 Encoding



Data portion of message:
Mb4AAACrAAAAAAA *etc*

telnet

- *telnet* - Network Terminal Protocol RFC 854
- Connects to a *port* on a remote machine
- Typically used to log in remotely to a machine.
- May be used to connect to any port, for example the timeserver port 13:
telnet zaphod.eng 13
- See *man telnet* for further details.

ftp

- *ftp* - File Transfer Protocol RFC 959
- Connects to a *port* on a remote machine for interactive file transfer.
- Common commands:
 - get, put - get a file from remote machine, put a file on remote machine
 - asc, bin - ASCII or binary mode (see later)
 - dir - directory
 - mget, mput - multiple file get/put
 - prompt - toggle interactive mode (yes/no to multiple file transfer)

ftp mode

- Binary - don't change any bytes
- ASCII - perform carriage return/line feed conversion according to host machine:
 - Unix uses line feed (LF, 0A Hex) for end-of-line
 - DOS uses carriage return (CR, 0D Hex) followed by line feed for end-of-line
 - Mac uses carriage return only.

ftp mode

Text file transferred from DOS to Unix in binary:

here is a line ^ M

here is another line ^ M

Has a carriage return, then an extra linefeed

File transferred from Unix to DOS in binary:

here is a line

here is another line

and yet another line

Has a line-feed but no carriage return

Naturally, for bitmap files, compressed files and executables, **binary mode** is necessary.

Hypertext

- “Net Browsers” – Netscape, Internet Explorer (graphical), lynx (text)
- Text, plus links to other sites/files (URL = Uniform Resource Locator)
- HTTP = HyperText Transfer Protocol
- HTML = HyperText Markup Language (.html extension)

HTML

```
<html>
```

```
<!-- a simple html example -->
```

```
<body>
```

```
  <h1>This is a heading</h1>
```

```
  <p>
```

```
  This is in a paragraph of its own.
```

```
  </p>
```

```
  <a href="http://www.usq.edu.au/">Home Page</a>
```

```
</body>
```

```
</html>
```

URL - Uniform Resource Locator

A URL is used to locate a resource on a remote machine.

`protocol:subprotocol://host.domain:port/path/file`

- **protocol** (required) The transaction protocol: http, ftp, smtp, jdbc, ...
- **subprotocol** (optional) The remote sub-protocol: seldom used.

URL - Uniform Resource Locator

- **host** (required) Specifies the specific machine. Example: `athena`
- **domain** (optional) The machine Internet domain address. Example: `usq.edu.au`
- **port** The protocol port used. Protocols have a default port (`http=80`)
- **path** (optional) The directory path. Example: `webfiles/someuser`
- **file** (optional) The requested file. Example: `index.html`

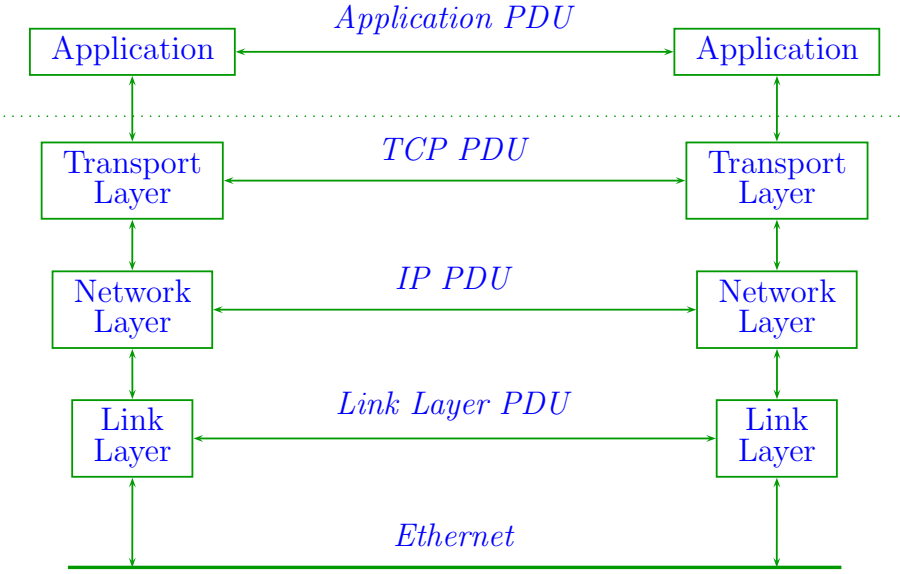
URL - Uniform Resource Locator

- **Protocol** examples include:
hypertext transfer protocol (http),
file transfer protocol (ftp),
rmi (remote method invocation),
jdbc (Java database connectivity)
- **Subprotocol** is seldom used, but may be specified for a two-step communication such as odbc (open database connectivity).
- **host.domain** are usually combined: `athena.usq.edu.au` A convention is to use a host name of “www” for web servers, but this is not required.

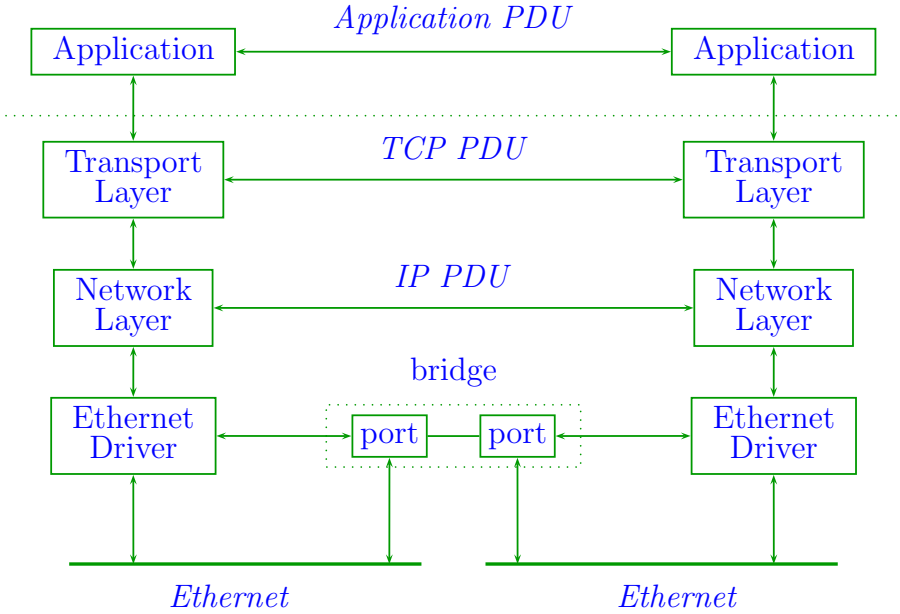
URL - Uniform Resource Locator

- **Port**, if not specified, defaults to the standard port for the protocol (http uses port 80).
- **File**, if not specified, may have a default. Example: http servers may default to `index.html` or `default.html` depending on their configuration.

Protocol Stack

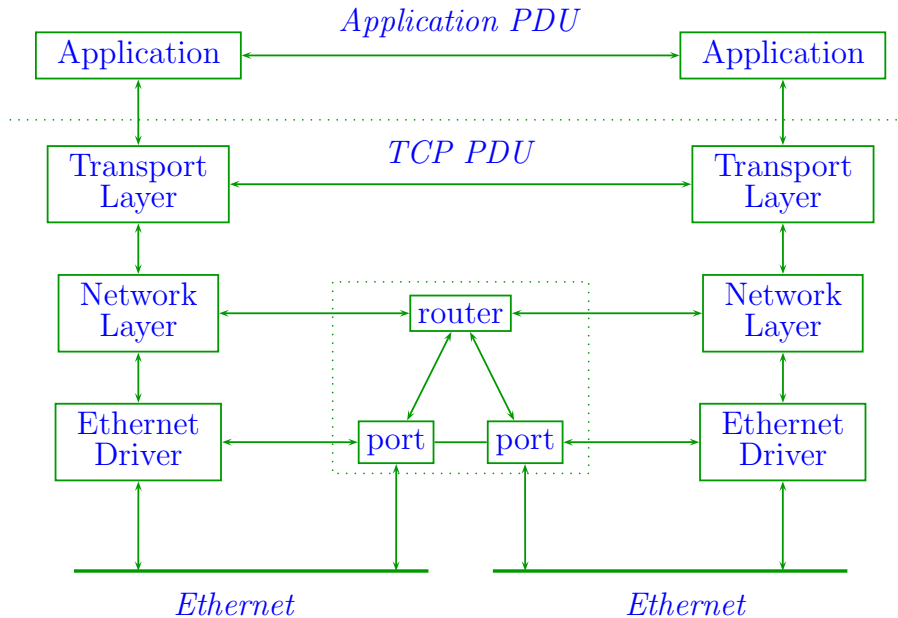


Bridge



Connection between Ethernets at the *Ethernet* or *link* layer.

Router



Connection between Ethernets at the *IP* or *network* layer.

Ethernet Addresses

- 6 byte (48 bit)
- Usually written with bytes in Hex, with '-' or ':' to separate bytes.
- Example: PCLAB1 is

00-AA-00-5B-20-3A

or

00:AA:00:5B:20:3A

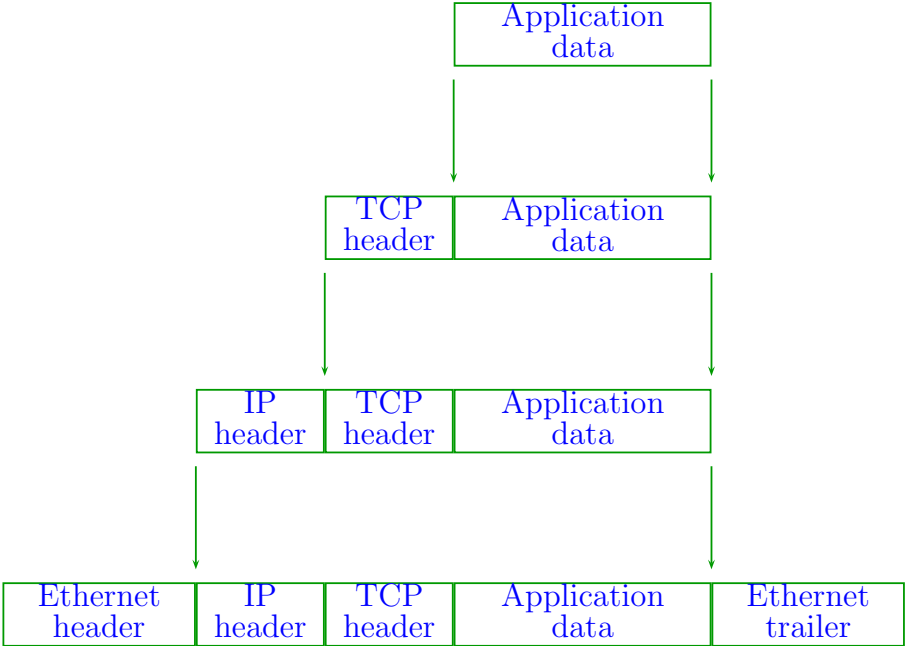
Ethernet Addresses

- Unique throughout the world (IEEE allocates numbers)
- 3-byte prefix is vendor code
 - 00:AA:00 = Intel
 - 00:00:C0 = Western Digital
 - 00:00:0C = Cisco
 - 02:60:8C = 3Com
 - 08:00:69 = Silicon Graphics
- PCLAB1 is thus an Intel Adapter
- Longer list at <http://www.cavebear.com/CaveBear/Ethernet/index.html>

IP Addresses

- Usually called “IP Numbers”
- 4 byte (32 bit)
- Usually written with bytes in decimal with ‘.’ to separate bytes.
- Example: PCLAB1 is 139.86.64.210
- Unique throughout the world (Internet authority allocates blocks of numbers)
- Divided into Classes, Networks and Hosts (see later)

TCP/IP Encapsulation – Ethernet Link



Ethernet Header

Protocol Data Units are called *packets*
Encapsulation of IP segments specified in standard RFC 894



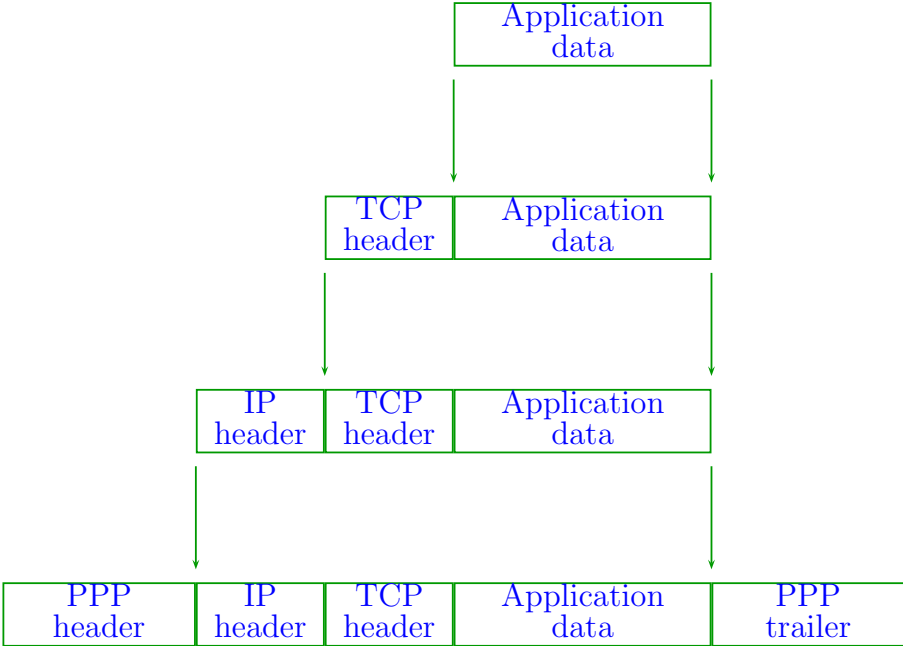
Ethernet Header

- Destination, Source = 6-byte addresses
- Type is:
 - 0800H if payload is an IP datagram
 - 0806H if an ARP request/reply
 - + *some others*
 - *more on these two later...*
- *Trailer* is a CRC (Cyclic Redundancy Check) (also called FCS or Frame Check Sequence) for error checking.
- IP encapsulation specified in RFC894

Point-to Point Protocol (PPP)

- PPP = Point-to Point Protocol. RFC1661 & RFC1662
- Supersedes older “slip” protocol.
- Used over dial-up serial lines (typically \ll 56k)
- Note: IP address may be assigned dynamically (not statically)
- IP Datagrams multiplexed onto PPP link.
- Has *link-control* and *IP-encapsulation* functions.
- Most TCP/IP transmission errors occur on slow/unreliable serial link, so PPP includes error detection.

TCP/IP Encapsulation – Serial Link



Point-to-Point Frame

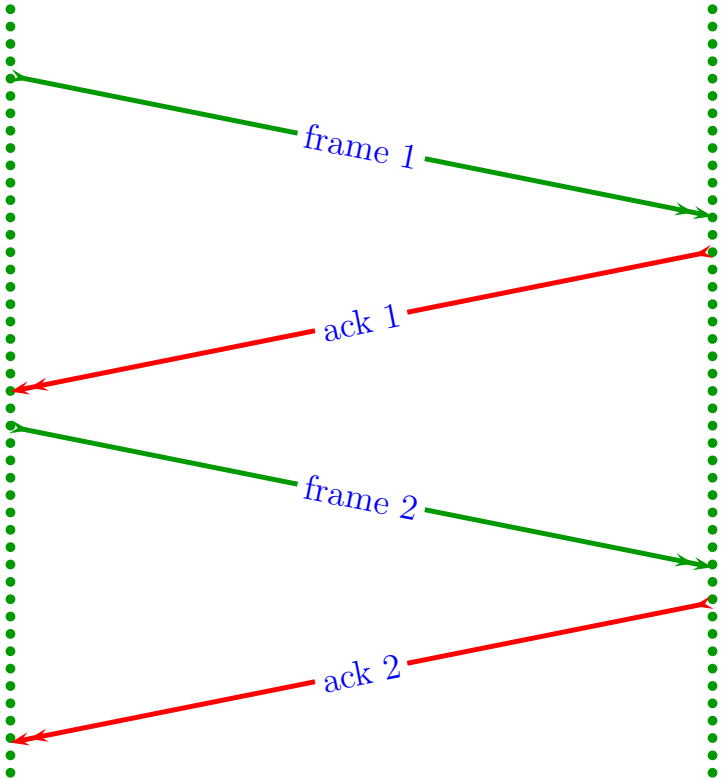
- Protocol Data Units are called *frames*.
- Encapsulation of IP segments specified in RFC1661 & RFC1662.
- Protocol for link negotiation or payload.
- Each end maintains *state* of link.
- Additional facilities for address assignment, authentication.



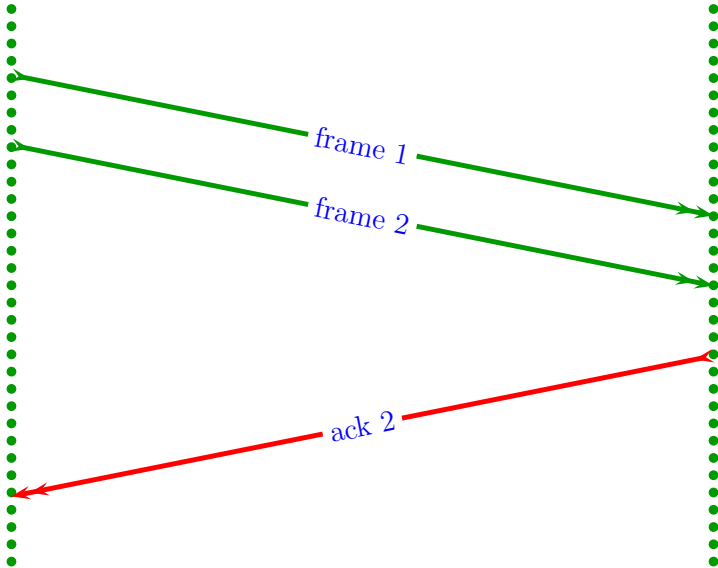
TCP Protocol

- TCP = Transmission Control Protocol
- Protocol Data Units are called *segments*
- It is *reliable* guaranteed to be error-free.
- Uses retransmissions to re-send corrupted segments.
- Timeout system, based on RTT (Round-Trip Time)
- Provides the application with a reliable *byte-stream*, akin to reading a file.

Send-and-Wait

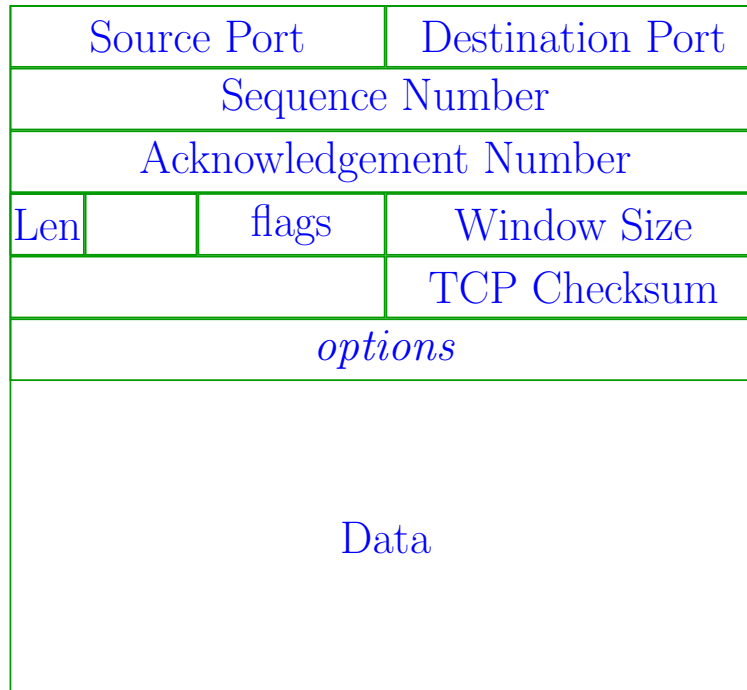


Windowed Acknowledgements



TCP Segment

32 bits

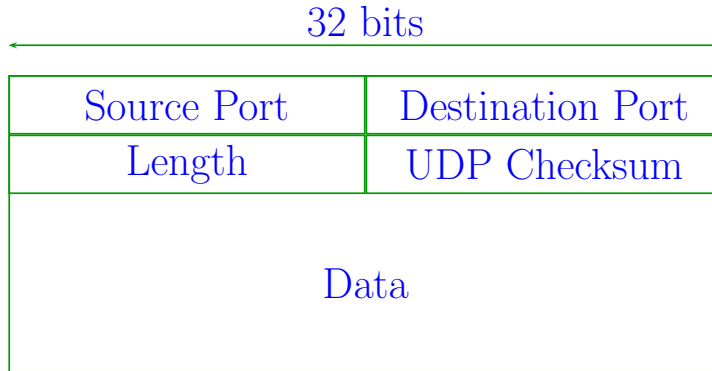


some header fields not shown

UDP Protocol

- UDP = User Datagram Protocol
- Protocol Data Units are called *datagrams*
- It is *unreliable* – *not* guaranteed to be error-free.
- Best-effort delivery.
- Fast, simple.
- Provides the application with datagrams only.
- Datagrams may be corrupted or out-of-order.
- Used for real-time applications: Voice over IP (VoIP) and Video over IP (VIP), where retransmission serves no purpose.

UDP Datagram



checksum checks header only, not data

TCP vs UDP

- TCP is *connection-oriented*
- UDP is *connectionless*
- Analogy:
 - telephone conversation is “connection-oriented” (TCP)
 - letter through post office is “connectionless” (UDP)

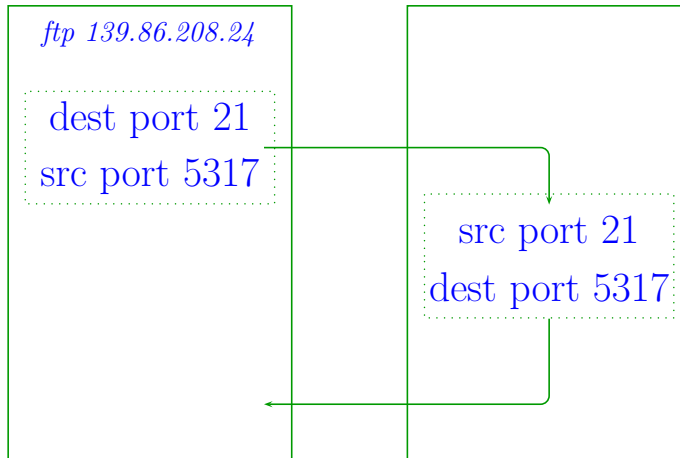
Protocol Ports

- Servers use “well-known” reserved ports 1 to 1023
- Example: time server on port 13, ftp is port 21, telnet is port 23, http is port 80
- Servers listen for connection requests on these ports
- Client ports are dynamically-allocated ‘ephemeral’ (short-lived)
- Try `netstat -a -n` while a web browser connects to `www.whitehouse.gov`
look at ports used.

Ports & Sockets

Combination of Port Number and IP Number define a *socket*, which defines a unique data flow.

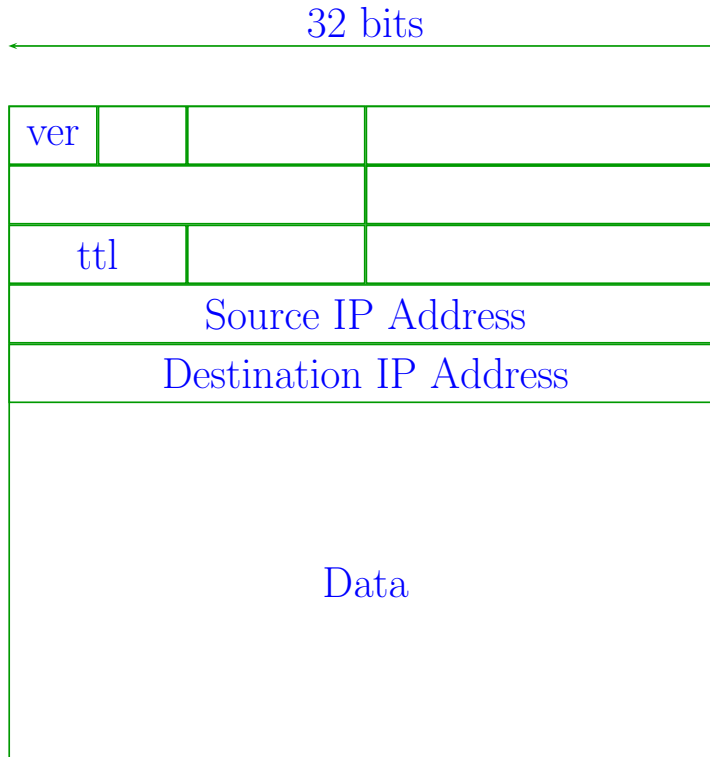
client 139.86.64.93 server 139.86.208.24



IP Datagrams

- Protocol Data Units are called *datagrams*
- IP Header is 20 bytes long

IP Datagram



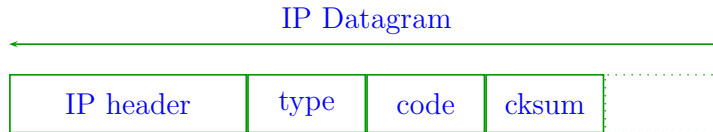
IP Header

C definition:

```
typedef struct
{
    byte    HeaderLength:4;
    byte    Version:4;
    byte    TypeOfService;
    word    TotalLength;
    word    Identification;
    word    Fragment;
    byte    TimeToLive;
    byte    Protocol;
    word    HeaderChecksum;
    ipaddr  SourceIPAddress;
    ipaddr  DestIPAddress;
} IP_HEADER;
```

ICMP

- ICMP = Internet Control Message Protocol
- Various control functions.
- Example: *ping* sends ICMP ECHO-REQUEST messages.



Ping/ICMP Example

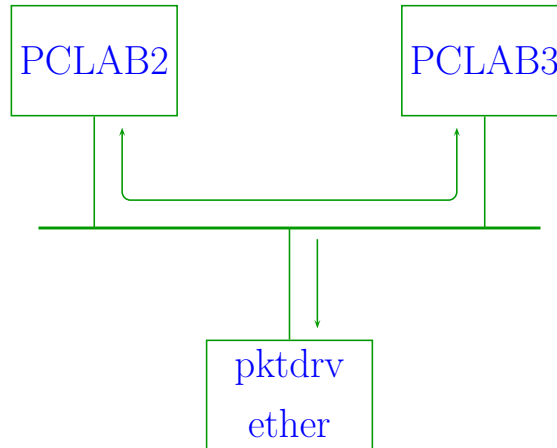
```
ping 139.86.64.212 -t
```

```
139.86.64.211
```

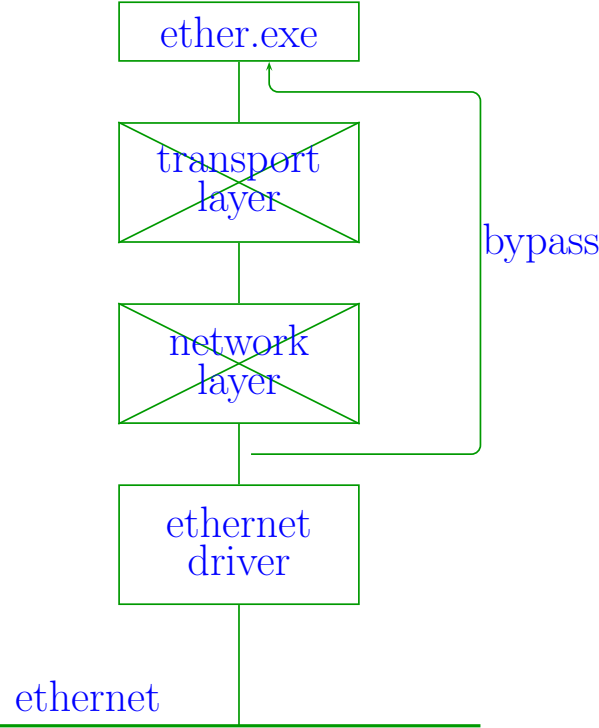
```
139.86.64.212
```

```
00-AA-00-5B-21-3A
```

```
00-AA-00-61-6F-1E
```



Packet Analyser



ICMP Echo Request

00 aa 00 61 6f 1e	Ethernet Destination
00 aa 00 5b 21 3a	Ethernet Source
08 00	Frame Type = IP (0800 Hex)
☐ 5 00 00 3c	IP Version 4 (first 4 bits)
75 02 00 00	
20 01	
8d 6b	IP Header Checksum
139 86 64 211	Source IP (shown in decimal)
139 86 64 212	Destination IP (shown in decimal)
08	Type 08 = ICMP Echo Request
00	Code 00 = ICMP Echo Request

Note network integer order is “big-endian”

ICMP Echo Reply

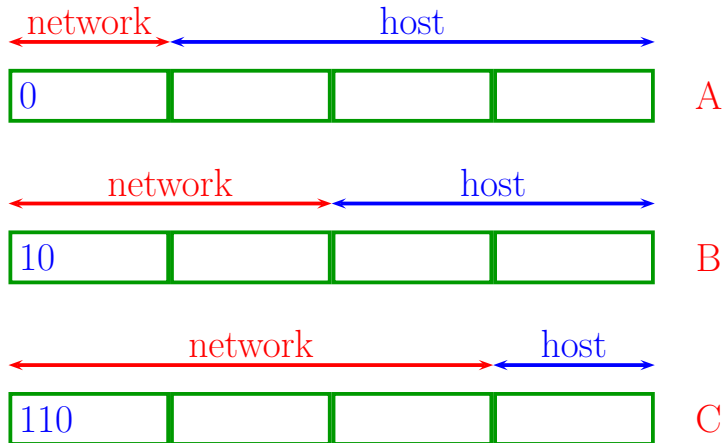
00 aa 00 5b 21 3a Ethernet Destination
00 aa 00 61 6f 1e Ethernet Source
08 00 Frame Type = IP (0800 Hex)

45 00 00 3c
81 02 00 00
20 01
81 6b
139.86.64.212 Source IP (shown in decimal)
139.86.64.211 Destination IP (shown in decimal)

00 Type 00 = ICMP Echo Reply
00 Code 00 = ICMP Echo Request

IP Addressing

- 32-bit, written as “dotted decimal”
- Three main types (classes):



IP Addressing

- Class A: < 128
- Class B: $128 \rightarrow 191$
- Class C: $192 \rightarrow 223$
- > 223 : reserved
- All host bits set to 0 = network itself
- See RFC 990

Classless IP Addressing

- Running out of IP addresses
- “traditional” class A/B/C addressing inefficient (only a few percent of addresses actually used)
- eg, a single Class A block could support more than 16 million hosts.
- Routing tables getting very large (10’s of 1000’s of entries)
- CIDR = Classless Inter-Domain Routing (“cider”)
- In CIDR, 206.13.01.48/25 means 25 bits for network
- Routing protocols must support CIDR
- Route aggregation
- RFC’s: 1517, 1518, 1519, 1520

Network Address Translation

- NAT = Network Address Translation
- Assign “local” IP address temporarily (eg dial-up or roaming wireless)
- Uses non-routable IP addresses for private network; not routed to the outside world.
- From RFC1918, Internet Assigned Numbers Authority (IANA) reserve address blocks in the IP address space for private internets:
 - 10.0.0.0 10.255.255.255 (10/8 prefix)*
 - 172.16.0.0 172.31.255.255 (172.16/12 prefix)*
 - 192.168.0.0 192.168.255.255 (192.168/16 prefix)*
- IP address no longer static: cannot advertise as a server
- Provider’s NAT maps private addresses to a single outside address.
- Must keep track of connections to do this

RFC3927 also specifies a “zero-config” address: *169.254/16 prefix*.

Mostly used for wireless – see <http://www.zeroconf.org/>

Subnetting

- Defines a “network-within-a-network” or “subnet”
- A 32-bit mask for local modification of addresses.
- Moves dividing line between network and host bits.
- Allows more networks each with fewer hosts.
- ‘1’ in subnet mask → network portion
- ‘0’ in subnet mask → host portion
- example: 255.255.248.0 =
1111 1111 . 1111 1111 . 1111 1000 . 0000 0000

Subnet Calculator

Exercise: subnet application (download via course home page)

The screenshot shows a window titled "subnet 1.0" with two main sections: "Classful Address" and "Subnetted Address".

Classful Address:

- IP Address: 192.168. 3. 12
- Address Type: Class C, 110xxxxx (Private 192.168/16)
- Address bits: 1110000010101000000000100001100 (The first three bits '111' are circled in green and labeled "Address class bits". The remaining bits are circled in blue and labeled "Host address".)
- Address mask: 255.255.255.0
- Network: 192.168.3.0
- Broadcast: 192.168.3.255
- Range: 192.168.3.1 - 192.168.3.254 (254)

Subnetted Address:

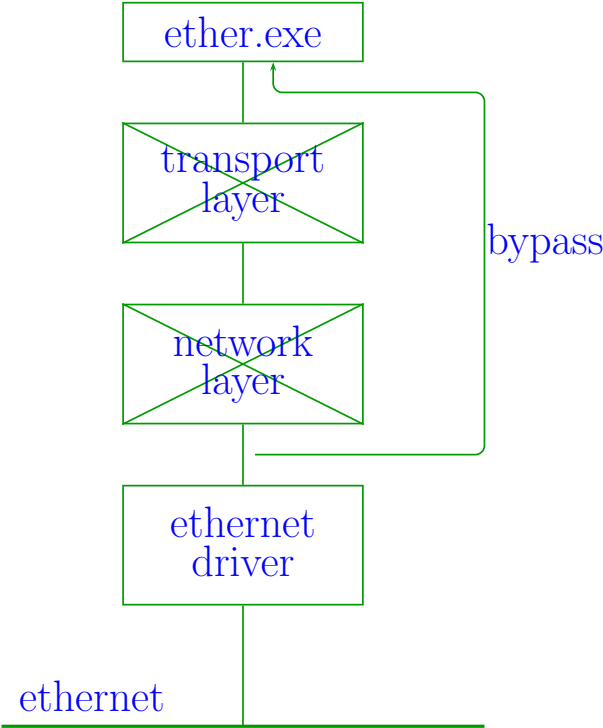
- Mask length: 27
- Mask bits: 1110000010101000000000100001100 (The first three bits '111' are circled in green and labeled "Address class bits". The next five bits '00000' are circled in blue and labeled "Subnetwork address". The remaining bits '00101100' are circled in blue and labeled "Host address".)
- Mask: 255.255.255.224
- Subnetwork: 192.168.3.0 (8 subnets)
- Broadcast: 192.168.3.31
- Range: 192.168.3.1 - 192.168.3.30 (30)

Buttons at the bottom: "calculator" and "exit".

ARP Protocol

- ARP → Address Resolution Protocol
- When a host on an Ethernet knows the destination IP address but not the Ethernet address.
- Broadcasts an ARP packet.

ARP Protocol



ARP Frame

dest ethernet	src ethernet	type
---------------	--------------	------

misc

sender ethernet	sender IP
-----------------	-----------

dest ethernet	dest IP
---------------	---------

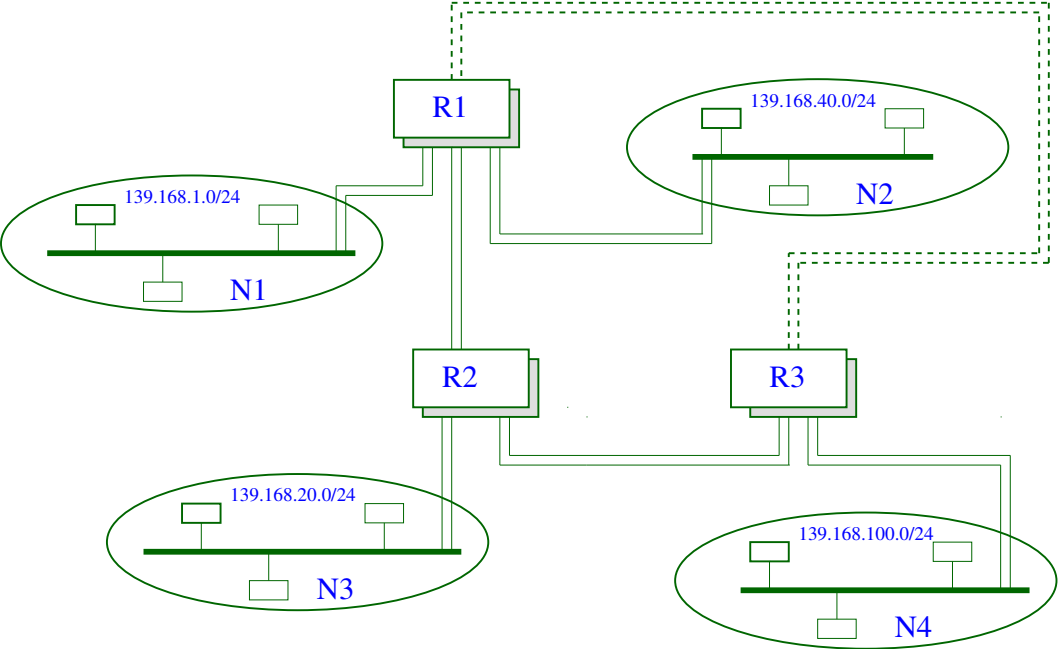
ARP Request

ff ff ff ff ff ff	Ethernet destination = Broadcast (all 1's)
00 aa 00 5b 21 3a	Ethernet source
08 06	Frame type = 0806 Hex (ARP)
00 01	Hardware Type = 1 for Ethernet
08 00	Protocol Type = 0800 Hex for IP
06	Hardware Address Size
04	Mapped (IP) Address Size
00 01	1 = ARP request (2 would be ARP reply)
00 aa 00 5b 21 3a	Sender Ethernet Address
139 86 64 211	Sender IP Address (shown in decimal)
00 00 00 00 00 00	Target Ethernet Address (unknown)
139 86 64 212	Target IP Address (shown in decimal)

Routing

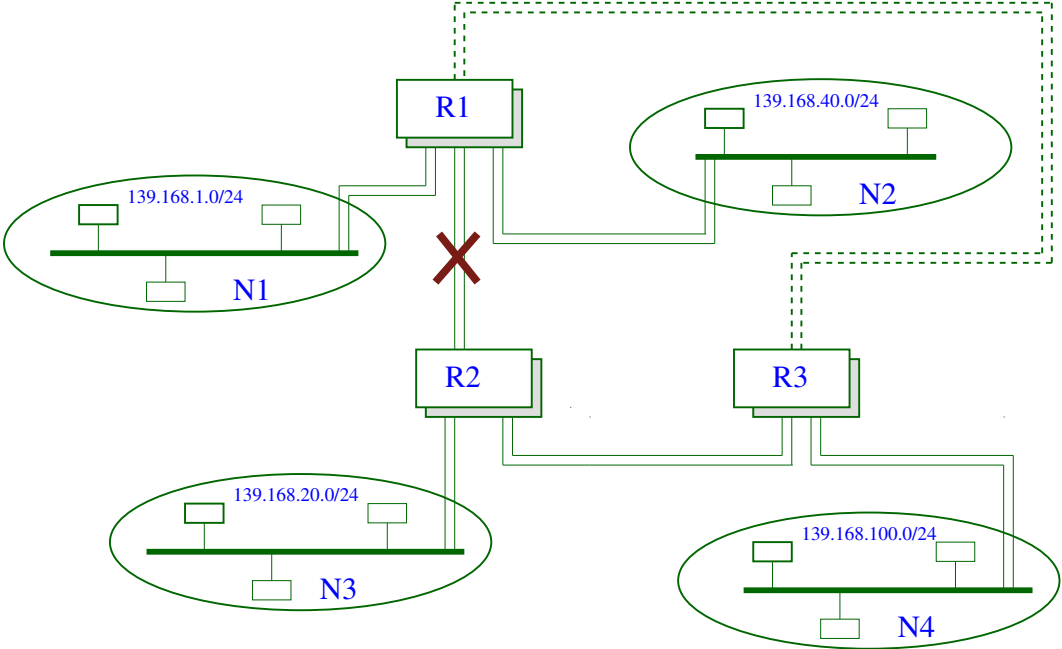
- Basically “next-hop” routing
- Each router forwards IP datagram so as to get one hop closer to ultimate destination.
- Directly-connected networks (Point to Point, LAN) – no need for routing as such.
- Static routes (fixed) or dynamic routing protocol.
- Multiple links for redundancy.

Routing



Note multiple-link path N1 → N3

Routing



Route around broken link

Routing

- Would like automatic configuration and automatic failover. Also network topology may change over time.
- Distributed computation of routes: each router has “next-hop” address table.
- Routing metric: simple such as number of hops; complex such as delay, queue length, instantaneous load, bandwidth, monetary cost...
- Router changes the Physical Address when forwarding; IP address unchanged
- Routing protocols: RIP (RFC 1058, 1723), OSPF, BGP (link-state and distance vector protocols)
- Routers exchange information periodically (“advertise”)
- RIP uses IP datagrams, port 520, 30 second update
- Possible problems include incorrect routes and routing loops

Network Configuration

- Network configurations (IP number, Ethernet number)
- Routing, Host reachability
- Statistics

The following are representative examples only.

Students are encouraged to try the commands for themselves where possible.

Host Configuration

ifconfig: Interface Configuration *Unix only*

```
helios[28] /usr/sbin/ifconfig lan0
```

```
lan0: flags=863<UP,BROADCAST,NOTRAILERS,  
RUNNING,MULTICAST>
```

```
inet 139.86.208.24 netmask ffff800 broadcast 139.86.215.255
```

Host Configuration

ipconfig: IP Configuration *Windows only*

```
d:\ > ipconfig /all
```

Windows IP Configuration

Host Name : estaff33.eng.usq.edu.au

DNS Servers : 139.86.128.2
139.86.64.101

Ethernet adapter Intel EtherExpress PRO:

Physical Address. : 00-AA-00-A4-1B-15

IP Address. : 139.86.64.93

Subnet Mask : 255.255.248.0

Default Gateway : 139.86.64.1

Host Reachability

ping: Check response *DOS, Unix - details may vary*

```
D:\ > ping 139.86.64.101
```

```
Pinging [139.86.64.101] with 32 bytes of data:
```

```
Reply from 139.86.64.101: bytes=32 time<10ms TTL=255
```

```
Reply from 139.86.64.101: bytes=32 time<10ms TTL=255
```

(edited slightly)

LAN Address Resolution

arp: Ethernet ⇒ IP Address *DOS, Unix*

```
D:\ > arp -a
```

```
Interface: 139.86.64.93
```

Internet Address	Physical Address	Type
139.86.64.6	00-80-5f-68-a5-d5	Dynamic

LAN Address Resolution

arp: Ethernet ⇒ IP Address *DOS, Unix*

```
D:\ > ping 139.86.64.101
```

```
Pinging [139.86.64.101] with 32 bytes of data:
```

```
Reply from 139.86.64.101: bytes=32 time<10ms TTL=255
```

```
Reply from 139.86.64.101: bytes=32 time<10ms TTL=255
```

```
D:\ arp -a
```

```
Interface: 139.86.64.93
```

Internet Address	Physical Address	Type
139.86.64.101	08-00-20-1b-0e-0c	Dynamic
139.86.64.6	00-80-5f-68-a5-d5	Dynamic

LAN/IP Status

netstat: network status & statistics

D:\ > netstat

Active Connections

Proto	Local Address	Foreign Address
TCP	estaff33.eng.usq.edu.au:1025	max:nbsession
TCP	estaff33.eng.usq.edu.au:1027	helios:telnet

LAN/IP Status

netstat: network status & statistics

helios[2] netstat -nr

Routing tables

Destination	Gateway	Flags	Refs	Use
127.0.0.1	127.0.0.1	UH	0	308
139.86.208.24	127.0.0.1	UH	0	40208
default	139.86.208.1	UG	100	1559985
139.86.208.0	139.86.208.24	U	1	35108

LAN/IP Status

netstat: network status & statistics

- U Route is Up & operational
- H A route to a specific Host (not a network)
- G Route uses a Gateway
- Refs Number of references
- Use Number of packets sent via this route

LAN/IP Status

netstat: network status & statistics

helios[66] netstat -i

Name	Ipkts	Opkts	Coll
ni0*	0	0	0
ni1*	0	0	0
lo0	1187003	1187002	0
lan0	33875093	30383785	455393

$$\text{Ethernet Collision Rate} = \frac{455393}{30383785} \times 100 \approx 1.5\%$$

Routing

traceroute: trace route to host *DOS = tracet*

From 139.86.64.93:

D:\ > tracert helios

Tracing route to helios [139.86.208.24]

over a maximum of 30 hops:

1 <10 ms <10 ms <10 ms [139.86.64.1]

2 <10 ms <10 ms <10 ms helios [139.86.208.24]

Try

traceroute rs.internic.net

Network Programming

Unix:

- Berkeley Software Distribution (BSD) “socket” library
- *Kernel* extensions to support TCP/IP
- Free \Rightarrow widespread adoption.

Network Programming

Windows:

- BSD-like socket library “winsock”
- Essentially the same Applications Programming Interface (API) with some extensions due to Windows task scheduling limitations (blocking tasks).
- File “wsock32.dll” (Dynamic Link Library) contains the interface code.

The following examples use a pseudo-code representation.

Network Programming Headers

Include Files: *Protocol constants, data structures.*

- Unix: *sys/socket.h, netinet/in.h, netdb.h* usually under */usr/include*
- Windows: *winsock.h* in `<compiler-path> \include`

Endian Order

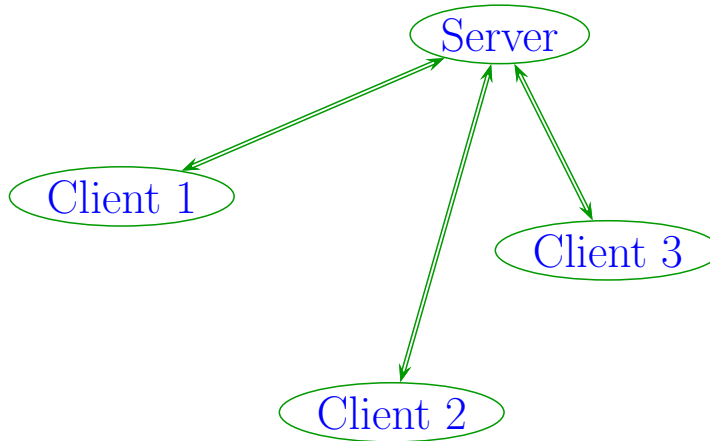
Host to network endian ordering:

- *ntohs()*, *htons()* : Network \leftrightarrow Host byte order conversion, for short integers (16 bits)

To retrieve host name:

- *gethostname()* - retrieve the name of the host on which the code is executing.

Client-Server Transaction



Server creates a child process to handle incoming requests:

`fork ()` *Create a new process context*

`exec ()` *Load new process & execute*

Unix Server

```
ListenSocket = socket()  
bind(ListenSocket, Port)  
listen(ListenSocket)
```

⋮

```
ConSocket = accept(ListenSocket)  
read(ConSocket)  
write(ConSocket)  
close(ConSocket)
```

Note: listen() and read() may block the process.

*obtain a socket
bind socket to port
wait...*

*...accept the call
read data
reply
release socket*

Unix Client

```
ServerAddr = inet_addr()
```

server's IP given

or

```
ServerAddr = gethostbyname()
```

server's hostname given

```
DataSocket= socket()
```

get a socket

```
connect(DataSocket, ServerAddr, Port)
```

connect...

⋮

```
write(DataSocket)
```

write some data...

```
read(DataSocket)
```

...get a response

```
close(DataSocket)
```

release socket

Note: connect() and read() may block the process.

Winsock Server

```
LoadLibrary(wsock32.dll)
WSAStartup()
ListenSocket = socket()
ioctlsocket(ListenSocket, NONBLOCKING)
bind(ListenSocket, Port)
WSAAsyncSelect(ListenSocket, CON_READY)
listen(ListenSocket)
CON_READY:
ConSocket = accept(ListenSocket)
ioctlsocket(ConSocket, NONBLOCKING)
WSAAsyncSelect(ConSocket, DATA_READY)
DATA_READY:
recv(ConSocket)
send(ConSocket)
WSAAsyncSelect(0)
closesocket(ConSocket)
```

load DLL
initialise winsock
obtain a socket
non-blocking mode
socket bound to port
connect message
wait...
...call comes
accept the call
non-blocking mode
wait for data message...
...data ready
read data
reply
stop messages
release socket

Winsock Client

```
LoadLibrary(wsock32.dll)
WSAStartup()
ServerAddr = inet_addr()
```

or

```
ServerAddr = gethostbyname()
DataSocket= socket()
connect(DataSocket, ServerAddr, Port)
ioctlsocket(DataSocket, NONBLOCKING)
send(DataSocket)
WSAAsyncSelect(DataSocket, DATA_READY)
DATA_READY:
recv(DataSocket)
WSAAsyncSelect(0)
closesocket(DataSocket)
WSAxxx are Windows socket asynchronous extensions.
```

*load DLL
initialise winsock
server's IP given*

*server's hostname given
get a socket
connect
non-blocking mode
send message
data received message*

*read data
stop messages
release socket*

HTTP Connection

```
telnet 139.86.64.226 80
```

```
Trying 139.86.64.226...
```

```
Connected to 139.86.64.226.
```

```
Escape character is '^]'
```

```
GET /MyDoc.html HTTP/1.0
```

```
HTTP/1.0 200 Document Follows
```

```
Content-Length: 642
```

```
Content-Type: text/html
```

```
Date: Wed, 12 Aug 1998 06:41:26 GMT
```

```
Last-Modified: Wed, 12 Aug 1998 06:41:02 GMT
```

```
Server: OmniHTTPd/1.01 (Win32; i386)
```

```
<!-- DOCTYPE HTML -->
```

```
<!-- MyDoc.html -->
```

Remainder of HTML document follows...

Module Summary – Important Points

1. Must understand “protocol stack” concept
2. IP protocol – need for; practical details
3. IP related protocols (TCP, UDP)
4. Application protocols – overview
5. Interaction of wide-area and local-area networking
6. Must understand addressing & subnetting
7. Concepts involved in routing
8. Concepts involved in testing and using networks