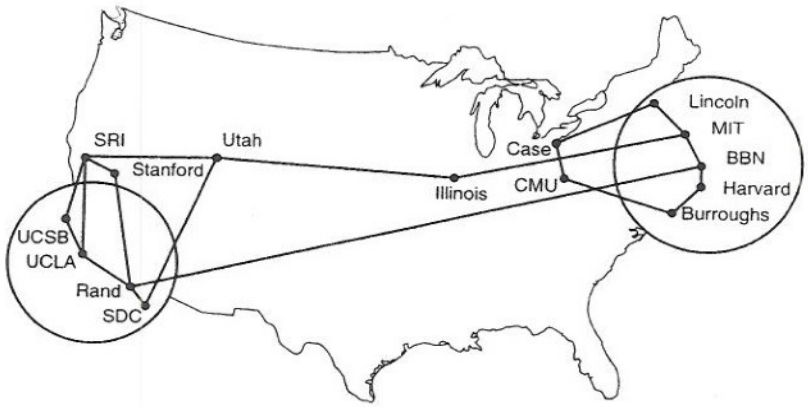


Internetworking

Outline/Goals

- Best Effort Service Model
- IP - The Protocol, RFC 791, STD 5
- Chapter 4!!!

The Problem

- Previously building a single network
  - Point to point
  - Shared media
  - Extended by bridges, hubs, and switches
- New Problem
  - Interconnecting different networks
- Issues
  - Scale … unknown size
  - Heterogeneity … service over clouds

Internetworking

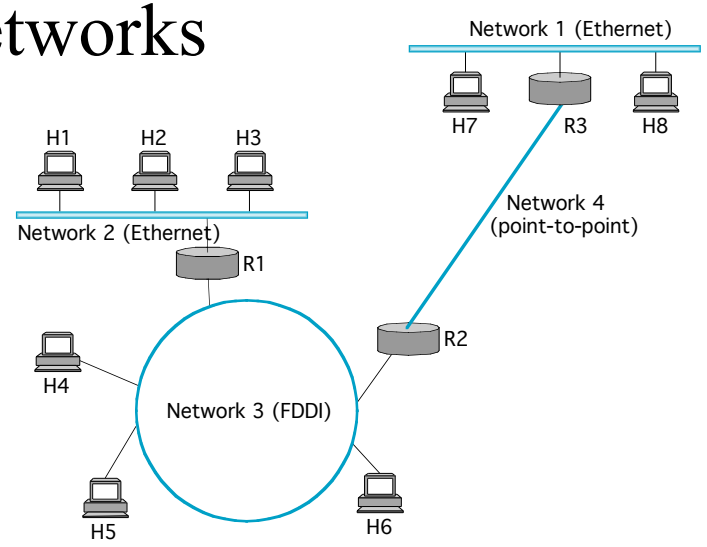
- Internet protocols define a format for a common network layer protocol and rules for transmitting this protocol over all known types of media.

- Restriction:

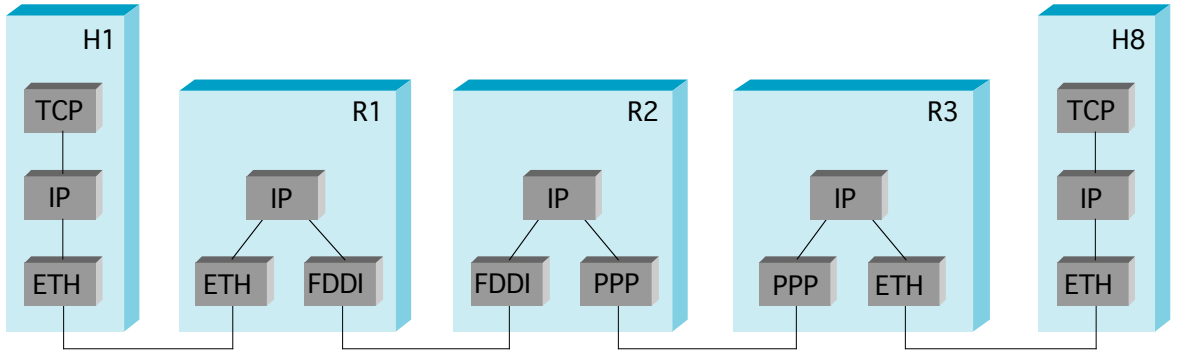
  Need to support the service offered by the least capable networking technology, e.g., dial-up modem

IP Internet

- Concatenation of Networks



- Protocol Stack



Routers forward based only on IP Addr

IP Service Model

- Global Addressing Scheme – find a node
- Packets or Datagrams – message units
- Connectionless (datagram-based) – no path determination by host
- Best-effort delivery (unreliable service) – never guarantee delivery
- Datagram format clearly specified by RFC – No Exceptions ??

IP Service Model: Why Packets?

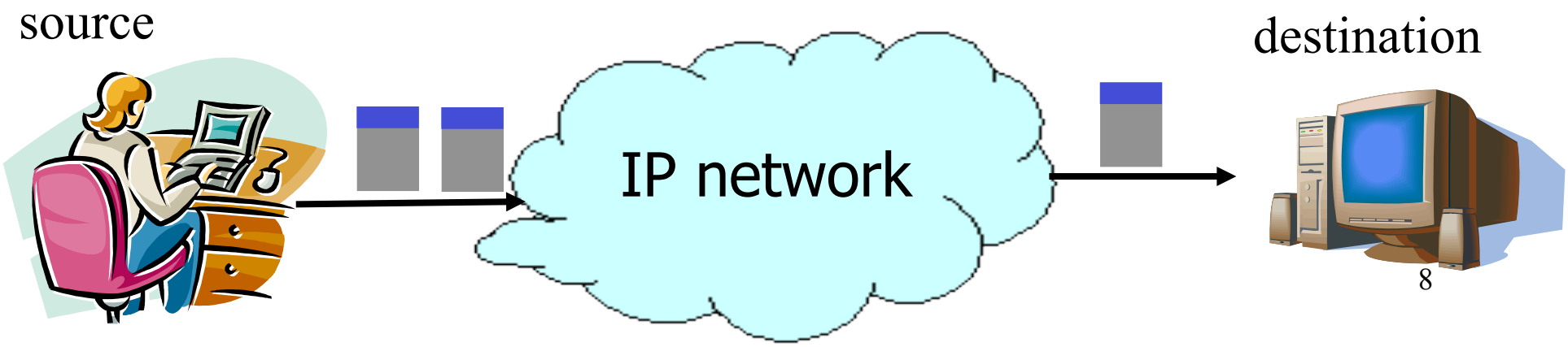
- Data traffic is bursty – Still True?? (Streaming)
  - Logging into remote machines
  - Exchanging e-mail messages
- Don't want to waste reserved bandwidth
  - No traffic exchanged during idle periods
- Better to allow multiplexing
  - Different transfers share access to same links
- Packets can be delivered by most anything
  - RFC 2549: IP over Avian Carriers (aka birds)
- … still, packet switching can be inefficient
  - Extra header bits on every packet

History: Why IP Packets?

- IP proposed in the early 1970s
  - Defense Advanced Research Project Agency (DARPA)
- Goal: connect existing networks
  - To develop an effective technique for multiplexed utilization of existing interconnected networks
  - E.g., connect packet radio networks to the ARPAnet
- Motivating applications
  - Remote login to server machines
  - Inherently bursty traffic with long silent periods
- Prior ARPAnet experience with packet switching
  - Previous DARPA project
  - Demonstrated store-and-forward packet switching

IP Service Model: Best-Effort Packet Delivery

- Packet switching
  - Divide messages into a sequence of packets
  - Headers with source and destination address
- Best-effort delivery – delivery service with NO state
  - Packets may be lost or Delayed
  - Packets may be corrupted
  - Packets may be delivered out of order

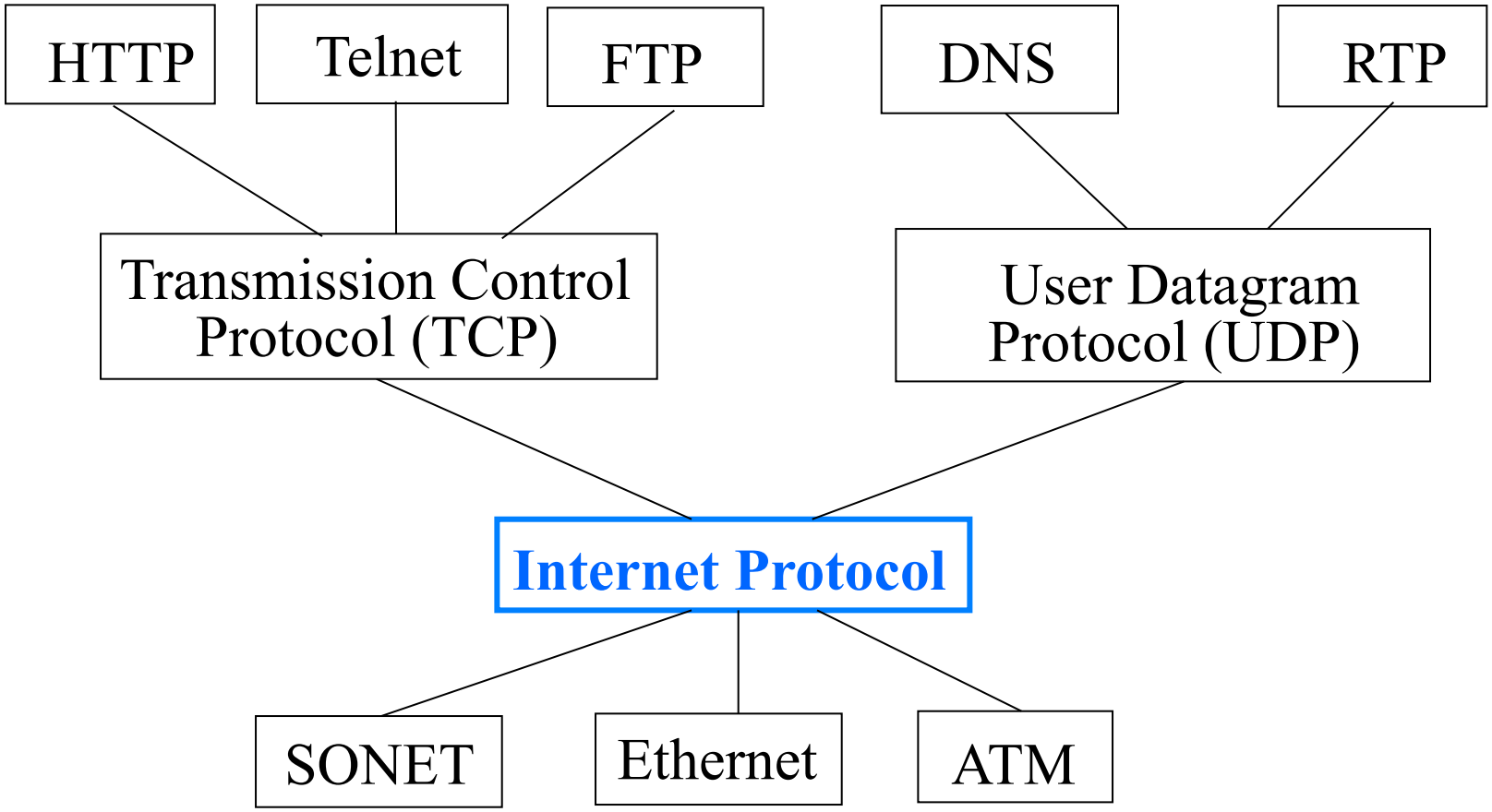


IP Service Model: Why Best-Effort?

- IP means never having to say you're sorry…
  - Don't need to reserve bandwidth and memory
  - Don't need to do error detection & correction
  - Don't need to remember from one packet to next
- Easier to survive failures
  - Transient disruptions are okay during failover

- … but, applications *do* want efficient, accurate transfer of data in order, in a timely fashion - NOT IPs problem

Layering in the IP Protocols



IP Service Model: Best-Effort is Enough ??

- No error detection or correction
  - Higher-level protocol can provide error checking
- Successive packets may not follow the same path
  - Not a problem as long as packets reach the destination
- Packets can be delivered out-of-order
  - Receiver can put packets back in order (if necessary)
- Packets may be lost or arbitrarily delayed
  - Sender can send the packets again (if desired)
- No network congestion control (beyond "drop")
  - Sender can slow down in response to loss or delay

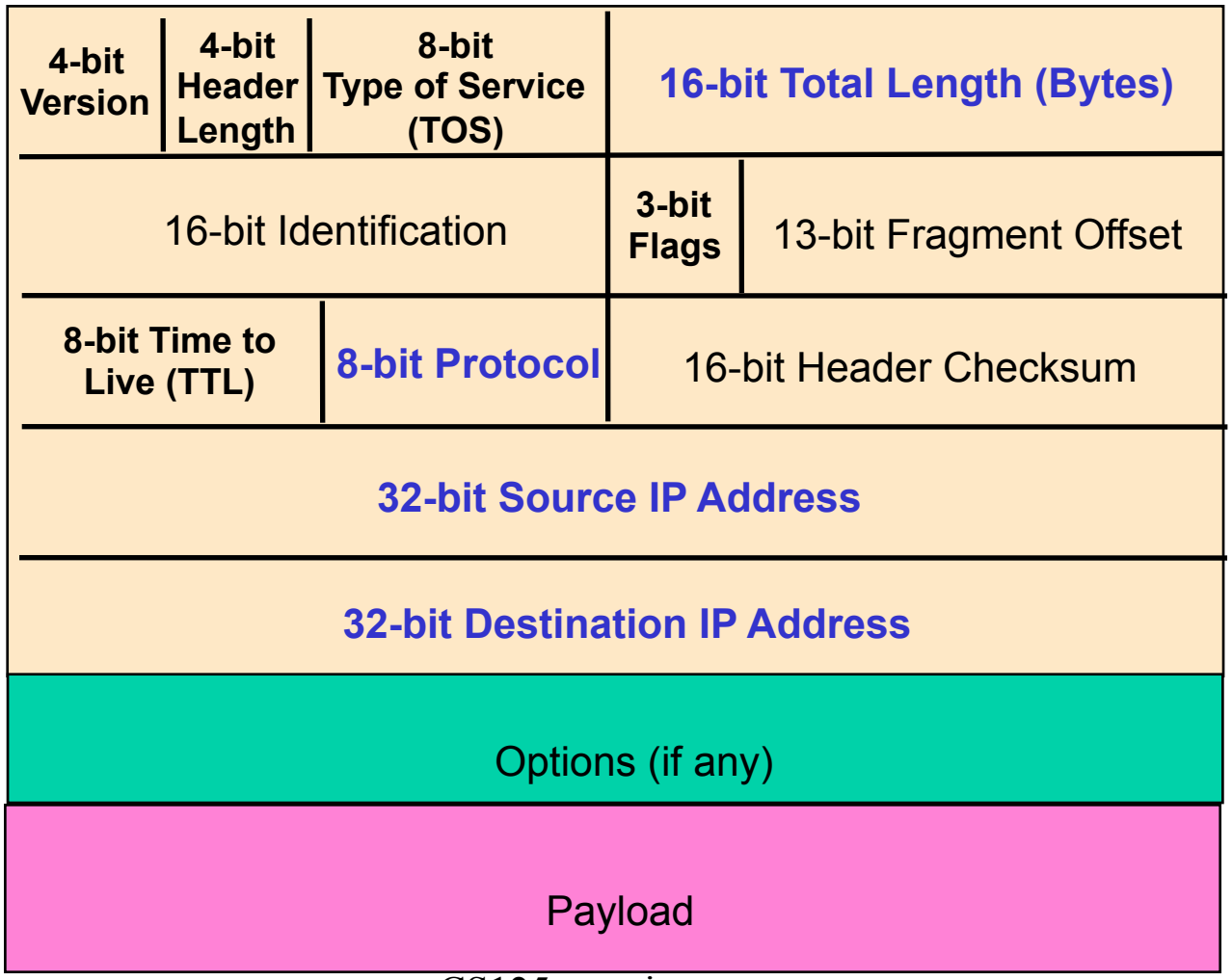
Other Main Driving Goals (In Order)

- Communication should continue despite failures
  - Survive equipment failure or physical attack
  - Traffic between two hosts continue on another path
- Support multiple types of communication services
  - Differing requirements for speed, latency, & reliability
  - Bidirectional reliable delivery vs. message service
- Accommodate a variety of networks
  - Both military and commercial facilities
  - Minimize assumptions about the underlying network

Other Driving Goals, Somewhat Met

- Permit distributed management of resources
  - Nodes managed by different institutions
  - … though this is still rather challenging
- Cost-effectiveness
  - Statistical multiplexing through packet switching
  - … though packet headers and retransmissions wasteful
- Ease of attaching new hosts
  - Standard implementations of end-host protocols
  - … though still need a fair amount of end-host software
- Accountability for use of resources
  - Monitoring functions in the nodes
  - … though this is still fairly limited and immature

IP Packet Structure



IP Packet Header Fields

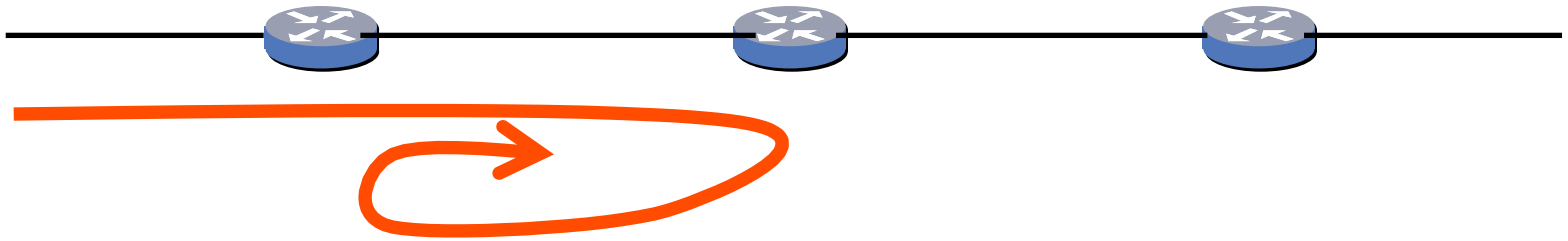
- Version number (4 bits)
  - Indicates the version of the IP protocol
  - Necessary to know what other fields to expect
  - Typically "4" (for IPv4), and sometimes "6" (for IPv6)
- Header length (4 bits)
  - Number of 32-bit words in the header
  - Typically "5" (for a 20-byte IPv4 header)
  - Can be more when "IP options" are used
- Type-of-Service (8 bits)
  - Allow packets to be treated differently based on needs
  - E.g., low delay for audio, high bandwidth for bulk transfer

IP Packet Header Fields (Continued)

- Total length (16 bits)
  - Number of bytes in the packet
  - Maximum size is 63,535 bytes (2^{16} -1)
  - … though underlying links may impose harder limits
- Fragmentation information (32 bits)
  - Packet identifier, flags, and fragment offset
  - Supports dividing a large IP packet into fragments
  - … in case a link cannot handle a large IP packet
- Time-To-Live (8 bits)
  - Used to identify packets stuck in forwarding loops

More:Time-to-Live (TTL) Field

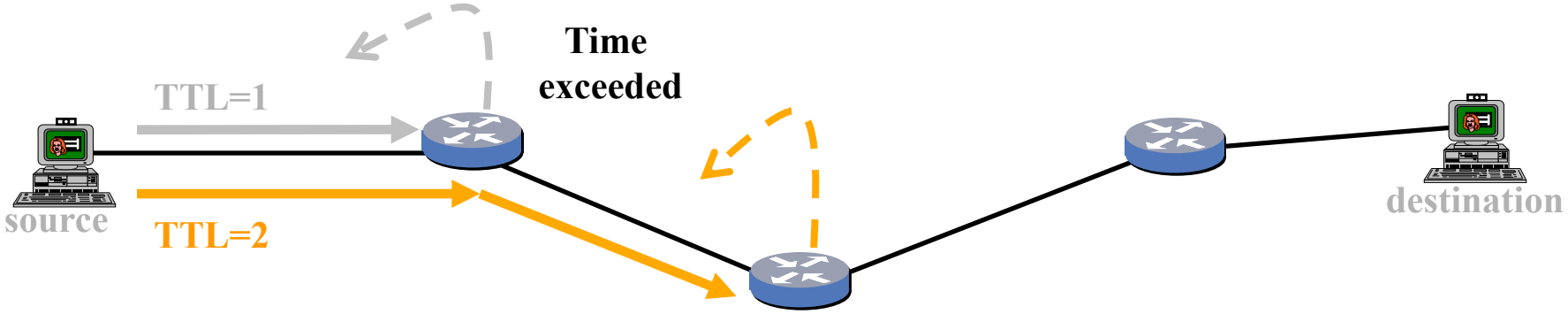
- Potential robustness problem
  - Forwarding loops can cause packets to cycle forever
  - Confusing if the packet arrives much later



- Time-to-live field in packet header
  - TTL field decremented by each router on the path
  - Packet is discarded when TTL field reaches 0…
  - …and "time exceeded" message is sent to the source
  - Used by Traceroute

Application of TTL in Traceroute

- Time-To-Live field in IP packet header
  - Source sends a packet with a TTL of *n*
  - Each router along the path decrements the TTL
  - "TTL exceeded" sent when TTL reaches *0*
- Traceroute tool exploits this TTL behavior



Send packets with TTL=1, 2, … and record source of "time exceeded" message

Ex: Traceroute: Berkeley to CNN

Hop number, IP address, DNS name

1	169.229.62.1	inr-daedalus-0.CS.Berkeley.EDU
2	169.229.59.225	soda-cr-1-1-soda-br-6-2
3	128.32.255.169	vlan242.inr-202-doecev.Berkeley.EDU
4	128.32.0.249	gigE6-0-0.inr-666-doecev.Berkeley.EDU
5	128.32.0.66	qsv-juniper--ucb-gw.calren2.net
6	209.247.159.109	POS1-0.hsipaccess1.SanJose1.Level3.net
7	*	?
8	64.159.1.46	?
9	209.247.9.170	pos8-0.hsa2.Atlanta2.Level3.net
10	66.185.138.33	pop2-atm-P0-2.atdn.net
11	*	?
12	66.185.136.17	pop1-atl-P4-0.atdn.net
13	64.236.16.52	www4.cnn.com

No response from router

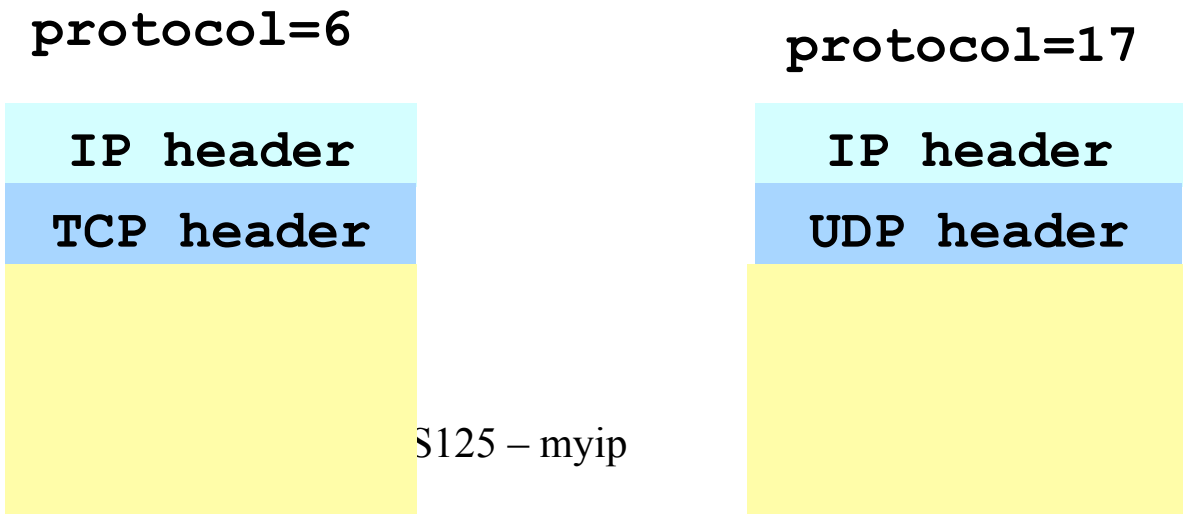
No name resolution

Try Running Traceroute Yourself

- On UNIX machine
  - Traceroute
  - E.g., “traceroute www.cs.cmu.edu”
- Common uses of traceroute
  - Discover the topology of the Internet – backbone routers
  - Debug performance and reachability problems
  - Historical transmission issues: NY to DC via Italy

IP Packet Header Fields (Continued)

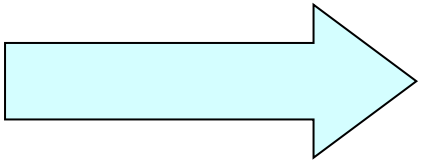
- Protocol (8 bits)
  - Identifies the higher-level protocol
    - E.g., "6" for the Transmission Control Protocol (TCP)
    - E.g., "17" for the User Datagram Protocol (UDP)
  - Important for demultiplexing at receiving host
    - Indicates what kind of header to expect next



IP Packet Header Fields (Continued)

- Checksum (16 bits)
  - Sum of all 16-bit words in the IP packet header
  - If any bits of the header are corrupted in transit
  - … the checksum won't match at receiving host
  - Receiving host discards corrupted packets
    - Sending host will retransmit the packet, if needed

```
  134
+ 212
-----
= 346
```



```
  134
+ 216
-----
= 350
```

Mismatch!

IP Packet Header (Continued)

- Two IP addresses
  - Source IP address (32 bits)
  - Destination IP address (32 bits)
- Destination address
  - Unique identifier for the receiving host
  - Allows each node to make forwarding decisions
- Source address
  - Unique identifier for the sending host
  - Recipient can decide whether to accept packet
  - Enables recipient to send a reply back to source

What if the Source Lies?

- Source address should be the sending host
  - But, who's checking, anyway?
  - You could send packets with any source you want
- Why would someone want to do this?
  - Launch a denial-of-service attack
    - Send excessive packets to the destination
    - … to overload the node, or the links leading to the node
  - Evade detection by "spoofing"
    - But, the victim could identify you by the source address
    - So, you can put someone else's source address in the packets
  - Also, an attack against the spoofed host
    - Spoofed host is wrongly blamed
    - Spoofed host may receive return traffic from the receiver
  - Ingress Filtering

IP Options

Option Class	Option Number	Length	Description
0	0	-	End of option list. Used if options do not end at end of header (also see header padding field).
0	1	-	No operation (used to align octets in a list of options).
0	2	11	Security and handling restrictions (for military applications).
0	3	var	Loose source routing. Used to route a datagram along a specified path.
0	7	var	Record route. Used to trace a route.
0	8	4	Stream Identifier. Used to carry a SATNET stream identifier (Obsolete).
0	9	var	Strict source routing. Used to route a datagram along a specified path.
2	4	var	Internet timestamp. Used to record timestamps along the route.

Figure 7.10 The eight possible IP options with their numeric class and number codes. The value *var* in the length column stands for *variable*.

IP Options

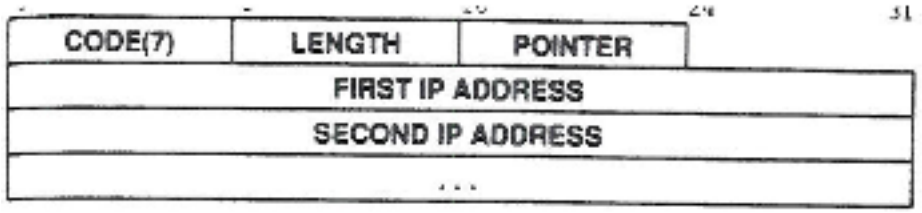


Figure 7.11 The format of the record route option in an IP datagram.

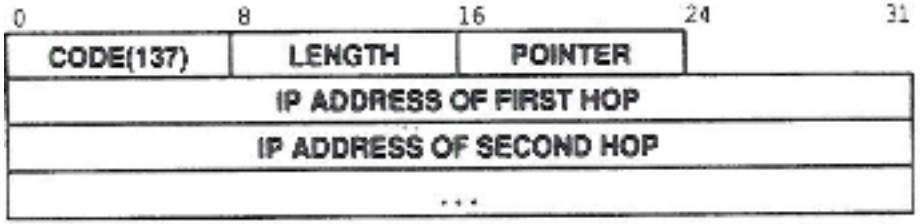
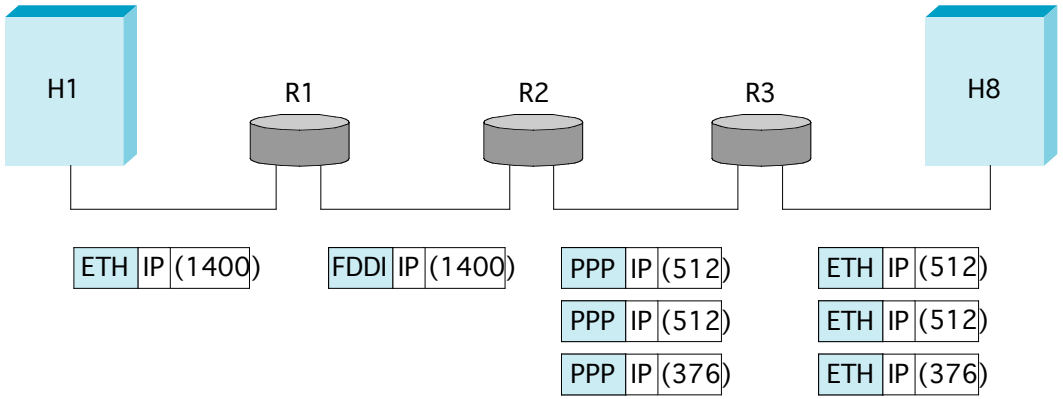


Figure 7.12 The strict source route option specifies an exact route by giving a list of IP addresses the datagram must follow.

Fragmentation and Reassembly

- Each network has some MTU
- Design decisions
  - fragment when necessary (MTU < Datagram)
  - try to avoid fragmentation at source host - does not know
  - re-fragmentation is possible
  - fragments are self-contained datagrams
  - use CS-PDU (not cells) for ATM
  - delay reassembly until destination host
  - do not recover from lost fragments
  - Router/Gateway works hard, fixes Source bad choice
  - When can you not fragment???

Example



(a)

Start of header				
Ident = x			0	Offset = 0
Rest of header				
1400 data bytes				

(b)

Start of header				
Ident = x			1	Offset = 0
Rest of header				
512 data bytes				

Start of header				
Ident = x			1	Offset = 64
Rest of header				
512 data bytes				

Start of header				
Ident = x			0	Offset = 128
Rest of header				
376 data bytes				

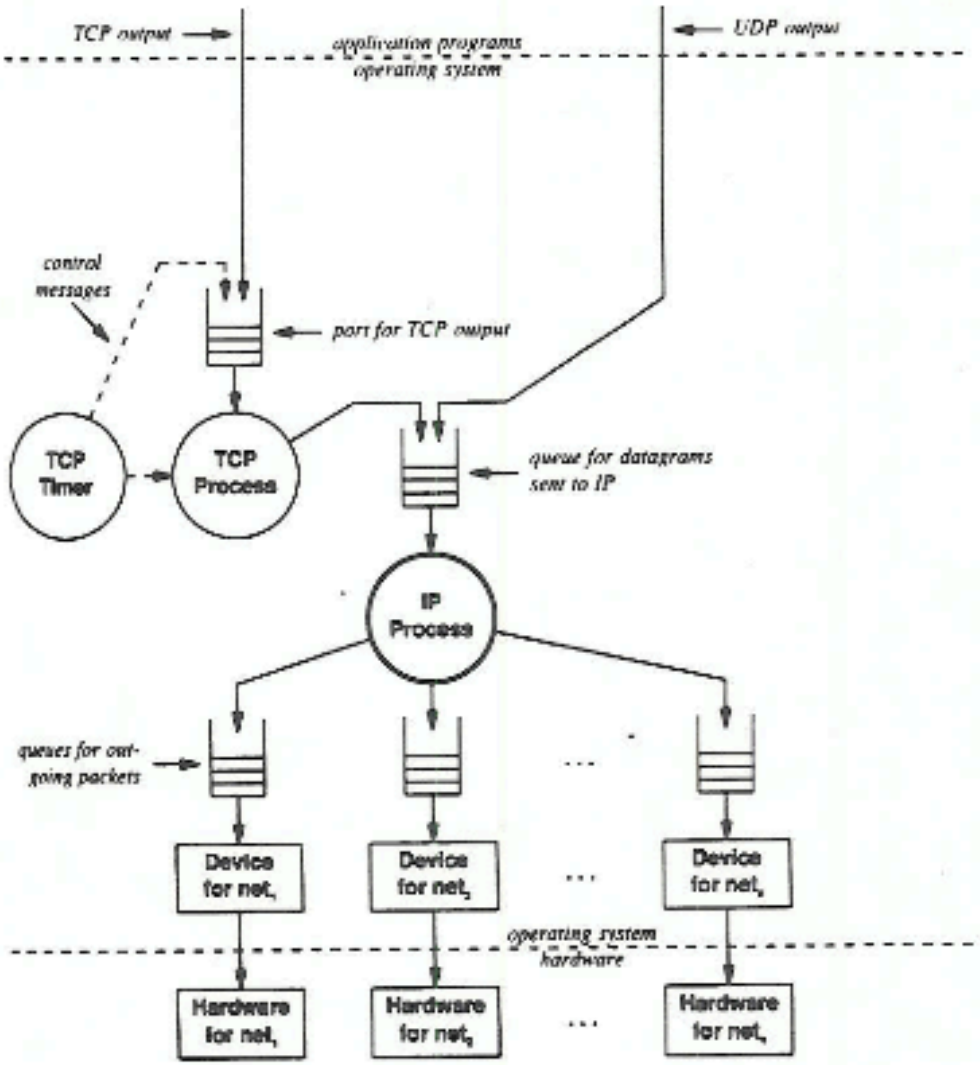
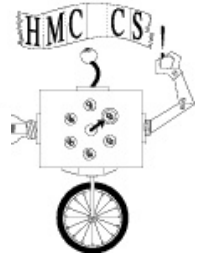


Figure 2.6 Output process structure showing the path of data between an application program and the network hardware. Output from the device queues is started at interrupt time. IP is a central part of the design – the software for input and output both share a single IP process.




```
/* Internet Protocol (IP)  Constants and Datagram Format                */

#define IP_ALEN 4                   /* IP address length in bytes (octets)  */
typedef char IPaddr[IP_ALEN];       /*  internet address                    */

#define IP_CLASSA(x)     ((x[0] & 0x80) == 0x00) /* IP Class A address      */
#define IP_CLASSB(x)     ((x[0] & 0xc0) == 0x80) /* IP Class B address      */
#define IP_CLASSC(x)     ((x[0] & 0xd0) == 0xc0) /* IP Class C address      */
#define IP_CLASSD(x)     ((x[0] & 0xf0) == 0xd0) /* IP Class D address      */
#define IP_CLASSE(x)     ((x[0] & 0xf0) == 0xf0) /* IP Class E address      */

/* Some Assigned Protocol Numbers */

#define IPT_ICMP         1          /* protocol type for ICMP packets       */
#define IPT_TCP          6          /* protocol type for TCP packets        */
#define IPT_EGP          8          /* protocol type for EGP packets        */
#define IPT_UDP          17         /* protocol type for UDP packets        */

struct  ip      {
        char    ip_verlen;          /* IP version & header length (in longs)*/
        char    ip_tos;             /* type of service                      */
        short   ip_len;             /* total packet length (in octets)      */
        short   ip_id;              /* datagram id                          */
        short   ip_fragoff;         /* fragment offset (in 8-octet's)       */
        char    ip_ttl;             /* time to live, in gateway hops        */
        char    ip_proto;           /* IP protocol (see IPT_* above)        */
        short   ip_cksum;           /* header checksum                      */
        IPaddr  ip_src;             /* IP address of source                 */
        IPaddr  ip_dst;             /* IP address of destination            */
        char    ip_data[1];         /* variable length data                 */
};
```