

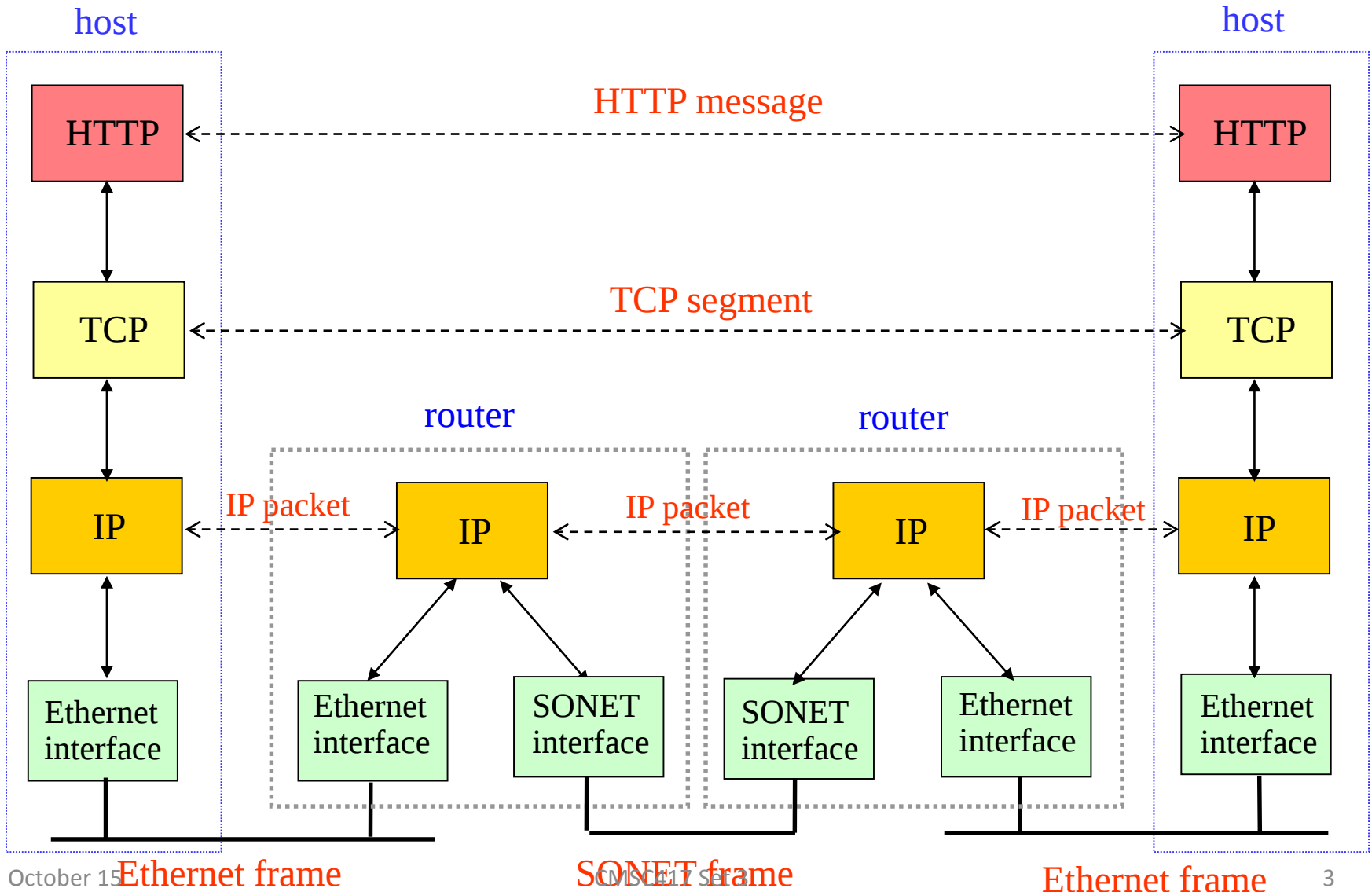
# CMSC 417

Computer Networks  
Prof. Ashok K Agrawala

© 2015 Ashok Agrawala  
Set 6

# The Network Layer

# Message, Segment, Packet, and Frame

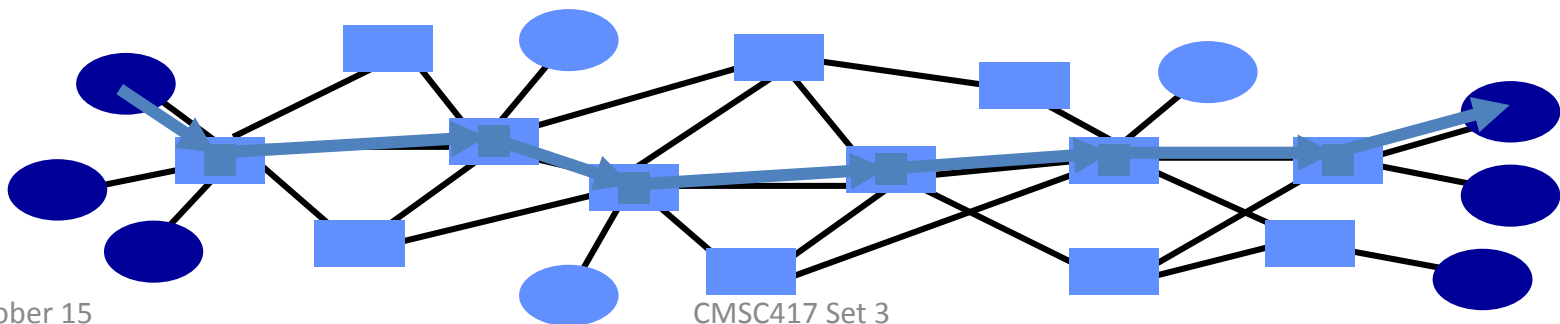


# Network Layer Design Issues

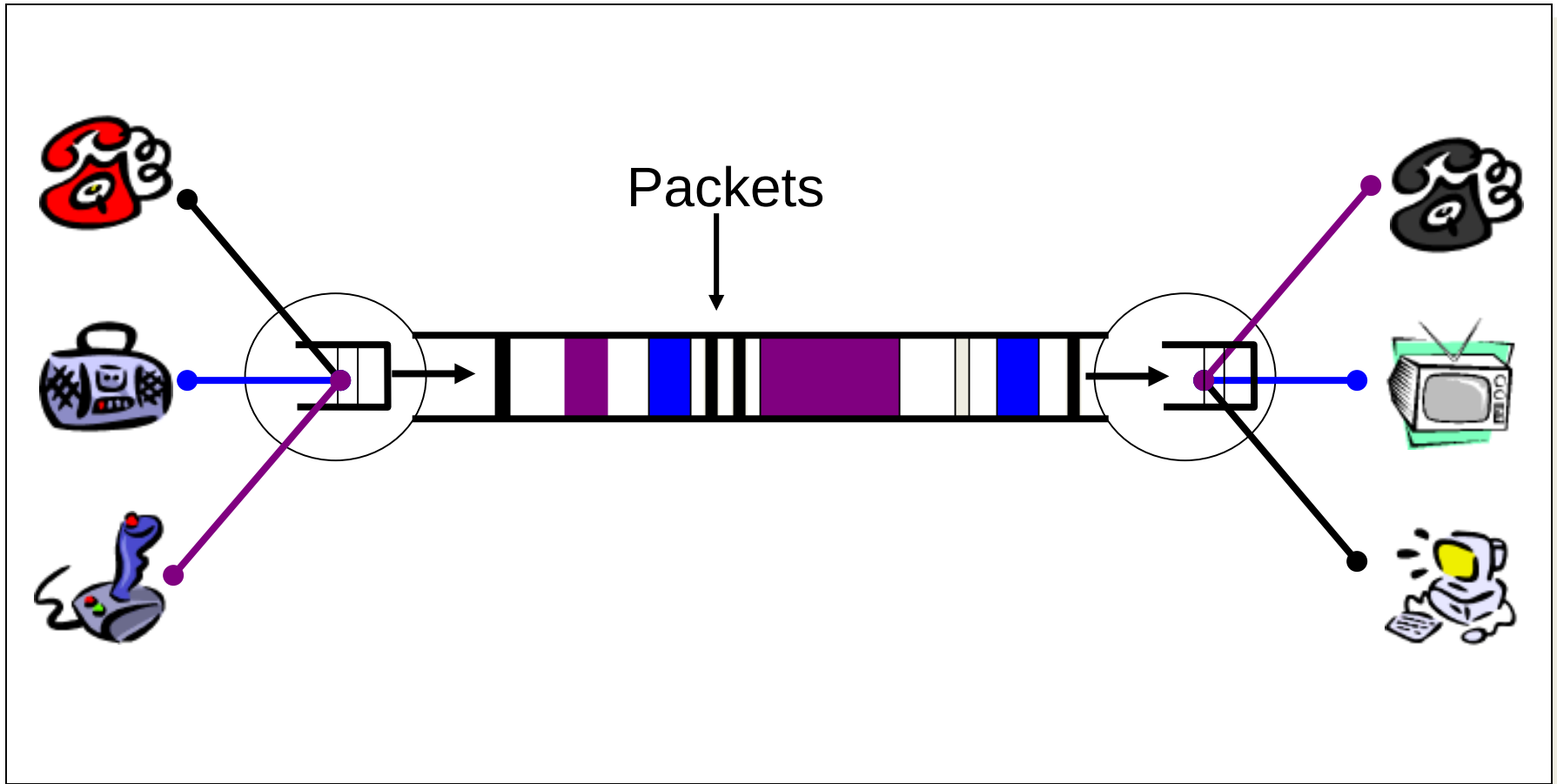
- Store-and-Forward Packet Switching
- Services Provided to the Transport Layer
- Implementation of Connectionless Service
- Implementation of Connection-Oriented Service
- Comparison of Virtual-Circuit and Datagram Subnets

# Packet Switching (e.g., Internet)

- Data traffic divided into packets
  - Each packet contains a header (with address)
- Packets travel separately through network
  - Packet forwarding based on the header
  - Network nodes may store packets temporarily
- Destination reconstructs the message

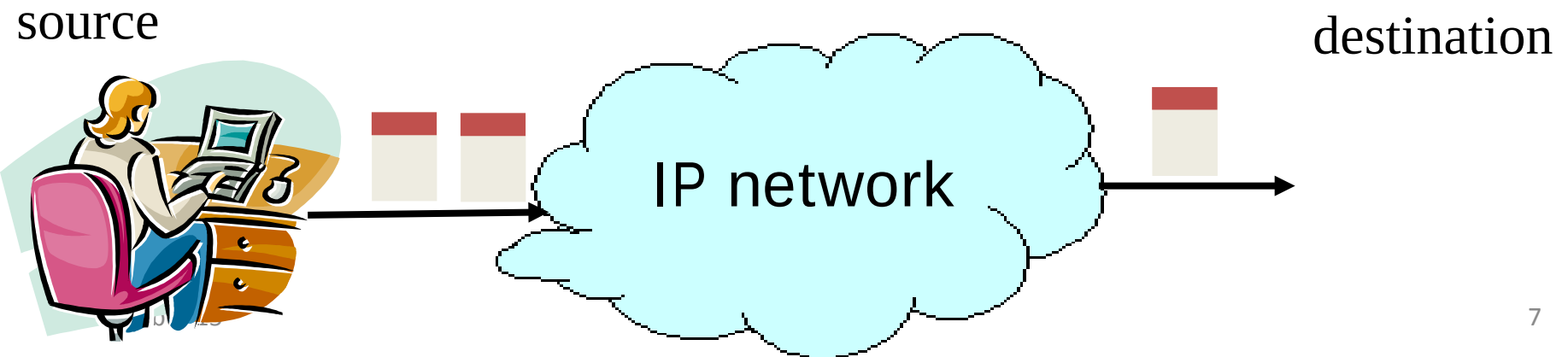
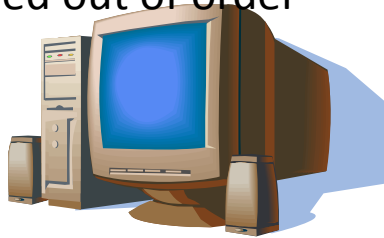


# Packet Switching: Statistical Multiplexing



# IP Service: Best-Effort Packet Delivery

- Packet switching
  - Divide messages into a sequence of packets
  - Headers with source and destination address
- Best-effort delivery
  - Packets may be lost
  - Packets may be corrupted
  - Packets may be delivered out of order



# IP Service Model: Why Packets?

- Data traffic is bursty
  - Logging in to 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

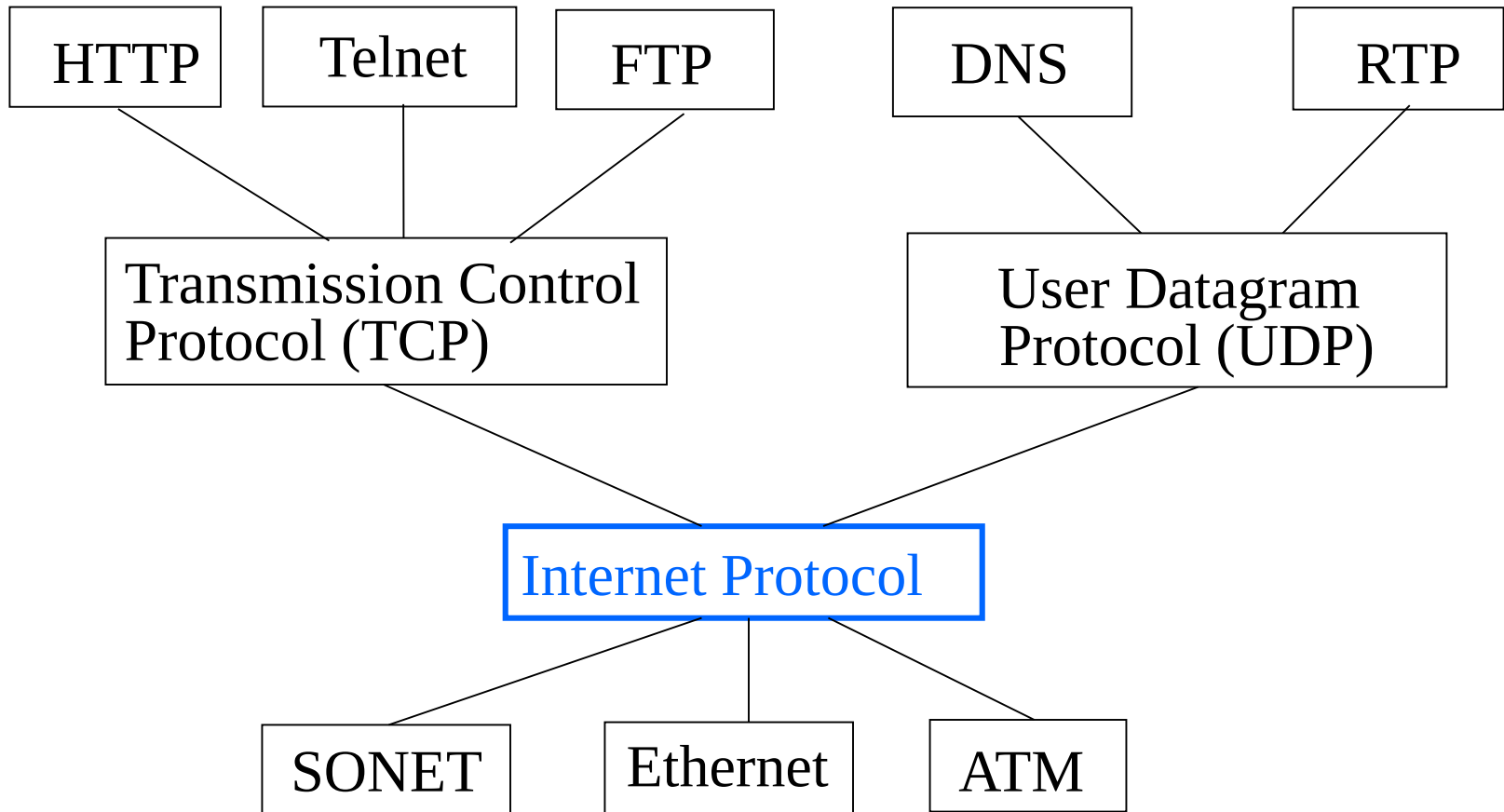
# IP Service Model: Why Best-Effort?

- 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

# IP Service: 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

# Layering in the IP Protocols



# 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

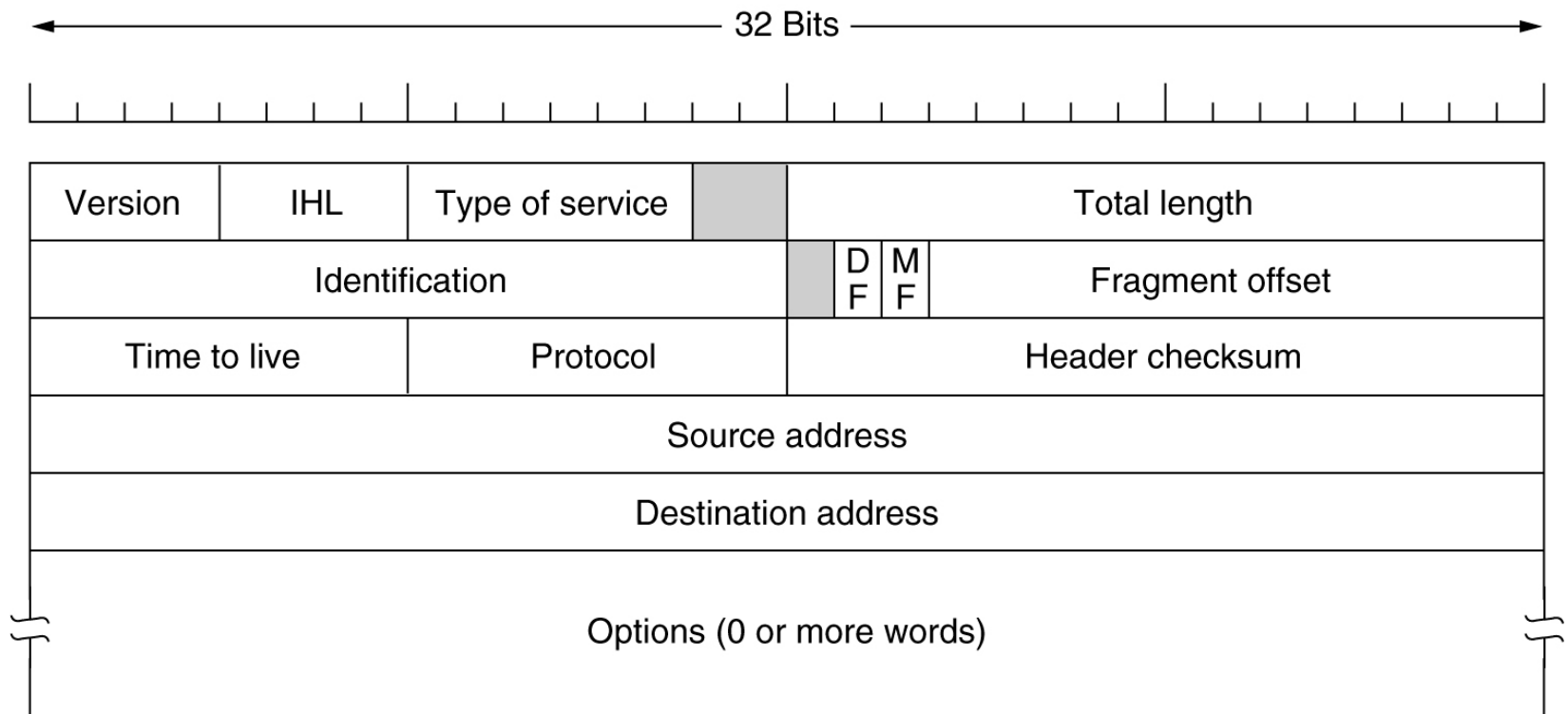
# Other Main Driving Goals (In Order)

- 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

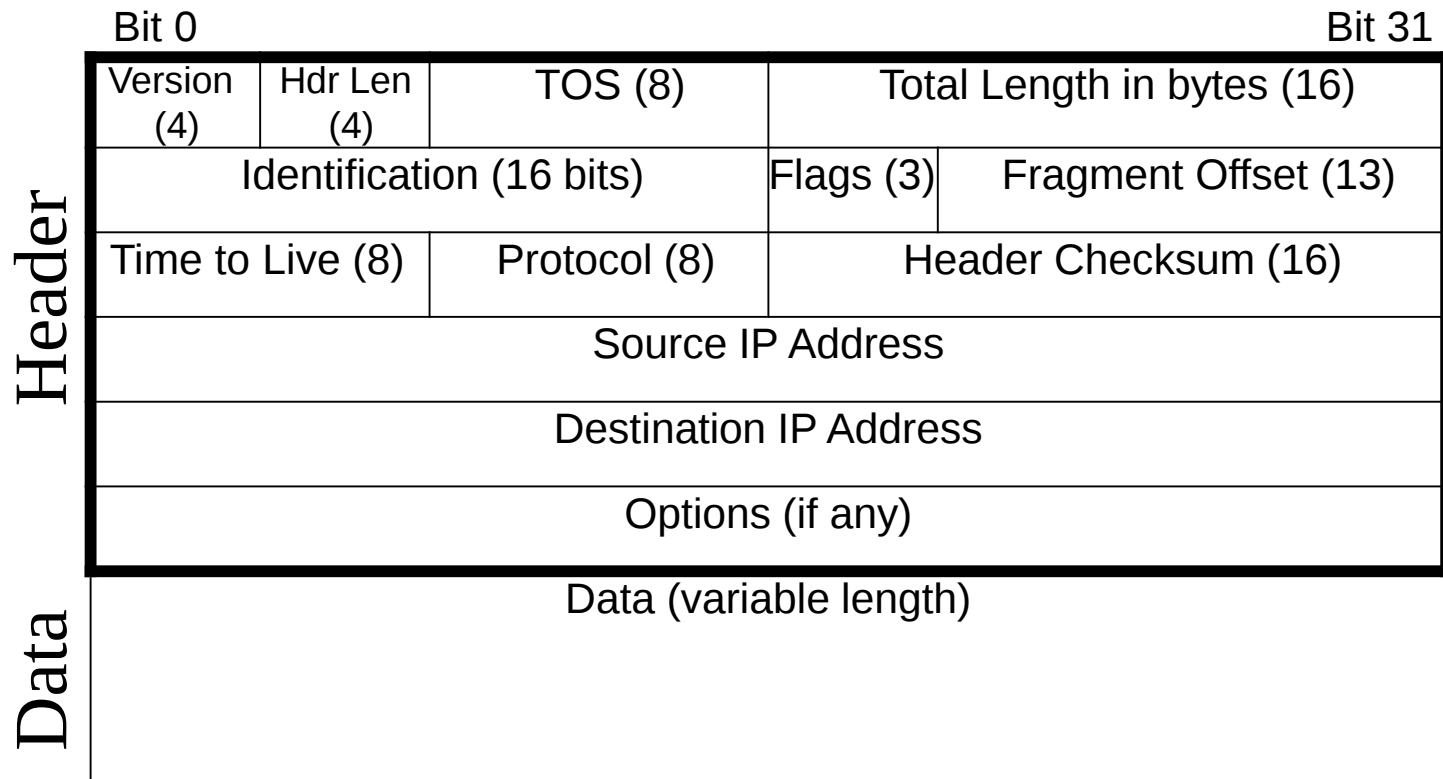
# Other Driving Goals, Somewhat Met

- 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

# The IP Protocol



# IP Datagram



# IP Packet Header

- Version
  - Version number of IP protocol
  - Current version is Version 4
  - Version 6 has different header format

|                          |                |              |                            |
|--------------------------|----------------|--------------|----------------------------|
| Bit 0                    |                | Bit 31       |                            |
| Version<br>(4)           | Hdr Len<br>(4) | TOS (8)      | Total Length in bytes (16) |
| Identification (16 bits) |                | Flags (3)    | Fragment Offset (13)       |
| Time to Live (8)         |                | Protocol (8) | Header Checksum (16)       |
| Source IP Address        |                |              |                            |
| Destination IP Address   |                |              |                            |
| Options (if any)         |                |              |                            |

# IP Packet Header

- Header Length (in 32 bit words)
  - Indicates end of header and beginning of payload
  - If no options, Header length = 5

|                          |                |                      |                            |
|--------------------------|----------------|----------------------|----------------------------|
| Bit 0                    |                | Bit 31               |                            |
| Version<br>(4)           | Hdr Len<br>(4) | TOS (8)              | Total Length in bytes (16) |
| Identification (16 bits) |                | Flags (3)            | Fragment Offset (13)       |
| Time to Live (8)         | Protocol (8)   | Header Checksum (16) |                            |
| Source IP Address        |                |                      |                            |
| Destination IP Address   |                |                      |                            |
| Options (if any)         |                |                      |                            |

# IP Packet Header

- Type of Service (TOS)
  - Allows different types of service to be requested
  - Initially, meaning was not well defined
  - Currently being defined (diffserv)

|                          |                |              |                            |
|--------------------------|----------------|--------------|----------------------------|
| Bit 0                    |                | Bit 31       |                            |
| Version<br>(4)           | Hdr Len<br>(4) | TOS (8)      | Total Length in bytes (16) |
| Identification (16 bits) |                | Flags (3)    | Fragment Offset (13)       |
| Time to Live (8)         |                | Protocol (8) | Header Checksum (16)       |
| Source IP Address        |                |              |                            |
| Destination IP Address   |                |              |                            |
| Options (if any)         |                |              |                            |

# IP Packet Header

- Packet Length (in Bytes)
  - Unambiguously specify end of packet
  - Max packet size =  $2^{16} = 65,535$  Bytes

|                          |                |              |                            |
|--------------------------|----------------|--------------|----------------------------|
| Bit 0                    |                | Bit 31       |                            |
| Version<br>(4)           | Hdr Len<br>(4) | TOS (8)      | Total Length in bytes (16) |
| Identification (16 bits) |                | Flags (3)    | Fragment Offset (13)       |
| Time to Live (8)         |                | Protocol (8) | Header Checksum (16)       |
| Source IP Address        |                |              |                            |
| Destination IP Address   |                |              |                            |
| Options (if any)         |                |              |                            |

# IP Packet Header

- These three fields for Fragmentation Control  
(will come back to them later)

| Bit 0                    |                | Bit 31       |                            |
|--------------------------|----------------|--------------|----------------------------|
| Version<br>(4)           | Hdr Len<br>(4) | TOS (8)      | Total Length in bytes (16) |
| Identification (16 bits) |                | Flags (3)    | Fragment Offset (13)       |
| Time to Live (8)         |                | Protocol (8) | Header Checksum (16)       |
| Source IP Address        |                |              |                            |
| Destination IP Address   |                |              |                            |
| Options (if any)         |                |              |                            |

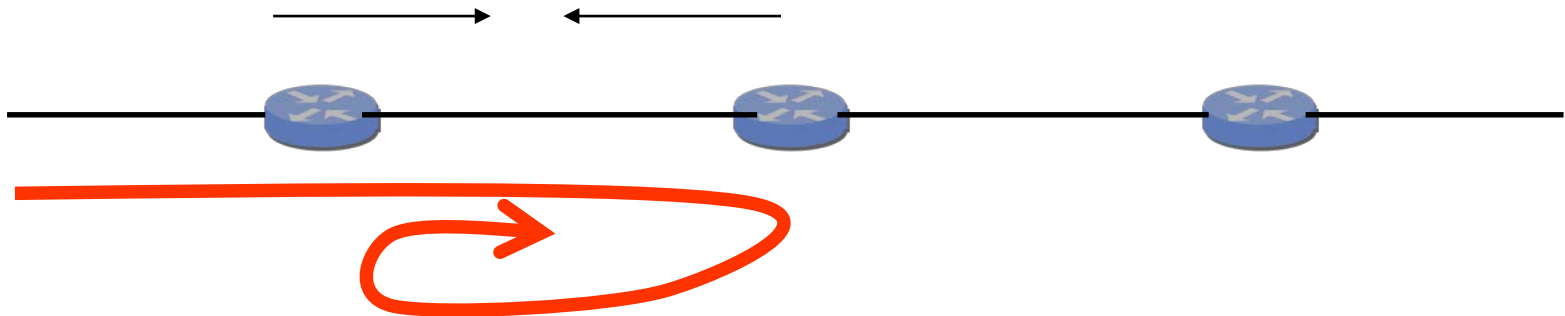
# IP Packet Header

- Time to Live
  - Initially set by sender (up to 255)
  - Decrement by each router
  - Discard when TTL = 0 to avoid infinite routing loops

|                          |                |              |                            |                      |
|--------------------------|----------------|--------------|----------------------------|----------------------|
| Version<br>(4)           | Hdr Len<br>(4) | TOS (8)      | Total Length in bytes (16) |                      |
| Identification (16 bits) |                |              | Flags (3)                  | Fragment Offset (13) |
| Time to Live (8)         |                | Protocol (8) | Header Checksum (16)       |                      |
| Source IP Address        |                |              |                            |                      |
| Destination IP Address   |                |              |                            |                      |
| Options (if any)         |                |              |                            |                      |

# Time-to-Live (TTL) Field

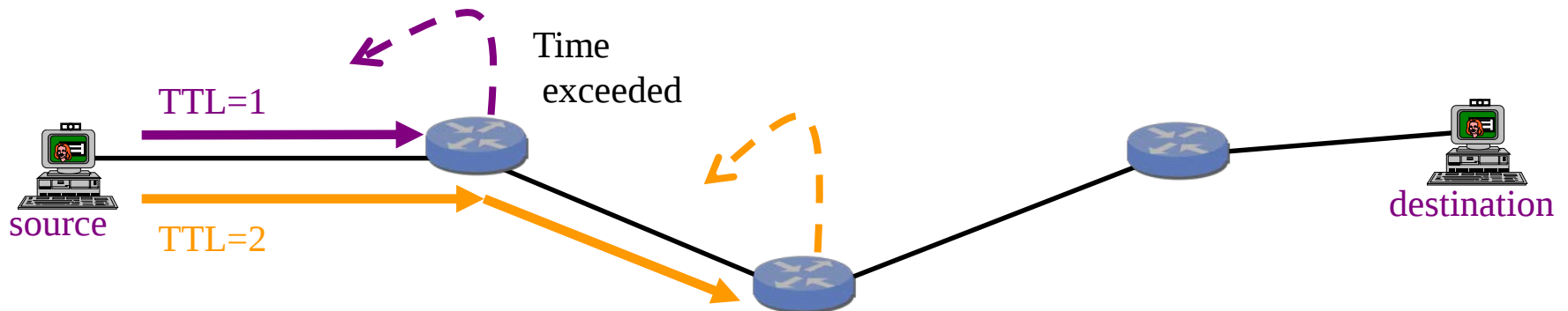
- Potential robustness problem
  - Forwarding loops can cause packets to cycle forever
  - Confusing if the packet arrives much later



- 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

# 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

# Example Traceroute: Berkeley to CNN

## Hop number, IP address, DNS name

No response  
from router

|    |                 |                                        |
|----|-----------------|----------------------------------------|
| 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 name resolution

# Try Running Traceroute Yourself

- On UNIX machine
  - Traceroute
  - E.g., “traceroute [www.cnn.com](http://www.cnn.com)” or “traceroute 12.1.1.1”
- On Windows machine
  - Tracert
  - E.g., “tracert [www.cnn.com](http://www.cnn.com)” or “tracert 12.1.1.1”
- Common uses of traceroute
  - Discover the topology of the Internet
  - Debug performance and reachability problems

# IP Packet Header

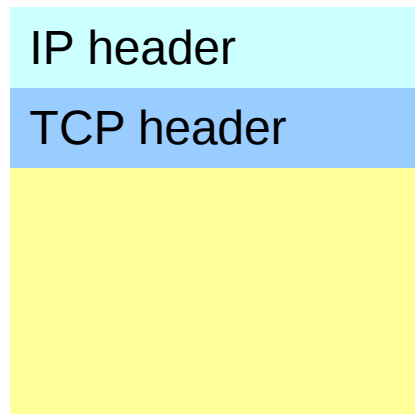
- Protocol
  - Value indicates what is in the data field
  - Example: TCP or UDP

|                          |                |         |                            |                      |
|--------------------------|----------------|---------|----------------------------|----------------------|
| Bit 0                    |                |         | Bit 31                     |                      |
| Version<br>(4)           | Hdr Len<br>(4) | TOS (8) | Total Length in bytes (16) |                      |
| Identification (16 bits) |                |         | Flags (3)                  | Fragment Offset (13) |
| Time to Live (8)         | Protocol (8)   |         | Header Checksum (16)       |                      |
| Source IP Address        |                |         |                            |                      |
| Destination IP Address   |                |         |                            |                      |
| Options (if any)         |                |         |                            |                      |

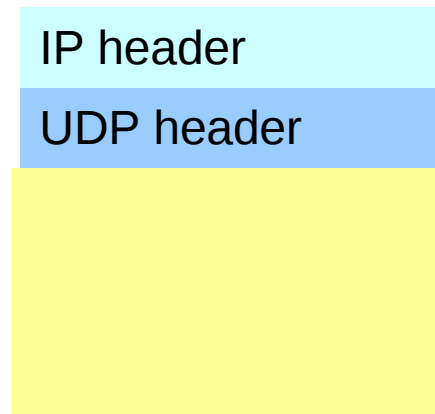
# IP Packet Header Fields (Continued)

- 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

protocol=6



protocol=17



# IP Packet Header

- Header Checksum
  - Checks for error in the header only
  - Bad headers can harm the network
  - If error found, packet is simply discarded

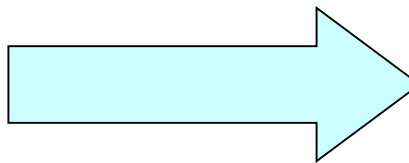
- Bit 31

|                          |                |         |                            |                      |
|--------------------------|----------------|---------|----------------------------|----------------------|
| Version<br>(4)           | Hdr Len<br>(4) | TOS (8) | Total Length in bytes (16) |                      |
| Identification (16 bits) |                |         | Flags (3)                  | Fragment Offset (13) |
| Time to Live (8)         | Protocol (8)   |         | Header Checksum (16)       |                      |
| Source IP Address        |                |         |                            |                      |
| Destination IP Address   |                |         |                            |                      |
| Options (if any)         |                |         |                            |                      |

# 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

$$\begin{array}{r} 134 \\ + 212 \\ \hline = 346 \end{array}$$



$$\begin{array}{r} 134 \\ + 216 \\ \hline = 350 \end{array}$$

**Mismatch!**

# IP Packet Header

- Source and Destination IP Addresses
  - Strings of 32 ones and zeros

| Bit 0                    |                | Bit 31       |                            |
|--------------------------|----------------|--------------|----------------------------|
| Version<br>(4)           | Hdr Len<br>(4) | TOS (8)      | Total Length in bytes (16) |
| Identification (16 bits) |                | Flags (3)    | Fragment Offset (13)       |
| Time to Live (8)         |                | Protocol (8) | Header Checksum (16)       |
| Source IP Address        |                |              |                            |
| Destination IP Address   |                |              |                            |
| Options (if any)         |                |              |                            |

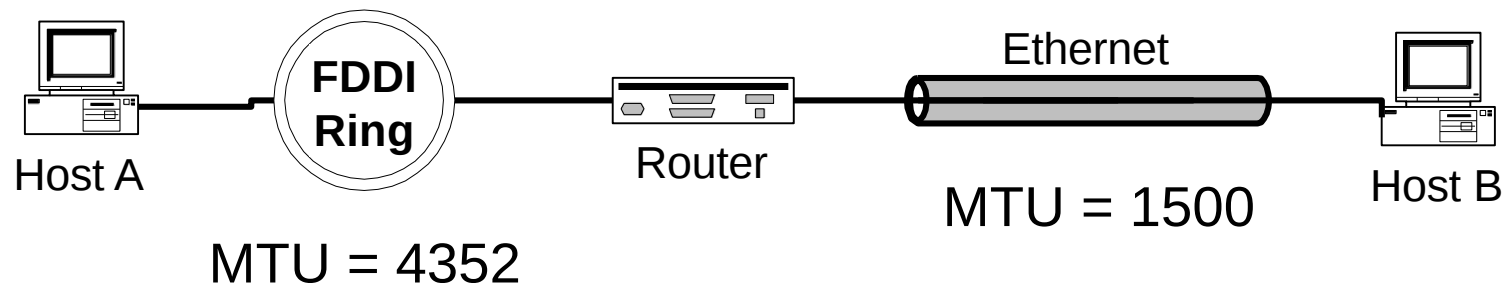
# IP Packet Header

- Options
  - Example: timestamp, record route, source route

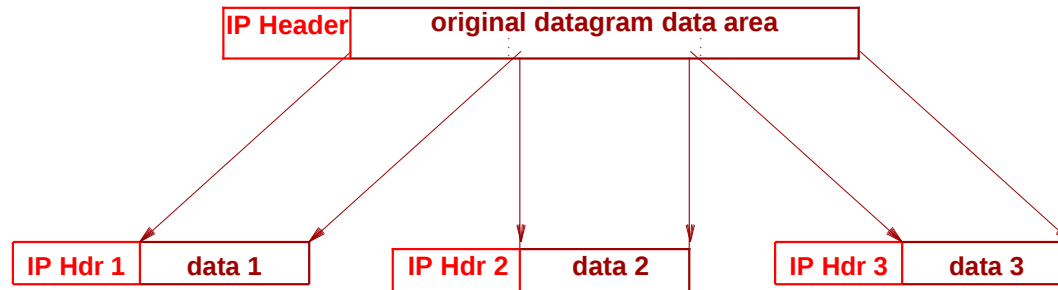
| Bit 0                    |                | Bit 31       |                            |
|--------------------------|----------------|--------------|----------------------------|
| Version<br>(4)           | Hdr Len<br>(4) | TOS (8)      | Total Length in bytes (16) |
| Identification (16 bits) |                | Flags (3)    | Fragment Offset (13)       |
| Time to Live (8)         |                | Protocol (8) | Header Checksum (16)       |
| Source IP Address        |                |              |                            |
| Destination IP Address   |                |              |                            |
| Options (if any)         |                |              |                            |

# IP Fragmentation & Reassembly

- Maximum Transmission Unit (MTU)
  - Largest IP packet a network will accept
  - Arriving IP packet may be larger (max IP packet size = 65,535 bytes)
- Sender or router will split the packet into multiple fragments
- Destination will reassemble the packet
- IP header fields used to identify and order related fragments



# Illustration Of Datagram Fragmentation



Each fragment has IP datagram header

Header fields

- Identify original datagram
- Indicate where fragment fits

# IP Packet Header

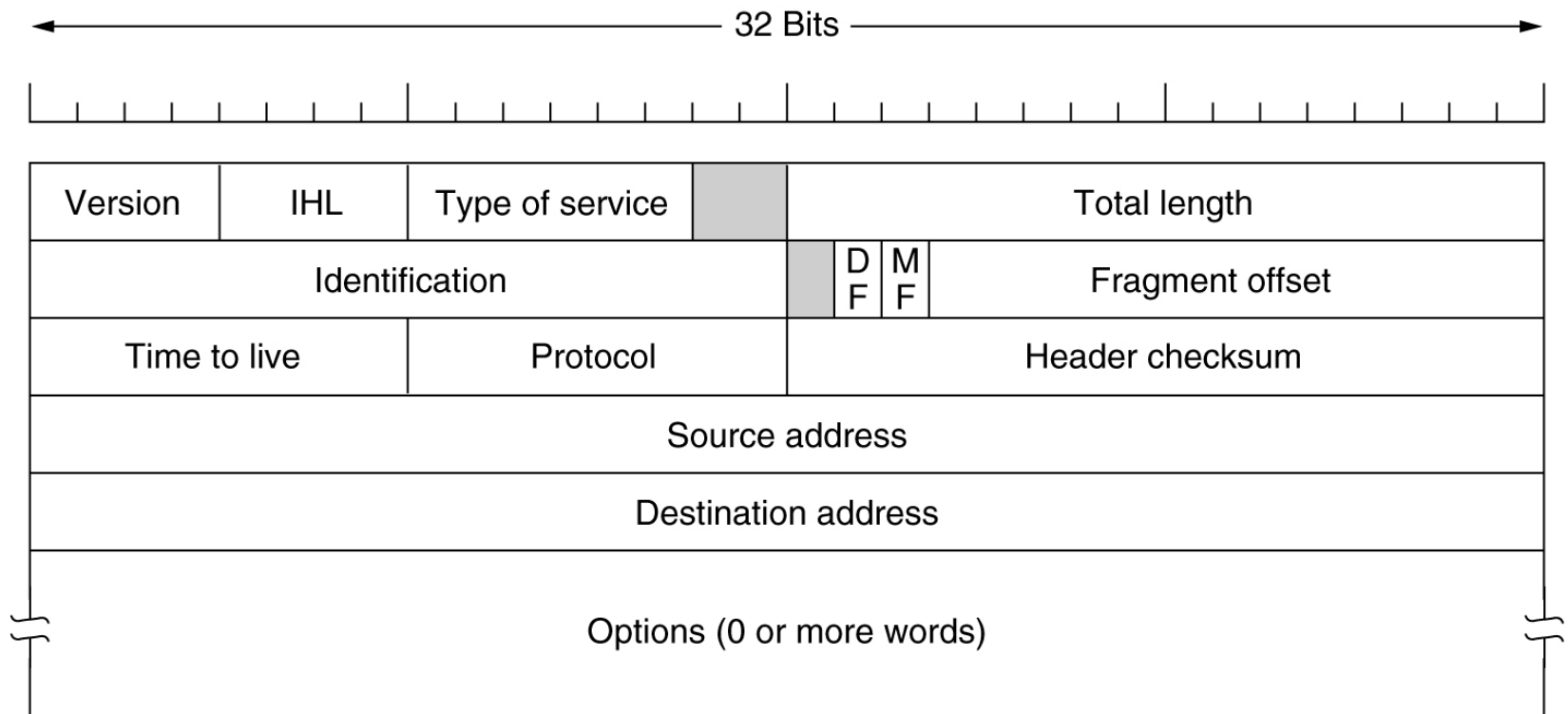
- Identification
  - – All fragments of a single datagram have the same identification number

• Bit 0

Bit 31

|                          |                |              |                            |                      |
|--------------------------|----------------|--------------|----------------------------|----------------------|
| Version<br>(4)           | Hdr Len<br>(4) | TOS (8)      | Total Length in bytes (16) |                      |
| Identification (16 bits) |                |              | Flags (3)                  | Fragment Offset (13) |
| Time to Live (8)         |                | Protocol (8) | Header Checksum (16)       |                      |
| Source IP Address        |                |              |                            |                      |
| Destination IP Address   |                |              |                            |                      |
| Options (if any)         |                |              |                            |                      |

# The IP Protocol



# IP Packet Header

- Flags:
  - 1<sup>st</sup> bit: reserved, must be zero
  - 2<sup>nd</sup> bit: DF -- Do Not Fragment
  - 3<sup>rd</sup> bit: MF -- More Fragments

|                          |                |              |                            |
|--------------------------|----------------|--------------|----------------------------|
| Bit 0                    |                | Bit 31       |                            |
| Version<br>(4)           | Hdr Len<br>(4) | TOS (8)      | Total Length in bytes (16) |
| Identification (16 bits) |                | Flags (3)    | Fragment Offset (13)       |
| Time to Live (8)         |                | Protocol (8) | Header Checksum (16)       |
| Source IP Address        |                |              |                            |
| Destination IP Address   |                |              |                            |
| Options (if any)         |                |              |                            |

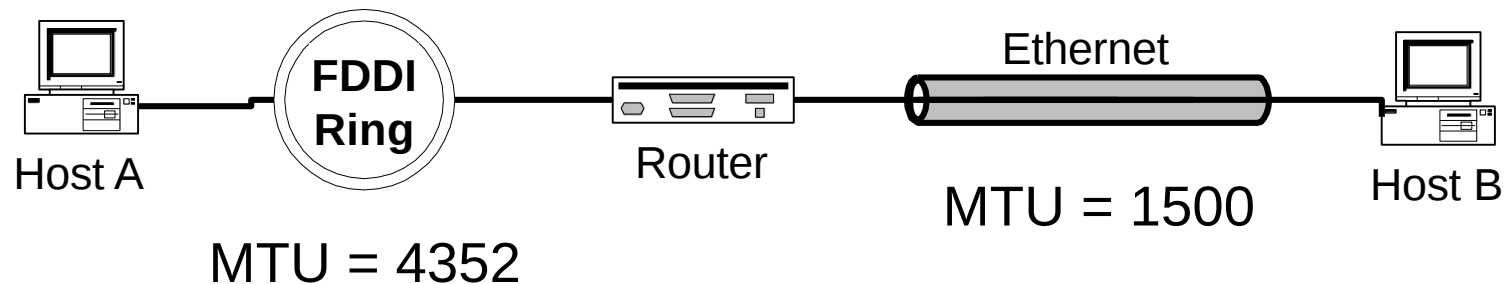
# IP Packet Header

- Fragment Offset (in units of 8 bytes)
  - Used for reassembly of packet
  - 1<sup>st</sup> fragment has offset = 0

| Bit 0                    |                | Bit 31       |                            |
|--------------------------|----------------|--------------|----------------------------|
| Version<br>(4)           | Hdr Len<br>(4) | TOS (8)      | Total Length in bytes (16) |
| Identification (16 bits) |                | Flags (3)    | Fragment Offset (13)       |
| Time to Live (8)         |                | Protocol (8) | Header Checksum (16)       |
| Source IP Address        |                |              |                            |
| Destination IP Address   |                |              |                            |
| Options (if any)         |                |              |                            |

# IP Fragmentation Example

- Host A wants to send to Host B an IP datagram of size = 4000 Bytes



# IP Fragmentation Example

|  |        |    |    |        |  |
|--|--------|----|----|--------|--|
|  | length | ID | MF | offset |  |
|  | =4000  | =x | =0 | =0     |  |

One large datagram becomes  
several smaller datagrams

|  |        |    |    |        |  |
|--|--------|----|----|--------|--|
|  | length | ID | MF | offset |  |
|  | =1500  | =x | =1 | =0     |  |

|  |        |    |    |        |  |
|--|--------|----|----|--------|--|
|  | length | ID | MF | offset |  |
|  | =1500  | =x | =1 | =1480  |  |

|  |        |    |    |        |  |
|--|--------|----|----|--------|--|
|  | length | ID | MF | offset |  |
|  | =1040  | =x | =0 | =2960  |  |

# Multiple Fragmenting Points

Let MTUs along internet path be

- 1500
- 1500
- 1000
- 1500
- 576
- 1500

Result: fragmentation can occur twice

# Fragmenting A Fragment

Needed when fragment too large for network MTU

Arbitrary subfragmentation possible

Router divides fragments into smaller pieces

All fragments at same “level”

- Offset given with respect to original datagram
- Destination cannot distinguish subfragments

# Fragment Loss

## Receiver

- Collects incoming fragments
- Reassembles when all fragments arrive
- Does not know identity of router that did fragmentation
- Cannot request missing pieces

Consequence: Loss of one fragment means entire datagram lost

# 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

# The IP Protocol

Some of the IP options.

| Option                | Description                                        |
|-----------------------|----------------------------------------------------|
| Security              | Specifies how secret the datagram is               |
| Strict source routing | Gives the complete path to be followed             |
| Loose source routing  | Gives a list of routers not to be missed           |
| Record route          | Makes each router append its IP address            |
| Timestamp             | Makes each router append its address and timestamp |

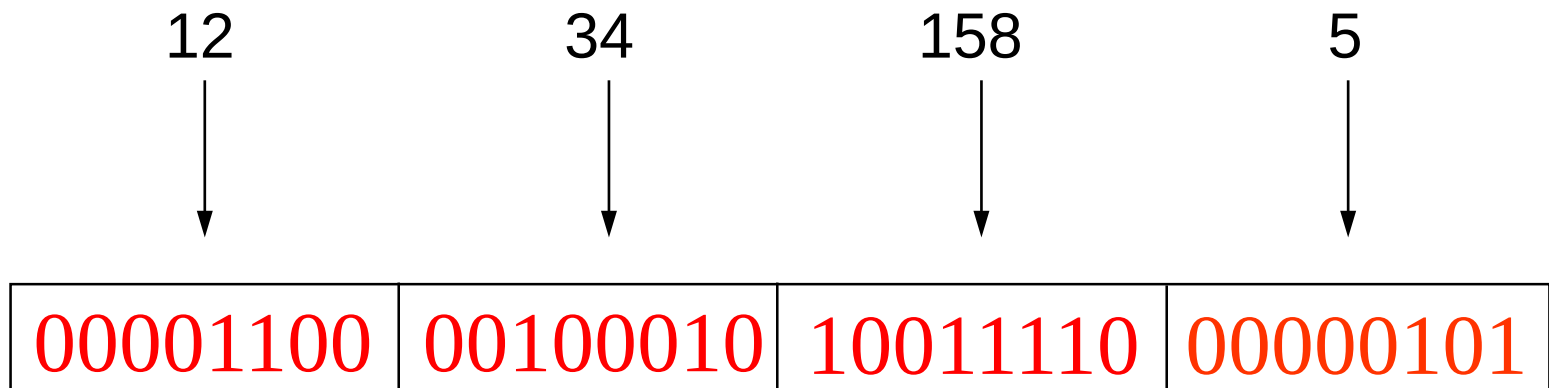
# What if the Source Lies?

- 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

# IP Addressing and Forwarding

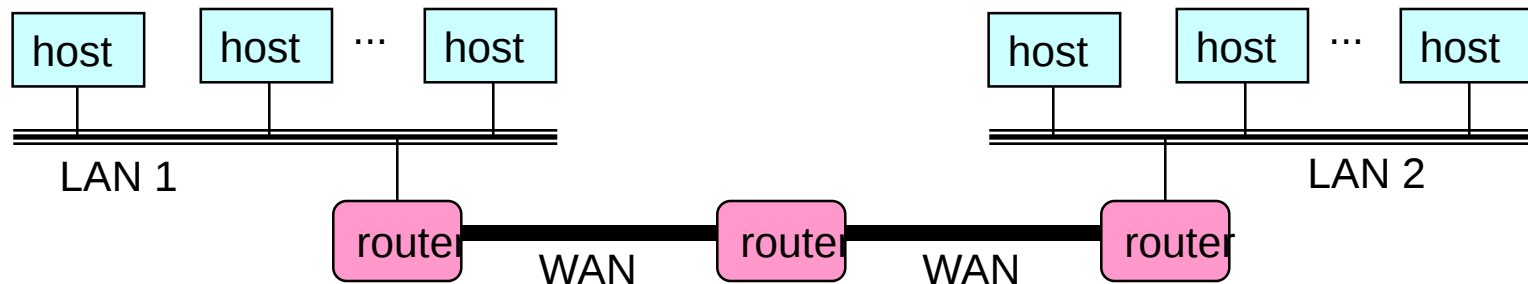
# IP Address (IPv4)

- A unique 32-bit number
- Identifies an interface (on a host, on a router, ...)
- Represented in dotted-quad notation



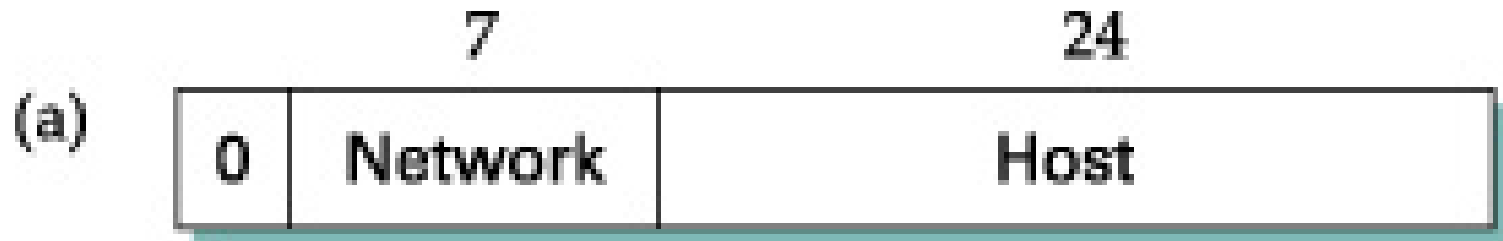
# Grouping Related Hosts

- The Internet is an “inter-network”
  - Used to connect *networks* together, not *hosts*
  - Needs a way to address a network (i.e., group of hosts)

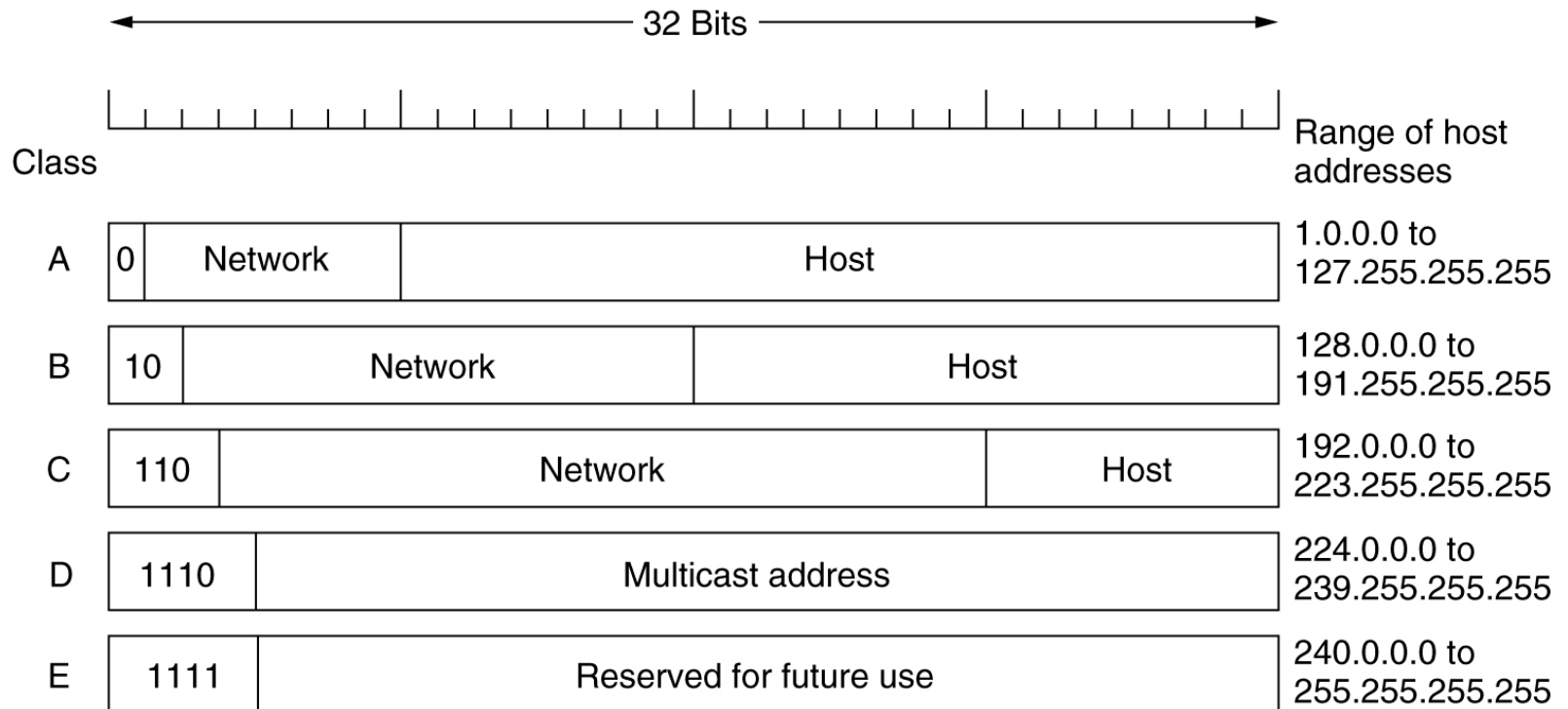


LAN = Local Area Network  
WAN = Wide Area Network

# IP Address Classes



# IP Addresses



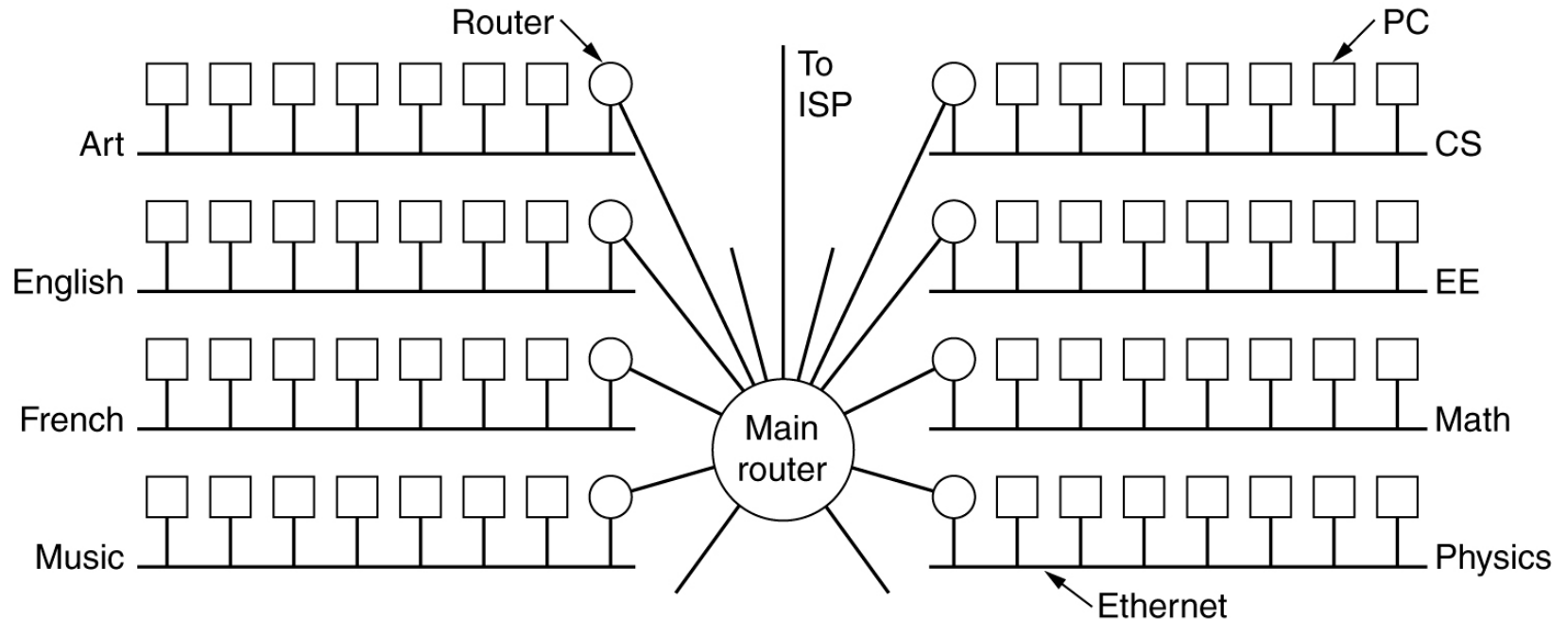
# IP Addresses (2)

## Special IP addresses.

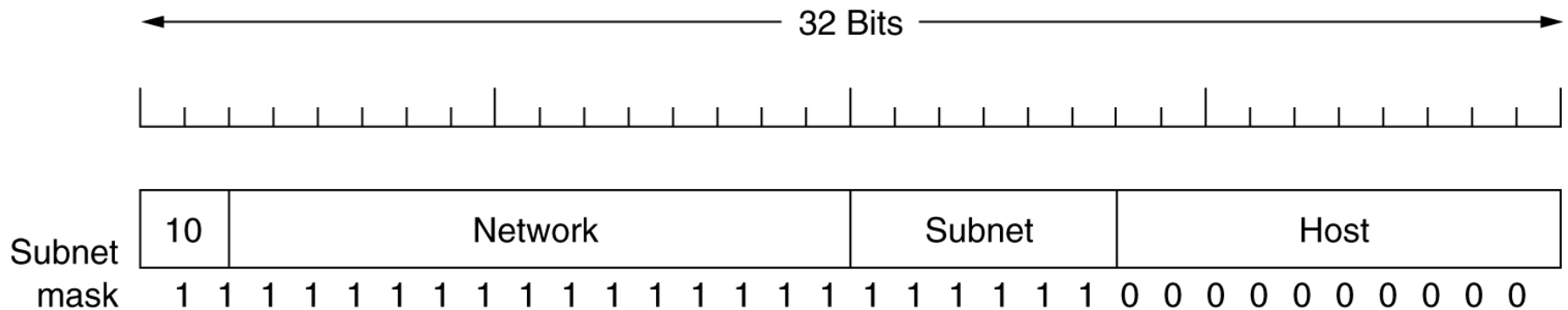
|                                                                 |  |  |  |  |  |  |  |            |  |  |  |  |  |  |  |         |  |  |  |  |  |  |  |     |  |  |  |  |  |  |  |                                |  |  |  |  |  |  |  |         |  |  |  |  |  |  |  |                                |
|-----------------------------------------------------------------|--|--|--|--|--|--|--|------------|--|--|--|--|--|--|--|---------|--|--|--|--|--|--|--|-----|--|--|--|--|--|--|--|--------------------------------|--|--|--|--|--|--|--|---------|--|--|--|--|--|--|--|--------------------------------|
| 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 |  |  |  |  |  |  |  |            |  |  |  |  |  |  |  |         |  |  |  |  |  |  |  |     |  |  |  |  |  |  |  | This host                      |  |  |  |  |  |  |  |         |  |  |  |  |  |  |  |                                |
| 0 0                                                             |  |  |  |  |  |  |  | ...        |  |  |  |  |  |  |  |         |  |  |  |  |  |  |  | 0 0 |  |  |  |  |  |  |  | Host                           |  |  |  |  |  |  |  |         |  |  |  |  |  |  |  | A host on this network         |
| 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 |  |  |  |  |  |  |  |            |  |  |  |  |  |  |  |         |  |  |  |  |  |  |  |     |  |  |  |  |  |  |  | Broadcast on the local network |  |  |  |  |  |  |  |         |  |  |  |  |  |  |  |                                |
| Network                                                         |  |  |  |  |  |  |  |            |  |  |  |  |  |  |  | 1 1 1 1 |  |  |  |  |  |  |  | ... |  |  |  |  |  |  |  |                                |  |  |  |  |  |  |  | 1 1 1 1 |  |  |  |  |  |  |  | Broadcast on a distant network |
| 127                                                             |  |  |  |  |  |  |  | (Anything) |  |  |  |  |  |  |  |         |  |  |  |  |  |  |  |     |  |  |  |  |  |  |  | Loopback                       |  |  |  |  |  |  |  |         |  |  |  |  |  |  |  |                                |

# Subnets

A campus network consisting of LANs for various departments.



# Subnets (2)



A class B network subnetted into 64 subnets.

# Subnetted Address

| Network number | Host number |
|----------------|-------------|
|----------------|-------------|

Class B address

|                                  |          |
|----------------------------------|----------|
| 11111111111111111111111111111111 | 00000000 |
|----------------------------------|----------|

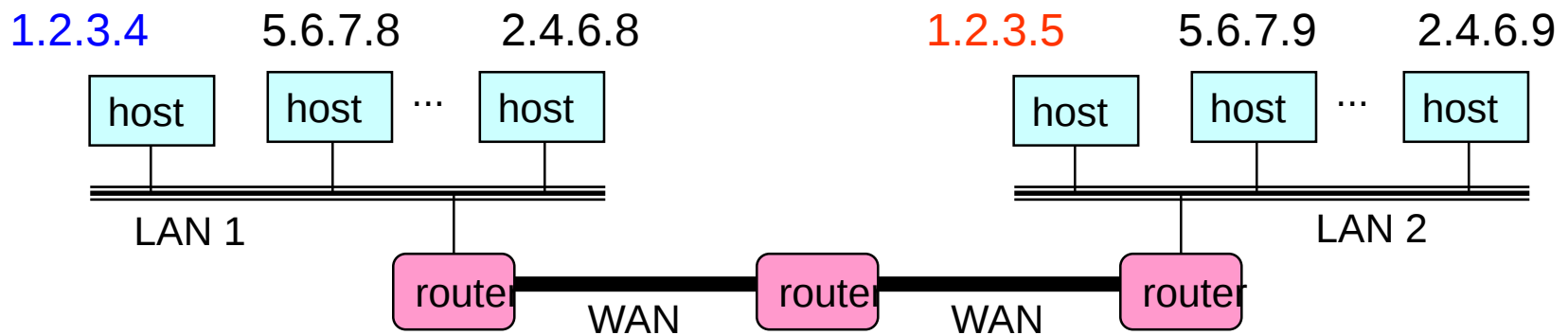
Subnet mask (255.255.255.0)

| Network number | Subnet ID | Host ID |
|----------------|-----------|---------|
|----------------|-----------|---------|

Subnetted address

# Scalability Challenge

- Suppose hosts had arbitrary addresses
  - Then every router would need a lot of information
  - ...to know how to direct packets toward the host

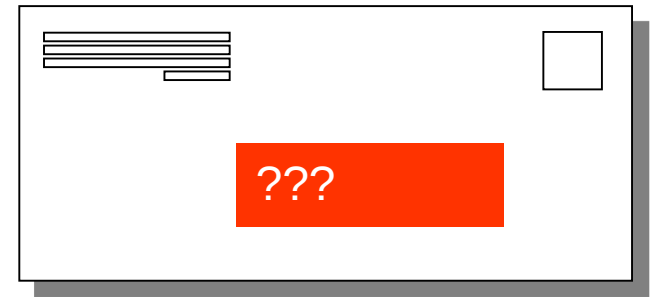


|         |   |
|---------|---|
| 1.2.3.4 | ← |
| 1.2.3.5 | → |
| ⋮       |   |

forwarding table

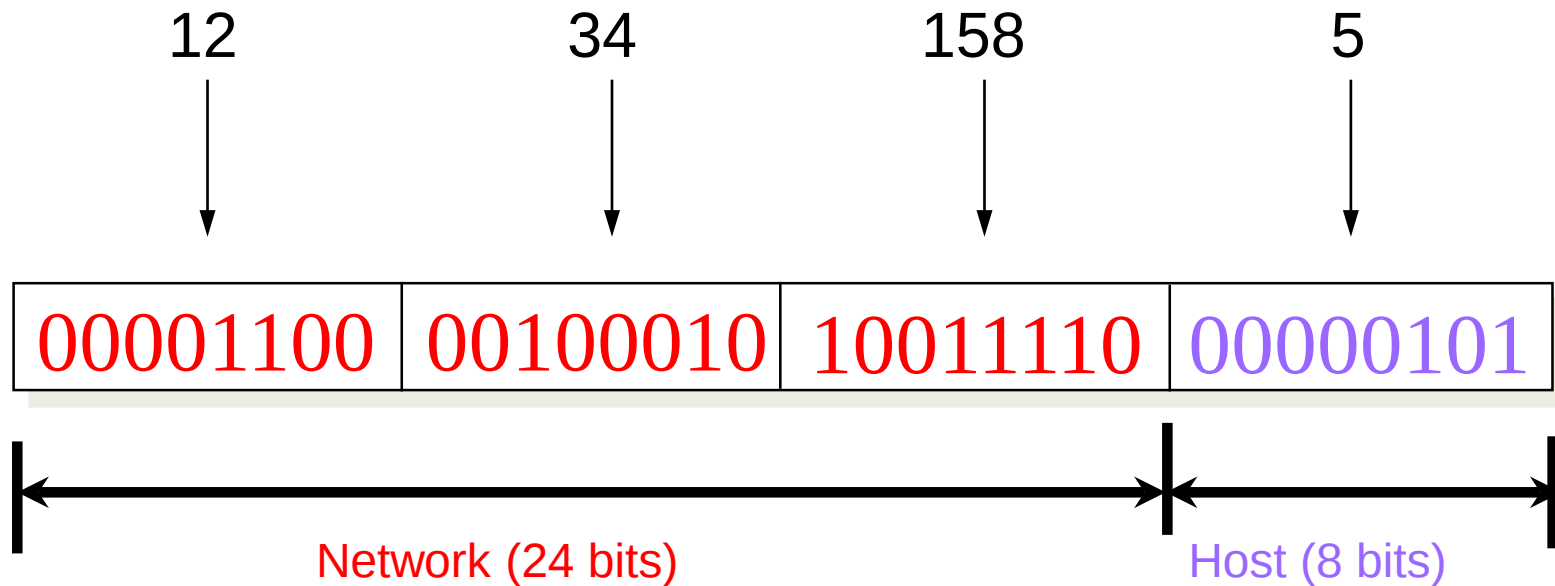
# Hierarchical Addressing in U.S. Mail

- Addressing in the U.S. mail
  - Zip code: 08540
  - Street: Olden Street
  - Building on street: 35
  - Room in building: 306
  - Name of occupant: Jennifer Rexford
- Forwarding the U.S. mail
  - Deliver letter to the post office in the zip code
  - Assign letter to mailman covering the street
  - Drop letter into mailbox for the building/room
  - Give letter to the appropriate person



# Hierarchical Addressing: IP Prefixes

- Divided into network & host portions (left and right)
- 12.34.158.0/24 is a 24-bit prefix with  $2^8$  addresses



# IP Address and a 24-bit Subnet Mask

Address

12

34

158

5



|          |          |          |          |
|----------|----------|----------|----------|
| 00001100 | 00100010 | 10011110 | 00000101 |
|----------|----------|----------|----------|

|          |          |          |          |
|----------|----------|----------|----------|
| 11111111 | 11111111 | 11111111 | 00000000 |
|----------|----------|----------|----------|



255

255

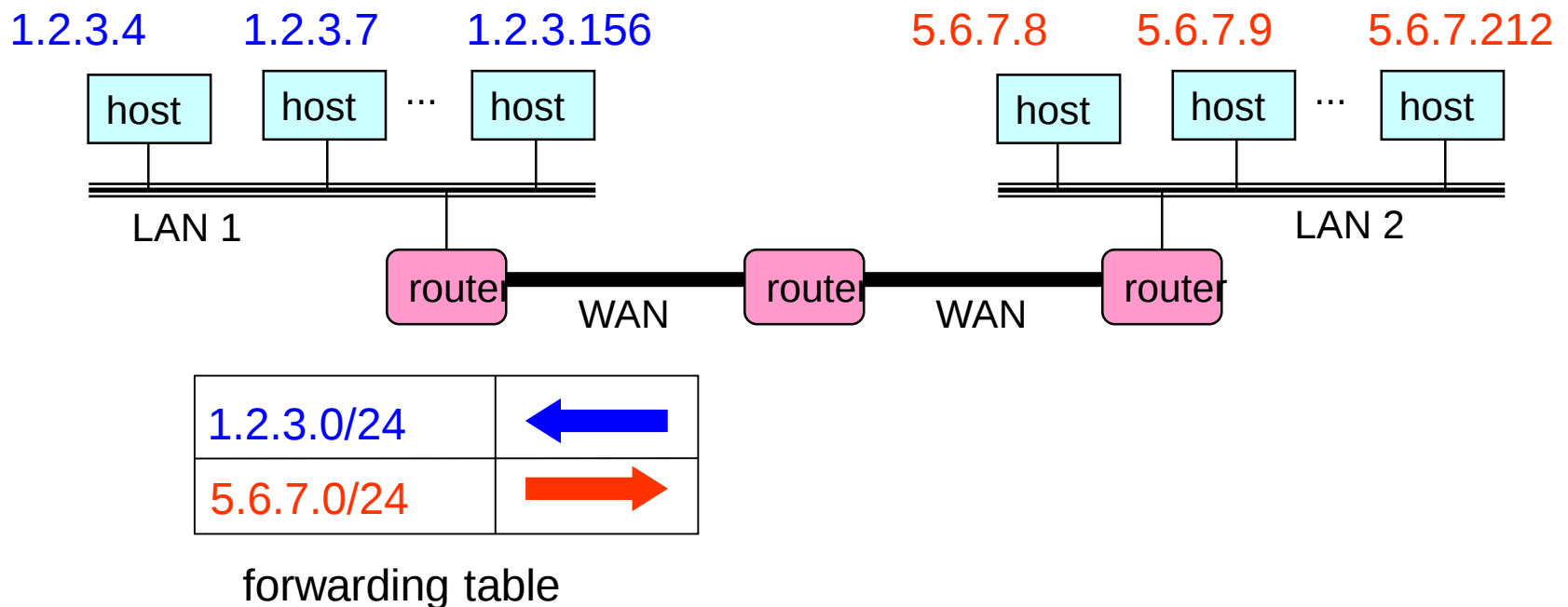
255

0

Mask

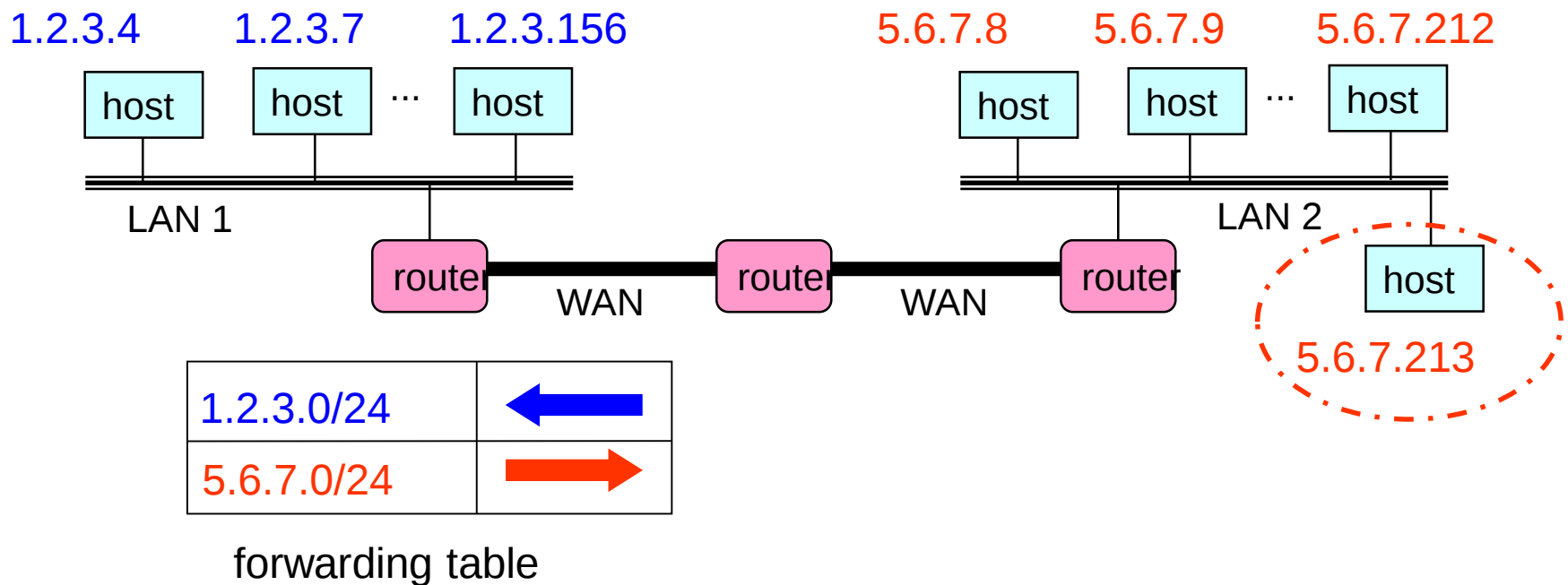
# Scalability Improved

- Number related hosts from a common subnet
  - 1.2.3.0/24 on the left LAN
  - 5.6.7.0/24 on the right LAN



# Easy to Add New Hosts

- No need to update the routers
  - E.g., adding a new host 5.6.7.213 on the right
  - Doesn't require adding a new forwarding entry



# Address Allocation

# Classful Addressing

- In the olden days, only fixed allocation sizes
  - Class A: 0\*
    - Very large /8 blocks (e.g., MIT has 18.0.0.0/8)
  - Class B: 10\*
    - Large /16 blocks (e.g., Princeton has 128.112.0.0/16)
  - Class C: 110\*
    - Small /24 blocks (e.g., AT&T Labs has 192.20.225.0/24)
  - Class D: 1110\*
    - Multicast groups
  - Class E: 11110\*
    - Reserved for future use
- This is why folks use dotted-quad notation!

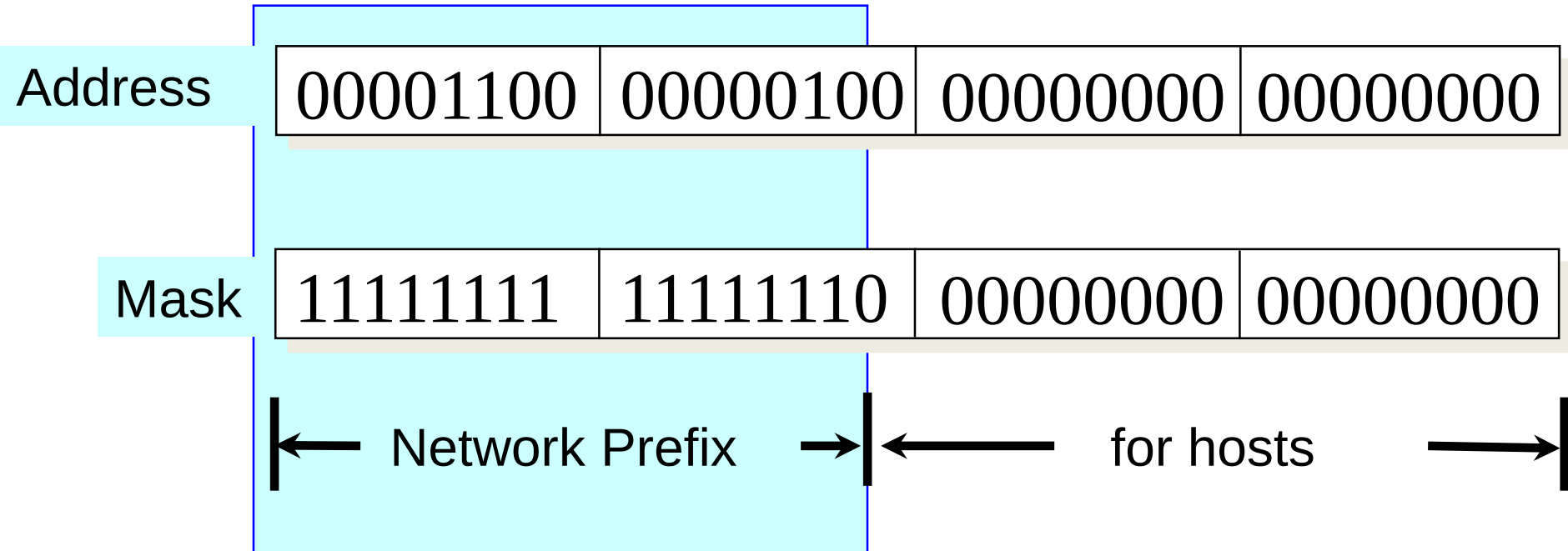
# Classless Inter-Domain Routing (CIDR)

Use two 32-bit numbers to represent a network.

Network number = IP address + Mask

IP Address : 12.4.0.0

IP Mask: 255.254.0.0



Written as 12.4.0.0/15

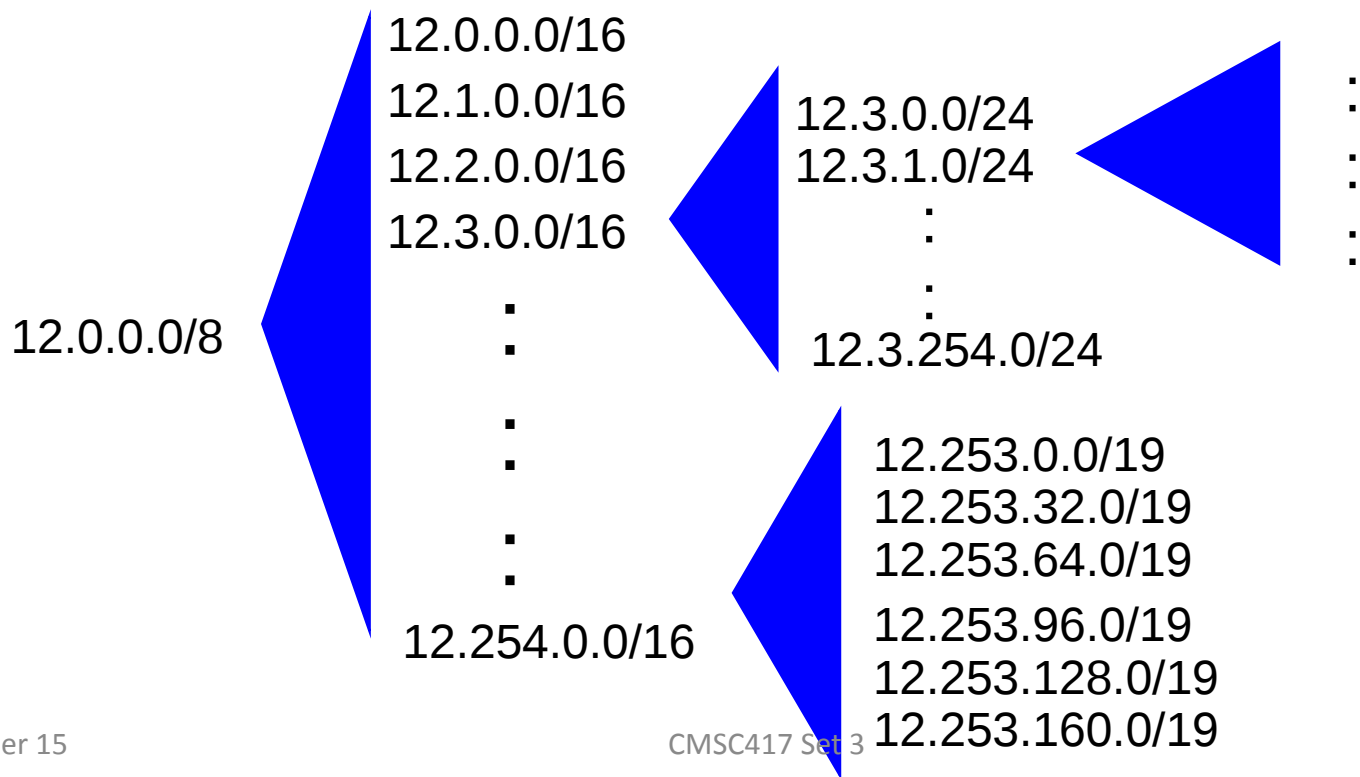
# CDR – Classless InterDomain Routing

A set of IP address assignments.

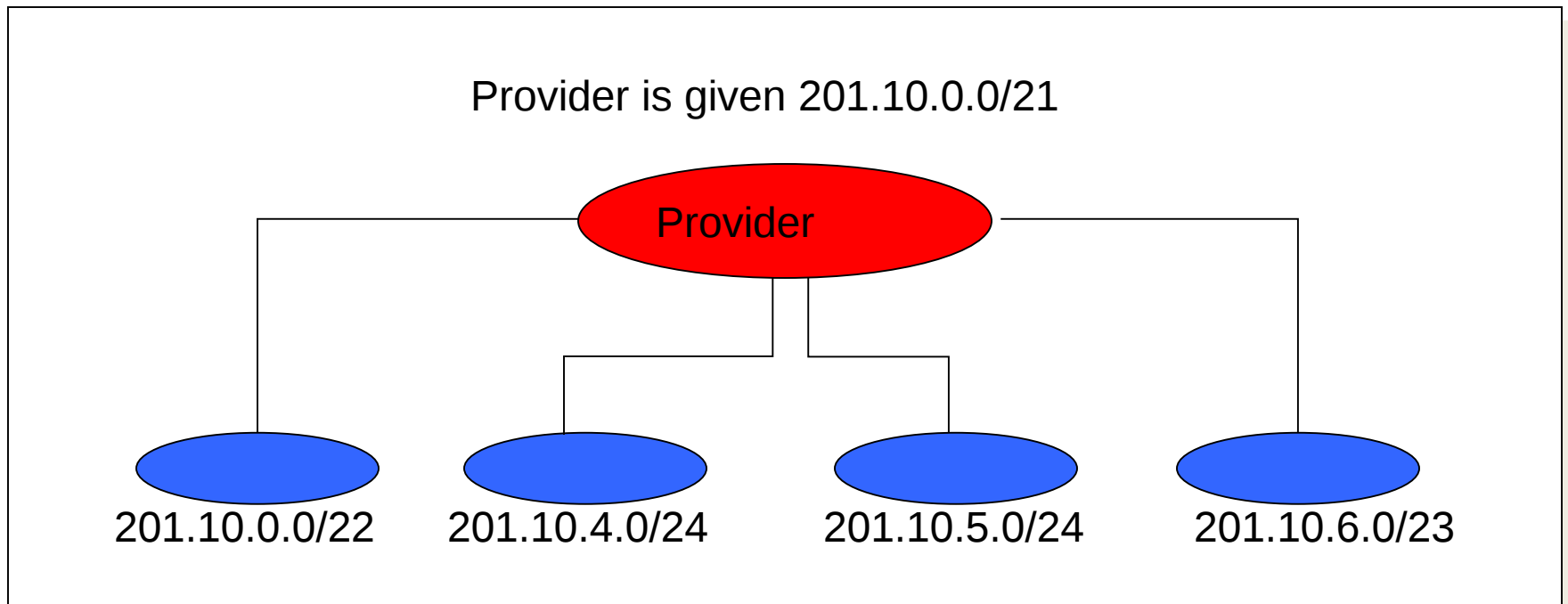
| University  | First address | Last address  | How many | Written as     |
|-------------|---------------|---------------|----------|----------------|
| Cambridge   | 194.24.0.0    | 194.24.7.255  | 2048     | 194.24.0.0/21  |
| Edinburgh   | 194.24.8.0    | 194.24.11.255 | 1024     | 194.24.8.0/22  |
| (Available) | 194.24.12.0   | 194.24.15.255 | 1024     | 194.24.12/22   |
| Oxford      | 194.24.16.0   | 194.24.31.255 | 4096     | 194.24.16.0/20 |

# CIDR: Hierarchal Address Allocation

- Prefixes are key to Internet scalability
  - Address allocated in contiguous chunks (prefixes)
  - Routing protocols and packet forwarding based on prefixes
  - Today, routing tables contain ~150,000-200,000 prefixes

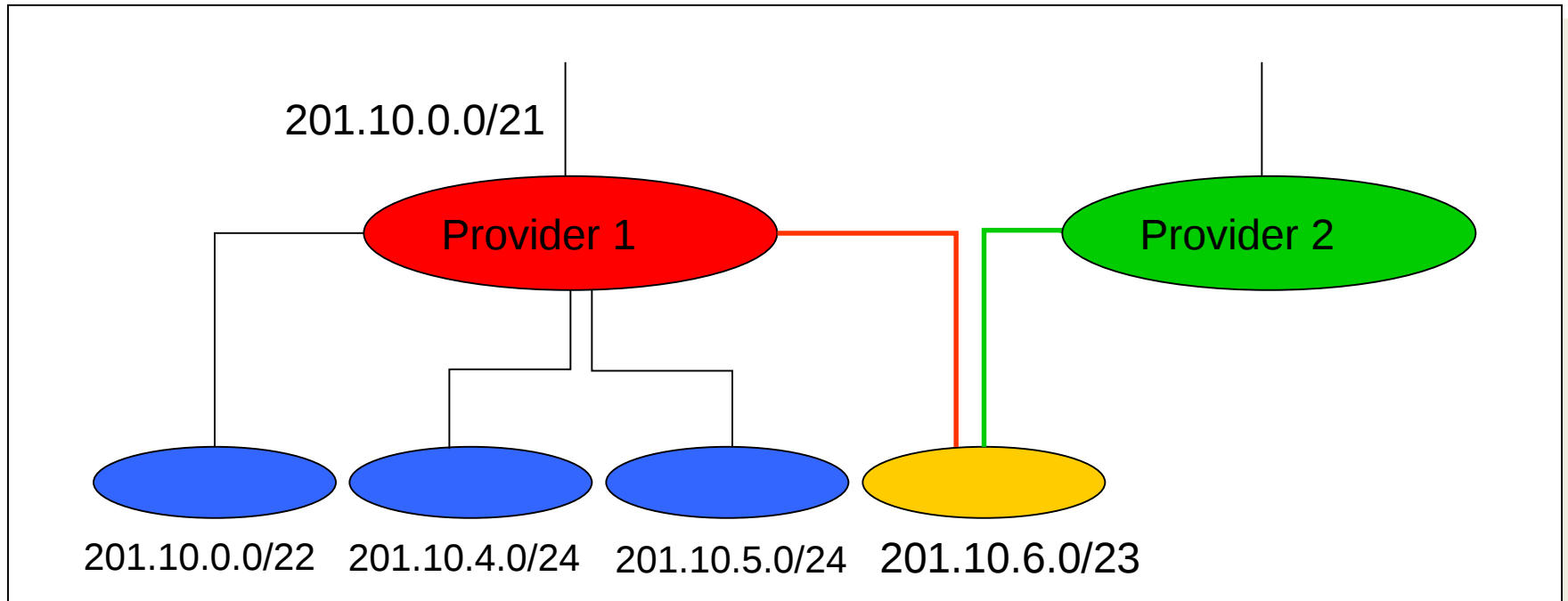


# Scalability: Address Aggregation



Routers in the rest of the Internet just need to know how to reach **201.10.0.0/21**. The provider can direct the IP packets to the appropriate **customer**.

# But, Aggregation Not Always Possible

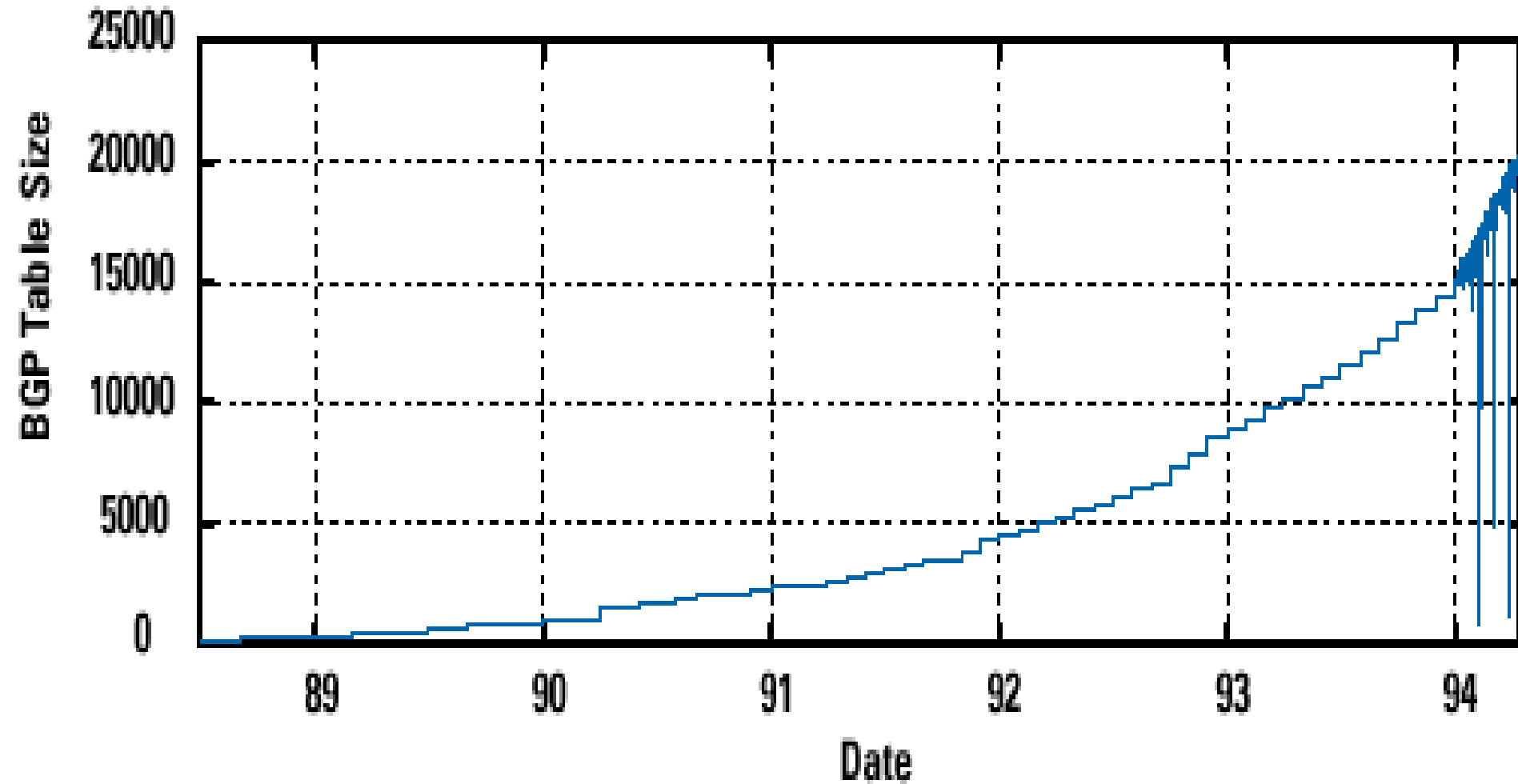


*Multi-homed* customer with 201.10.6.0/23 has two providers. Other parts of the Internet need to know how to reach these destinations through *both* providers.

# Scalability Through Hierarchy

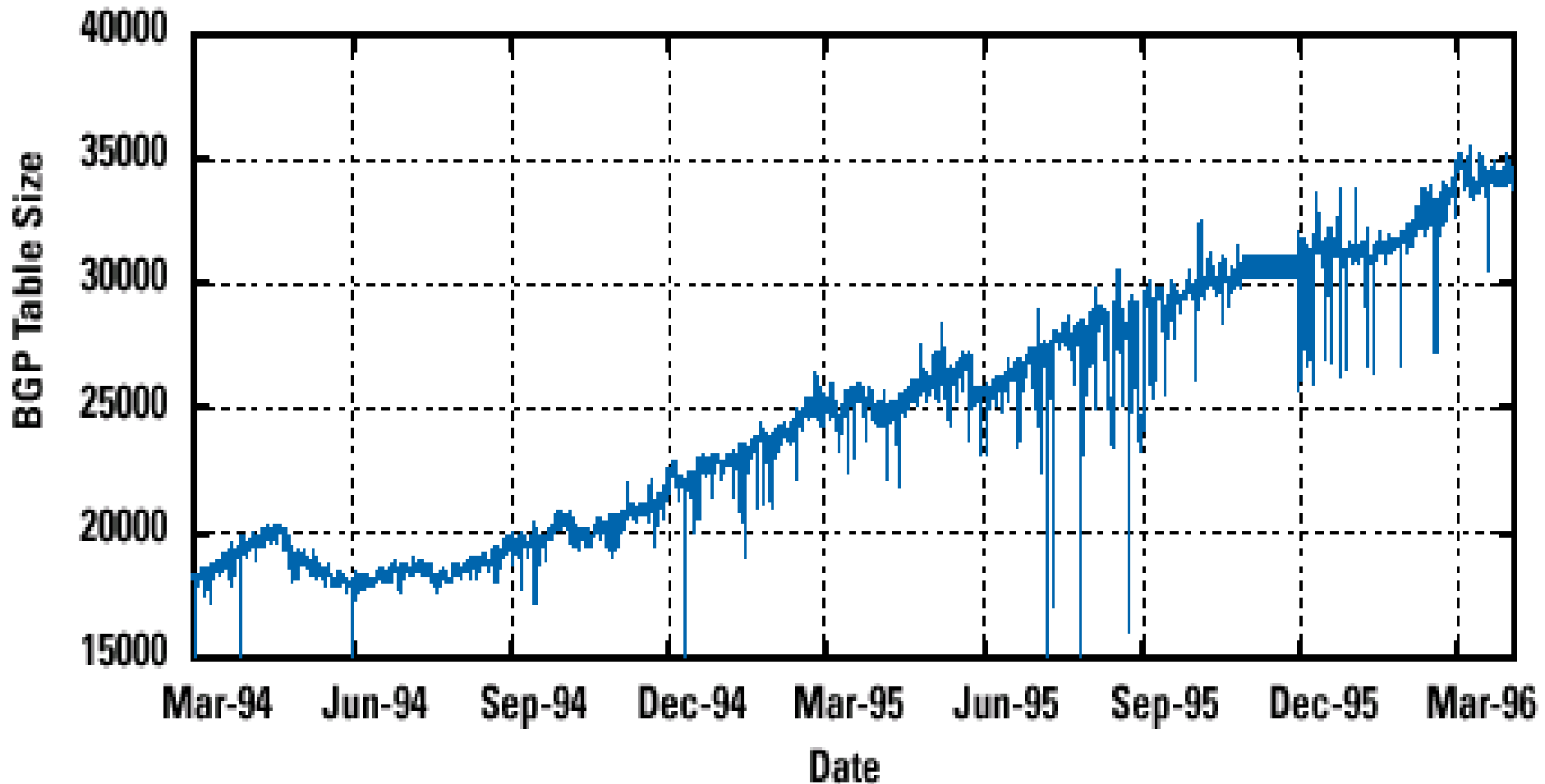
- Hierarchical addressing
  - Critical for scalable system
  - Don't require everyone to know everyone else
  - Reduces amount of updating when something changes
- Non-uniform hierarchy
  - Useful for heterogeneous networks of different sizes
  - Initial class-based addressing was far too coarse
  - Classless InterDomain Routing (CIDR) helps
- Next few slides
  - History of the number of globally-visible prefixes
  - Plots are # of prefixes vs. time

## Pre-CIDR (1988-1994): Steep Growth



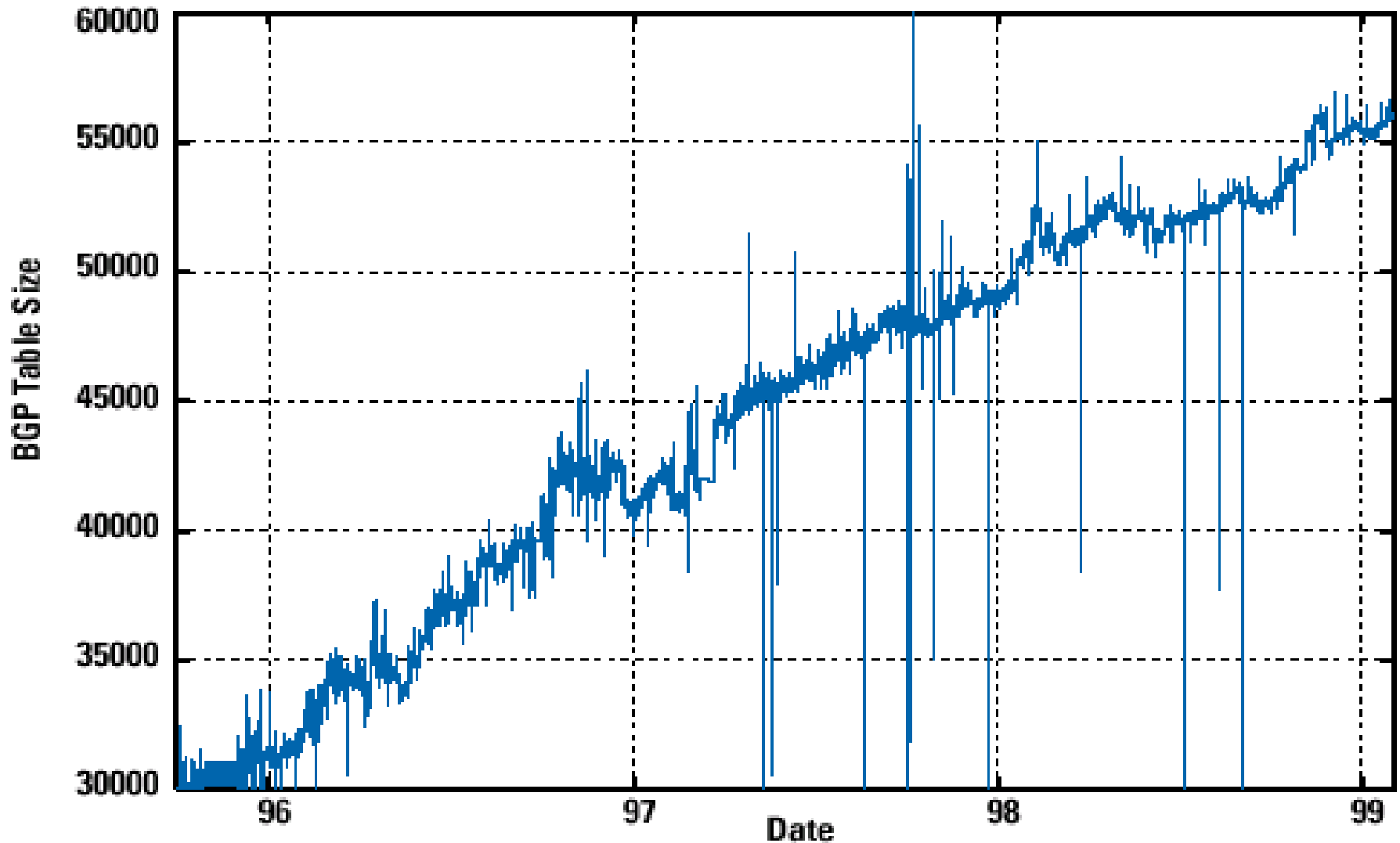
Growth faster than improvements in equipment capability

## CIDR Deployed (1994-1996): Much Flatter



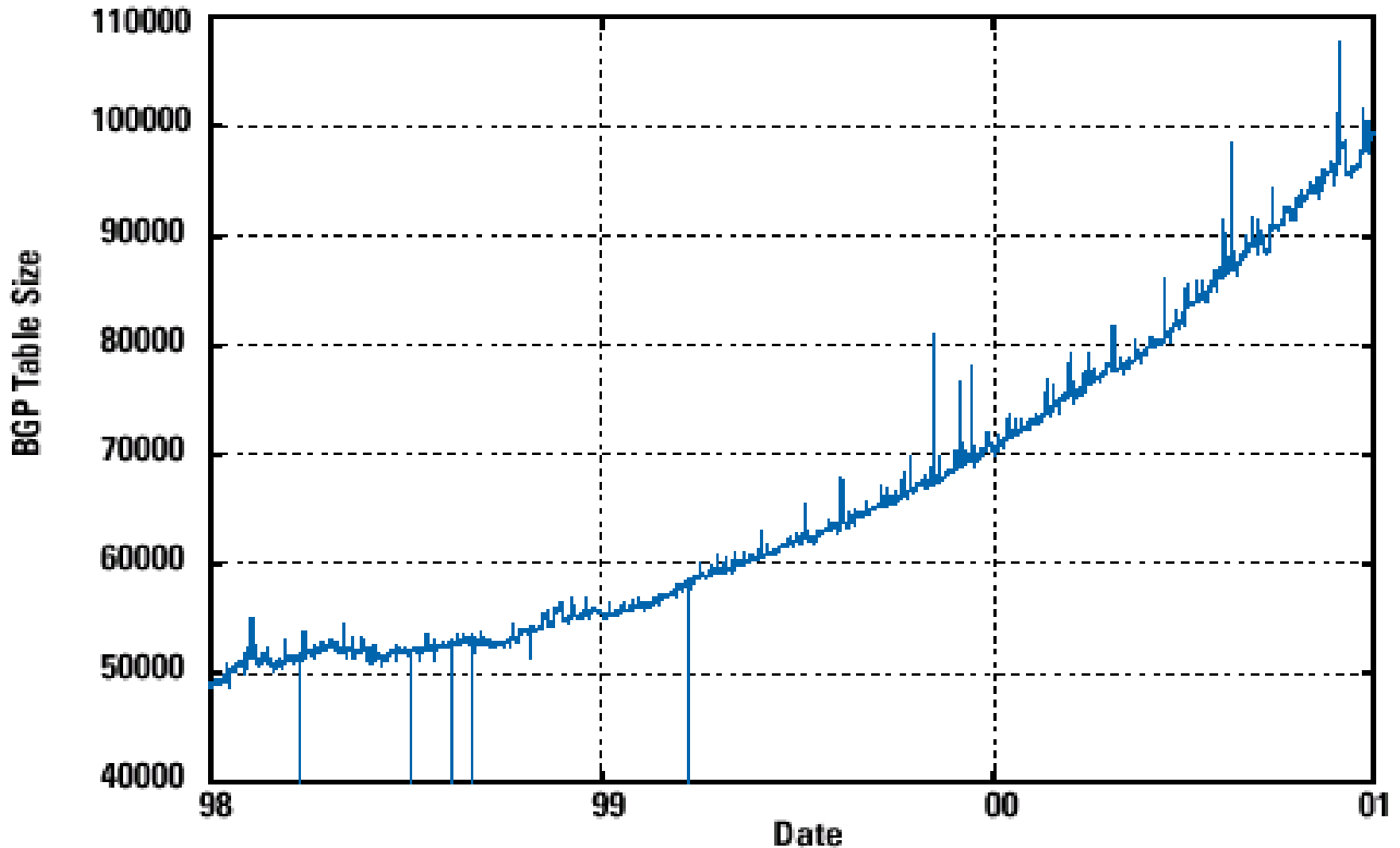
Efforts to aggregate (even decreases after IETF meetings!)

# CIDR Growth (1996-1998): Roughly Linear



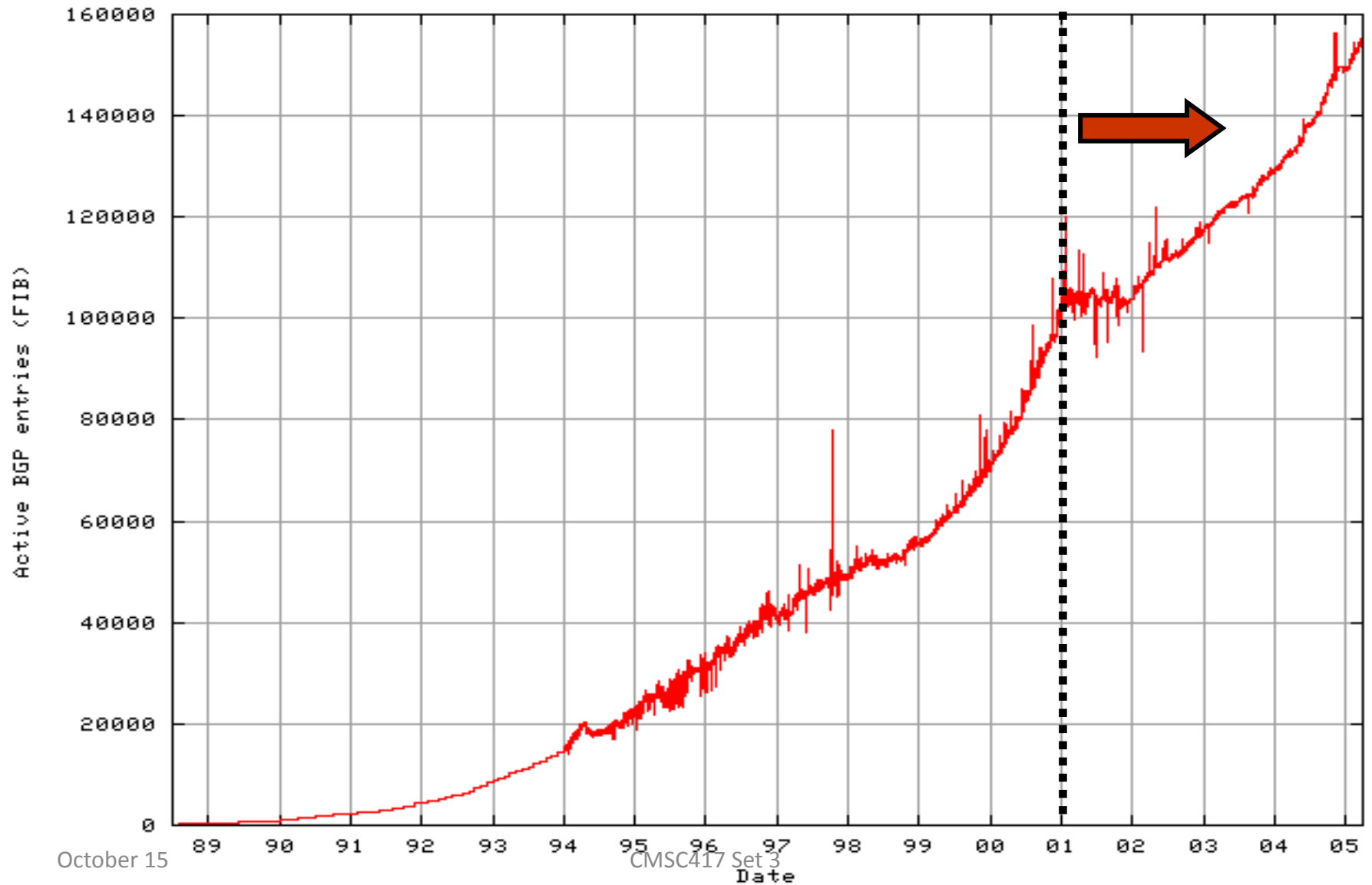
Good use of aggregation, and peer pressure in CIDR report

# Boom Period (1998-2001): Steep Growth



Internet boom and increased multi-homing

# Long-Term View (1989-2005): Post-Boom



# Obtaining a Block of Addresses

- Separation of control
  - Prefix: assigned *to* an institution
  - Addresses: assigned *by* the institution to their nodes
- Who assigns prefixes?
  - Internet Corporation for Assigned Names and Numbers
    - Allocates large address blocks to Regional Internet Registries
  - Regional Internet Registries (RIRs)
    - E.g., ARIN (American Registry for Internet Numbers)
    - Allocates address blocks within their regions
    - Allocated to Internet Service Providers and large institutions
  - Internet Service Providers (ISPs)
    - Allocate address blocks to their customers
    - Who may, in turn, allocate to their customers...

# Figuring Out Who Owns an Address

- Address registries
  - Public record of address allocations
  - Internet Service Providers (ISPs) should update when giving addresses to customers
  - However, records are notoriously out-of-date
- Ways to query
  - UNIX: “whois -h whois.arin.net 128.8.130.75”
  - <http://www.arin.net/whois/>
  - <http://www.geektools.com/whois.php>
  - ...

# Example Output for 128.8.130.75

OrgName: University of Maryland  
OrgID: [UNIVER-262](#)  
Address: Office of Information Technology  
Address: Patuxent Building  
City: College Park  
StateProv: MD PostalCode: 20742  
Country: US

NetRange: [128.8.0.0](#) - [128.8.255.255](#)  
CIDR: 128.8.0.0/16  
NetName: [UMDNET](#)  
NetHandle: [NET-128-8-0-0-1](#)  
Parent: [NET-128-0-0-0-0](#)  
NetType: Direct Assignment  
NameServer: NOC.UMD.EDU  
NameServer: NS1.UMD.EDU  
NameServer: NS2.UMD.EDU  
NameServer: NASANS4.NASA.GOV  
Comment:  
RegDate:  
Updated: 2004-04-12

RTechHandle: [UM-ORG-ARIN](#)  
RTechName: UMD DNS Admin Role Account  
RTechPhone: +1-301-405-3003  
RTechEmail: dnsadmin@noc.net.umd.edu

OrgAbuseHandle: [UARA-ARIN](#)  
OrgAbuseName: UMD Abuse Role Account  
OrgAbusePhone: +1-301-405-8787  
OrgAbuseEmail: abuse@umd.edu

OrgTechHandle: [UM-ORG-ARIN](#)  
OrgTechName: UMD DNS Admin Role Account  
OrgTechPhone: +1-301-405-3003  
OrgTechEmail: dnsadmin@noc.net.umd.edu

# Are 32-bit Addresses Enough?

- Not all that many unique addresses
  - $2^{32} = 4,294,967,296$  (just over four billion)
  - Plus, some are reserved for special purposes
  - And, addresses are allocated in larger blocks
- And, many devices need IP addresses
  - Computers, PDAs, routers, tanks, toasters, ...
- Long-term solution: a larger address space
  - IPv6 has 128-bit addresses ( $2^{128} = 3.403 \times 10^{38}$ )
- Short-term solutions: limping along with IPv4
  - Private addresses
  - Network address translation (NAT)
  - Dynamically-assigned addresses (DHCP)

# Hard Policy Questions

- How much address space per geographic region?
  - Equal amount per country?
  - Proportional to the population?
  - What about addresses already allocated?
- Address space portability?
  - Keep your address block when you change providers?
  - Pro: avoid having to renumber your equipment
  - Con: reduces the effectiveness of address aggregation
- Keeping the address registries up to date?
  - What about mergers and acquisitions?
  - Delegation of address blocks to customers?
  - As a result, the registries are horribly out of date

# Packet Forwarding

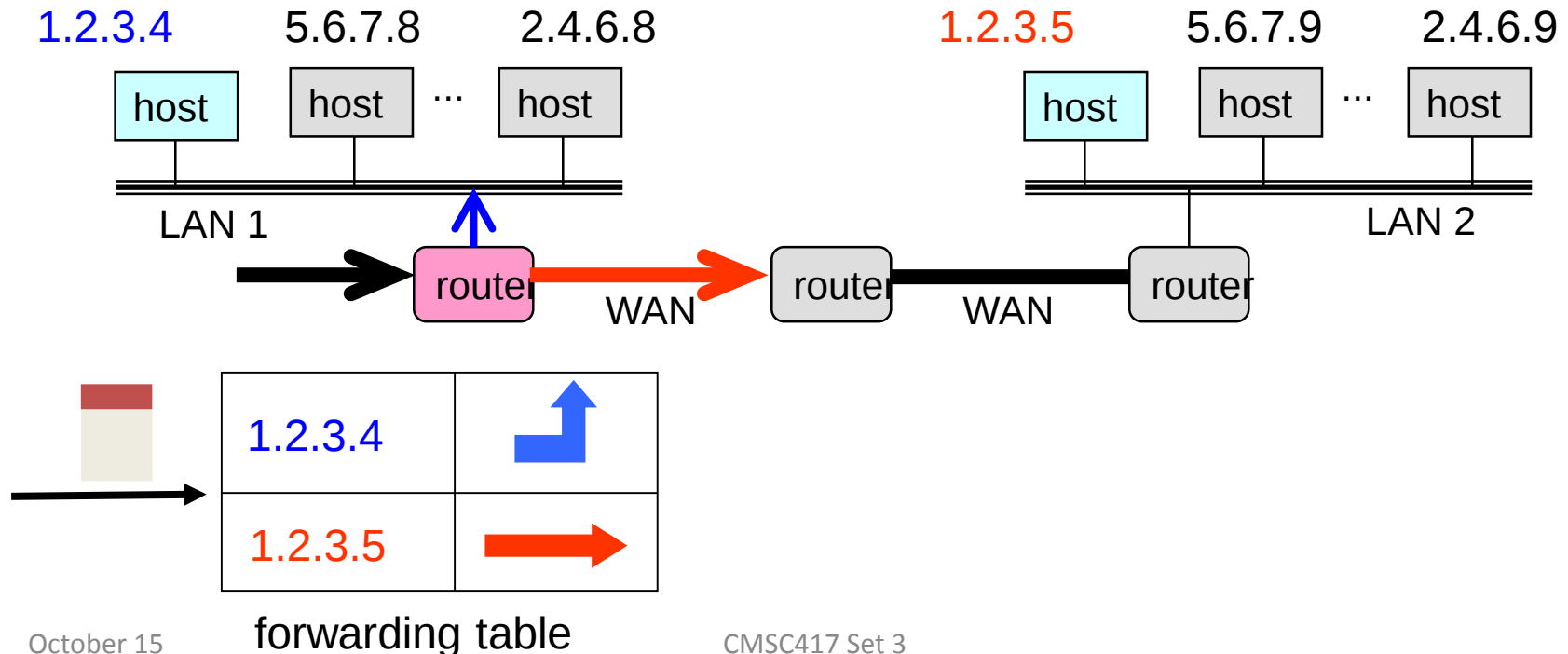
# Hop-by-Hop Packet Forwarding

- Each router has a forwarding table
  - Maps destination addresses...
  - ... to outgoing interfaces
- Upon receiving a packet
  - Inspect the destination IP address in the header
  - Index into the table
  - Determine the outgoing interface
  - Forward the packet out that interface
- Then, the next router in the path repeats
  - And the packet travels along the path to the destination



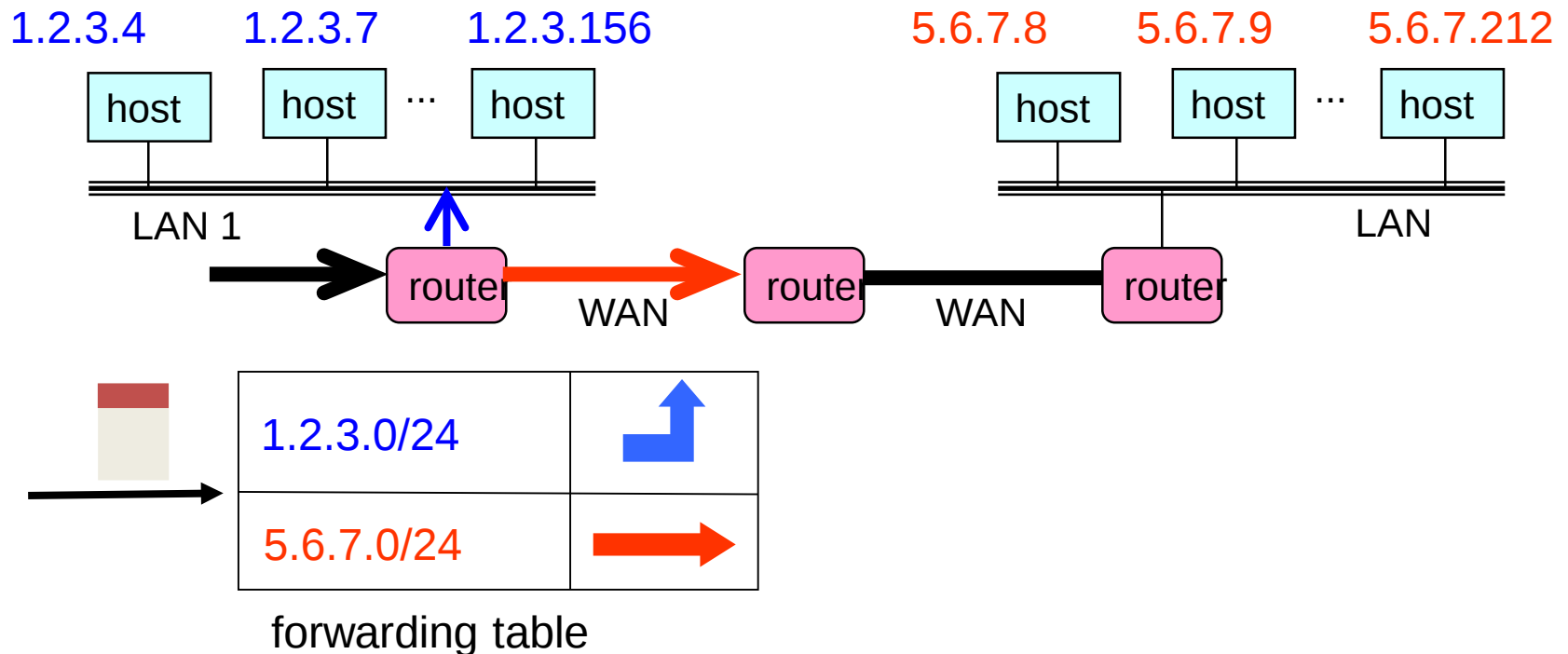
# Separate Table Entries Per Address

- If a router had a forwarding entry per IP address
  - Match *destination address* of incoming packet
  - ... to the *forwarding-table entry*
  - ... to determine the *outgoing interface*



# Separate Entry Per 24-bit Prefix

- If the router had an entry per 24-bit prefix
  - Look only at the top 24 bits of the destination address
  - Index into the table to determine the next-hop interface

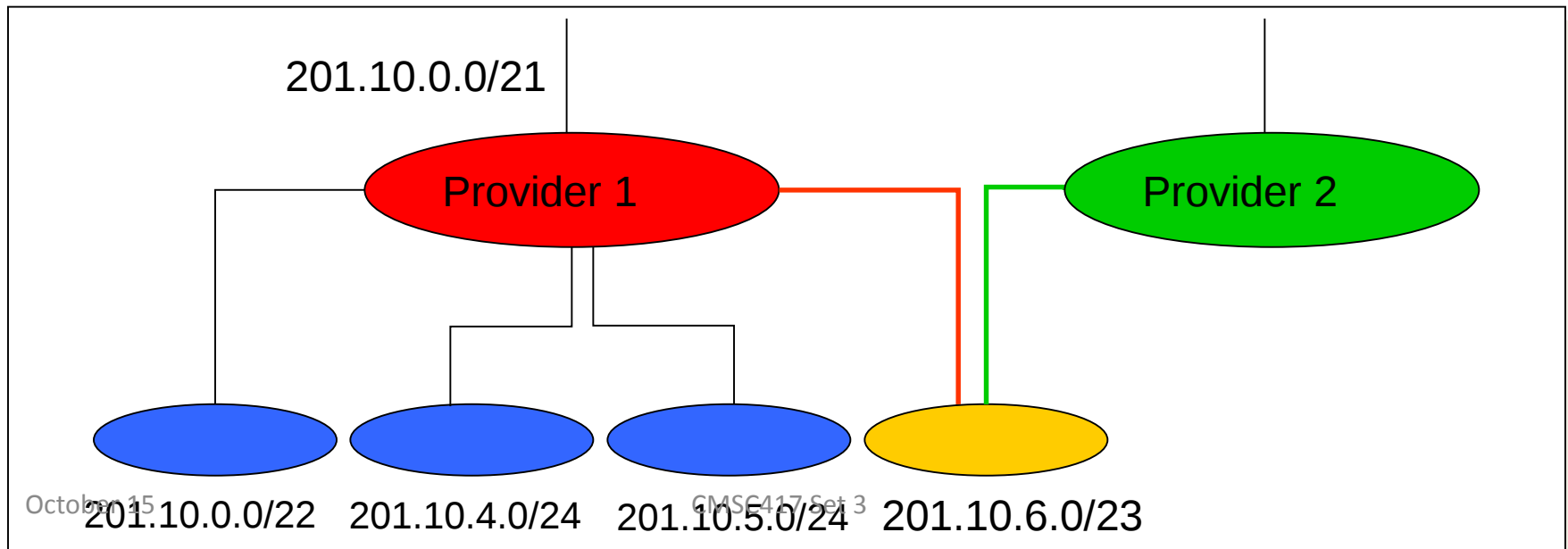


# Separate Entry Classful Address

- If the router had an entry per classful prefix
  - Mixture of Class A, B, and C addresses
  - Depends on the first couple of bits of the destination
- Identify the mask automatically from the address
  - First bit of 0: class A address (/8)
  - First two bits of 10: class B address (/16)
  - First three bits of 110: class C address (/24)
- Then, look in the forwarding table for the match
  - E.g., 1.2.3.4 maps to 1.2.3.0/24
  - Then, look up the entry for 1.2.3.0/24
  - ... to identify the outgoing interface

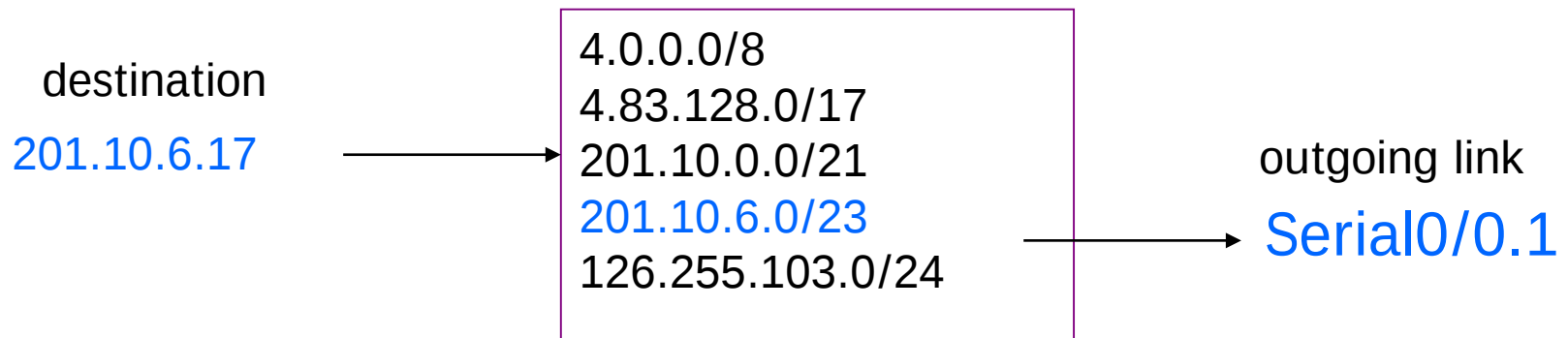
# CIDR Makes Packet Forwarding Harder

- There's no such thing as a free lunch
  - CIDR allows efficient use of the limited address space
  - But, CIDR makes packet forwarding much harder
- Forwarding table may have many matches
  - E.g., table entries for 201.10.0.0/21 and 201.10.6.0/23



# Longest Prefix Match Forwarding

- Forwarding tables in IP routers
    - Maps each IP prefix to next-hop link(s)
  - Destination-based forwarding
    - Packet has a destination address
    - Router identifies longest-matching prefix
    - Cute algorithmic problem: very fast lookups
- forwarding table

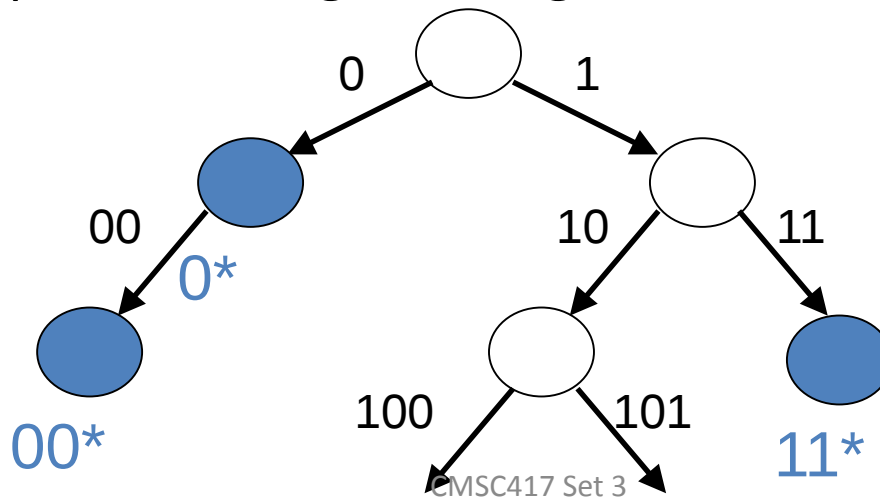


# Simplest Algorithm is Too Slow

- Scan the forwarding table one entry at a time
  - See if the destination matches the entry
  - If so, check the size of the mask for the prefix
  - Keep track of the entry with longest-matching prefix
- Overhead is linear in size of the forwarding table
  - Today, that means 150,000-200,000 entries!
  - And, the router may have just a few nanoseconds
  - ... before the next packet is arriving
- Need greater efficiency to keep up with *line rate*
  - Better algorithms
  - Hardware implementations

# Patricia Tree

- Store the prefixes as a tree
  - One bit for each level of the tree
  - Some nodes correspond to valid prefixes
  - ... which have next-hop interfaces in a table
- When a packet arrives
  - Traverse the tree based on the destination address
  - Stop upon reaching the longest matching prefix



# Even Faster Lookups

- Patricia tree is faster than linear scan
  - Proportional to number of bits in the address
- Patricia tree can be made faster
  - Can make a k-ary tree
    - E.g., 4-ary tree with four children (00, 01, 10, and 11)
  - Faster lookup, though requires more space
- Can use special hardware
  - Content Addressable Memories (CAMs)
  - Allows look-ups on a key rather than flat address
- Huge innovations in the mid-to-late 1990s
  - After CIDR was introduced (in 1994)
  - ... and longest-prefix match was a major bottleneck

# Where do Forwarding Tables Come From?

- Routers have forwarding tables
  - Map prefix to outgoing link(s)
- Entries can be statically configured
  - E.g., “map 12.34.158.0/24 to Serial0/0.1”
- But, this doesn’t adapt
  - To failures
  - To new equipment
  - To the need to balance load
  - ...
- That is where other technologies come in...
  - Routing protocols, DHCP, and ARP (later in course)

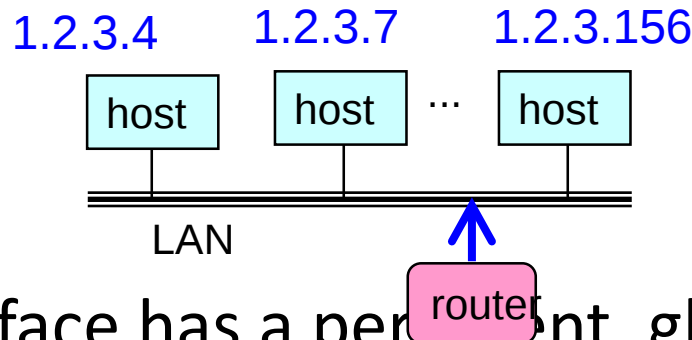
# What End Hosts Sending to Others?

- End host with single network interface
  - PC with an Ethernet link
  - Laptop with a wireless link
- Don't need to run a routing protocol
  - Packets to the host itself (e.g., 1.2.3.4/32)
    - Delivered locally
  - Packets to other hosts on the LAN (e.g., 1.2.3.0/24)
    - Sent out the interface
  - Packets to external hosts (e.g., 0.0.0.0/0)
    - Sent out interface to local gateway
- How this information is learned
  - Static setting of address, subnet mask, and gateway
  - Dynamic Host Configuration Protocol (DHCP)



# What About Reaching the End Hosts?

- How does the last router reach the destination?



- Each interface has a permanent, global identifier
  - MAC (Media Access Control) address
  - Burned in to the adaptors Read-Only Memory (ROM)
  - Flat address structure (i.e., no hierarchy)
- Constructing an address resolution table
  - Mapping MAC address to/from IP address
  - Address Resolution Protocol (ARP)

# Conclusions

- IP address
  - A 32-bit number
  - Allocated in prefixes
  - Non-uniform hierarchy for scalability and flexibility
- Packet forwarding
  - Based on IP prefixes
  - Longest-prefix-match forwarding
- We'll cover some topics later
  - Routing protocols, DHCP, and ARP

# Routing

# Routing Algorithms

- The Optimality Principle
- Shortest Path Routing
- Flooding
- Distance Vector Routing
- Link State Routing
- Hierarchical Routing
- Broadcast Routing
- Multicast Routing
- Routing for Mobile Hosts
- Routing in Ad Hoc Networks

# What is Routing?

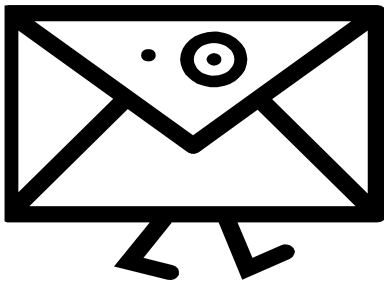
- A famous quotation from RFC 791

*“A name indicates what we seek.*

*An address indicates where it is.*

*A route indicates how we get there”*

-- Jon Postel

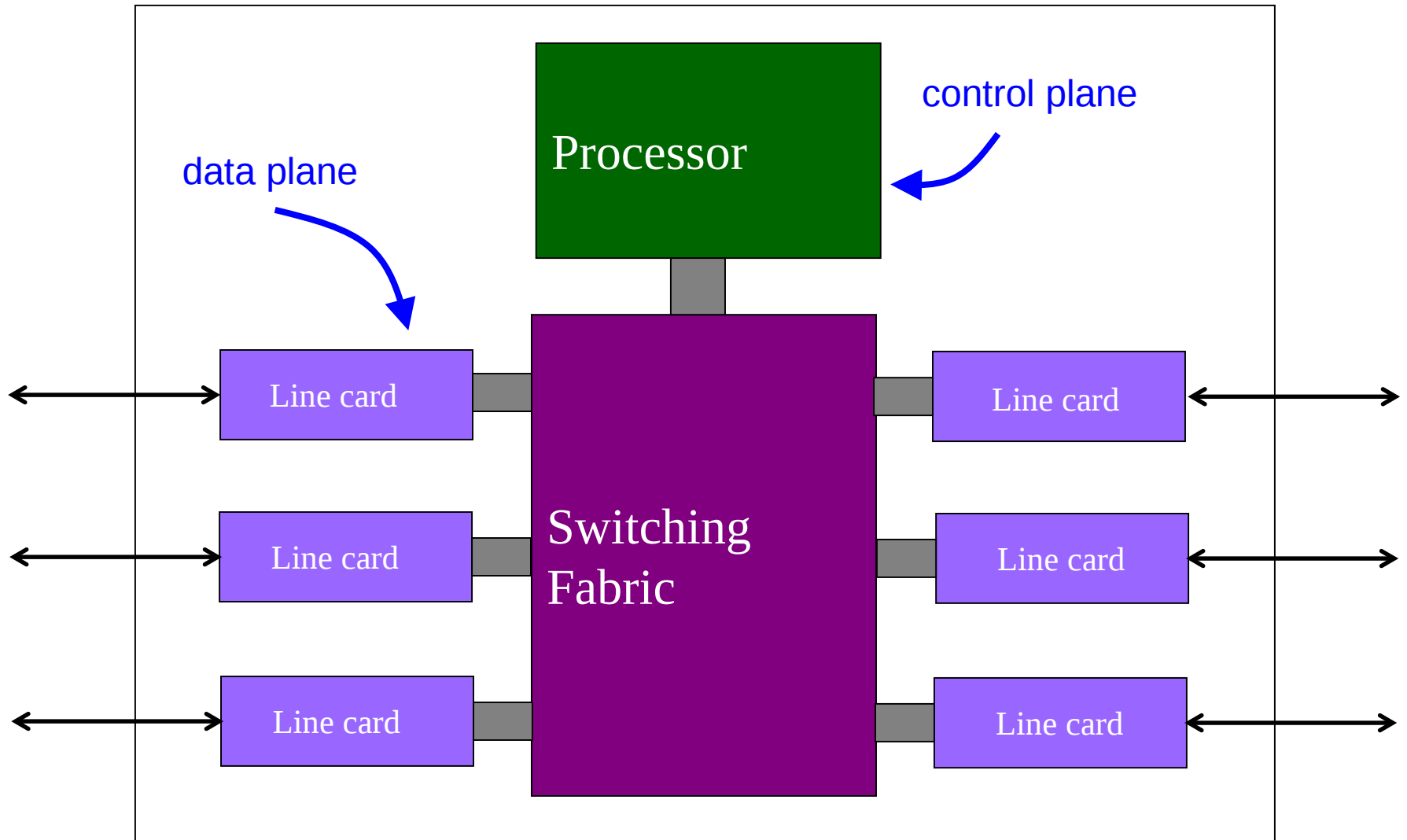


# Routing vs. Forwarding

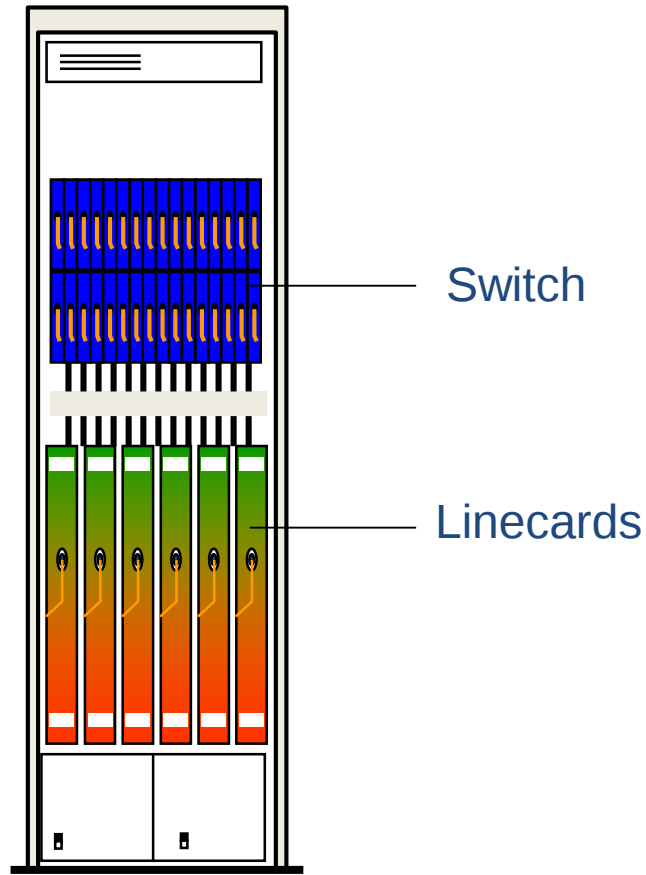
- Routing: control plane
  - Computing paths the packets will follow
  - Routers talking amongst themselves
  - Individual router *creating* a forwarding table
- Forwarding: data plane
  - Directing a data packet to an outgoing link
  - Individual router *using* a forwarding table



# Data and Control Planes



# Router Physical Layout



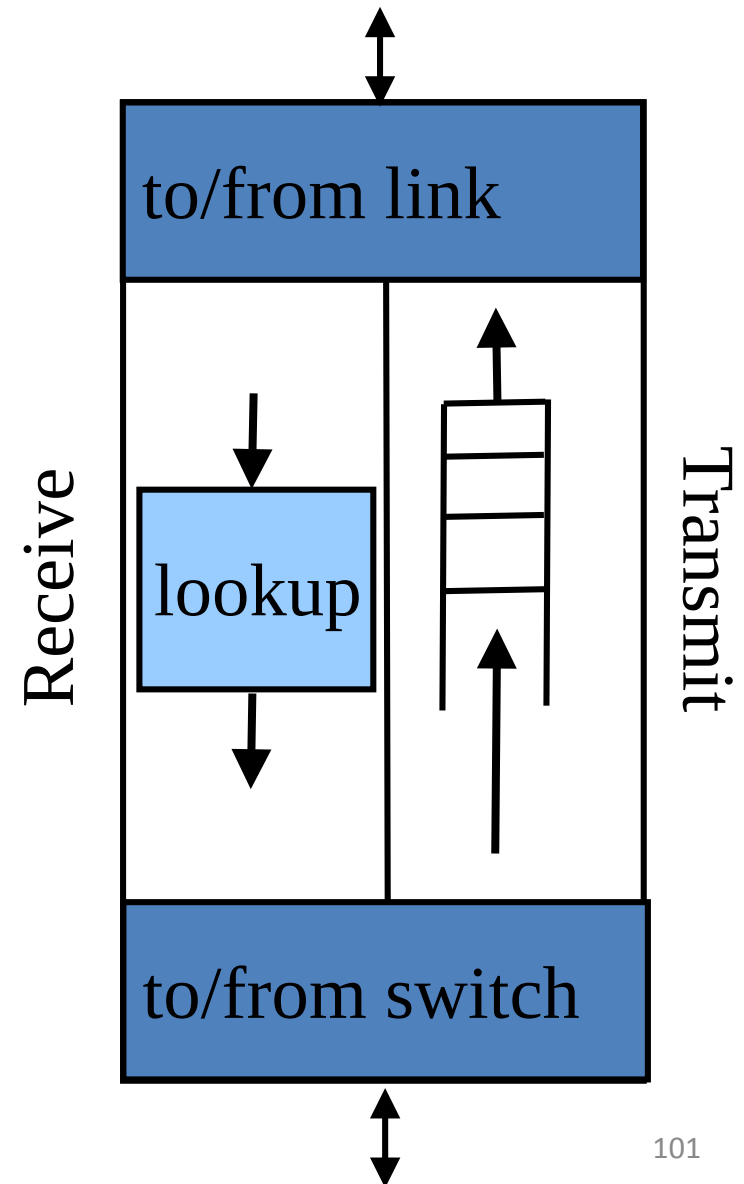
Juniper T series



Cisco 12000

# Line Cards (Interface Cards, Adaptors)

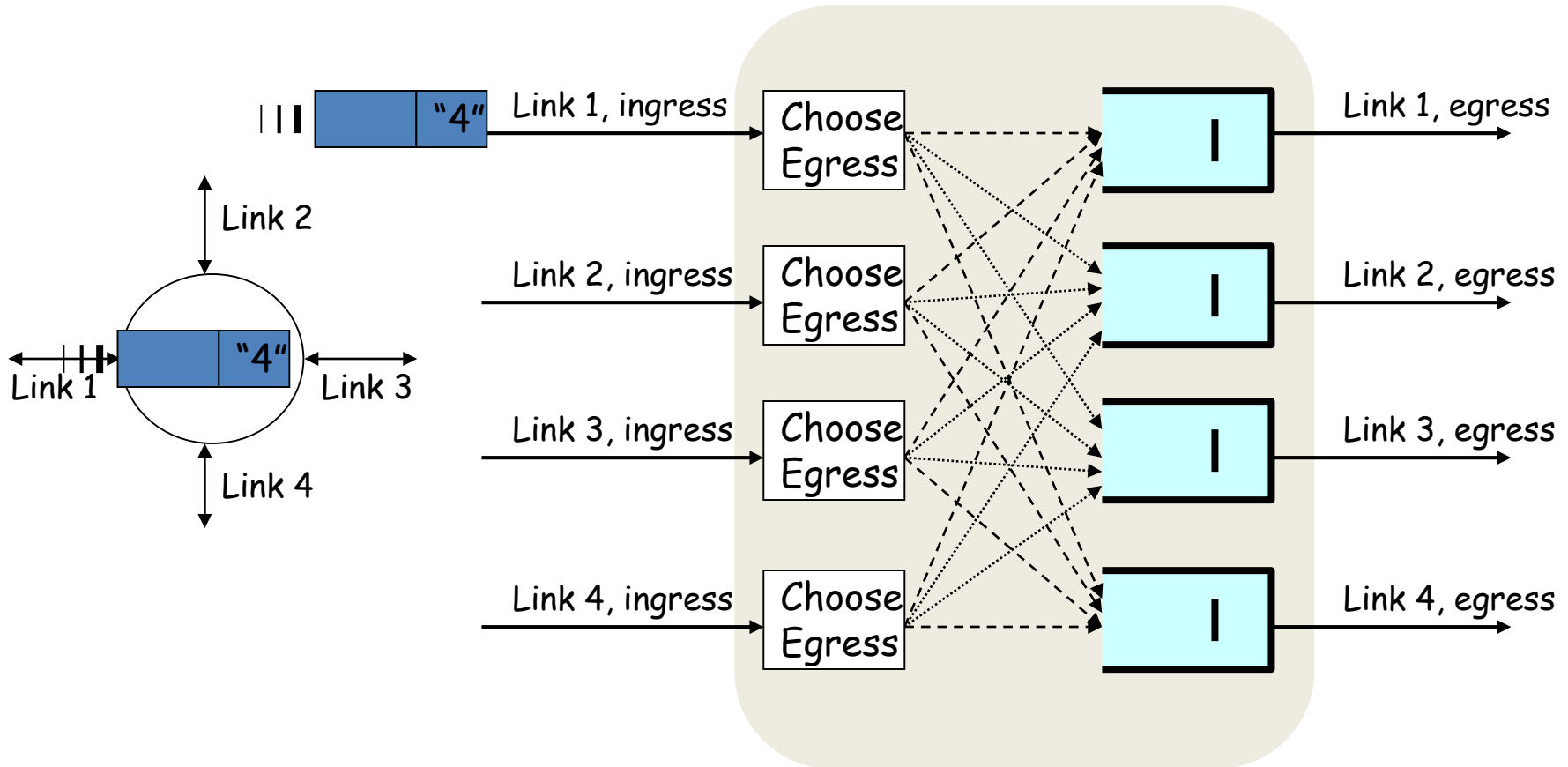
- Interfacing
  - Physical link
  - Switching fabric
- Packet handling
  - Packet forwarding
  - Decrement time-to-live
  - Buffer management
  - Link scheduling
  - Packet filtering
  - Rate limiting
  - Packet marking
  - Measurement



# Switching Fabric

- Deliver packet inside the router
  - From incoming interface to outgoing interface
  - A small network in and of itself
- Must operate very quickly
  - Multiple packets going to same outgoing interface
  - Switch scheduling to match inputs to outputs
- Implementation techniques
  - Bus, crossbar, interconnection network, ...
  - Running at a faster speed (e.g., 2X) than links
  - Dividing variable-length packets into fixed-size cells

# Packet Switching



# Router Processor

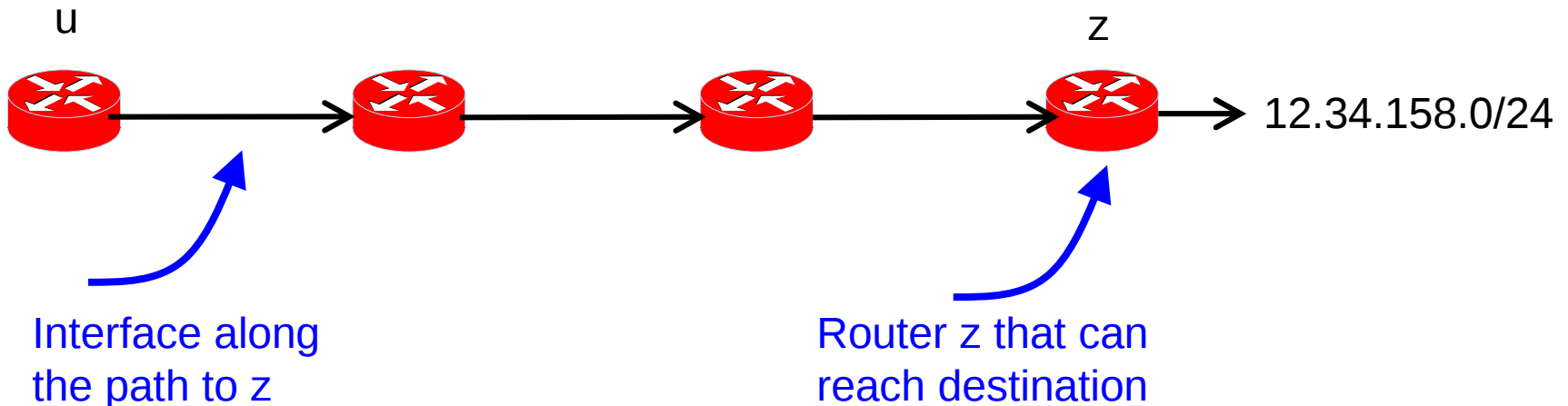
- So-called “Loopback” interface
  - IP address of the CPU on the router
- Interface to network administrators
  - Command-line interface for configuration
  - Transmission of measurement statistics
- Handling of special data packets
  - Packets with IP options enabled
  - Packets with expired Time-To-Live field
- Control-plane software
  - Implementation of the routing protocols
  - Creation of forwarding table for the line cards

## Where do Forwarding Tables Come From?

- Routers have forwarding tables
  - Map IP prefix to outgoing link(s)
- Entries can be statically configured
  - E.g., “map 12.34.158.0/24 to Serial0/0.1”
- But, this doesn’t adapt
  - To failures
  - To new equipment
  - To the need to balance load
- That is where routing protocols come in

# Computing Paths Between Routers

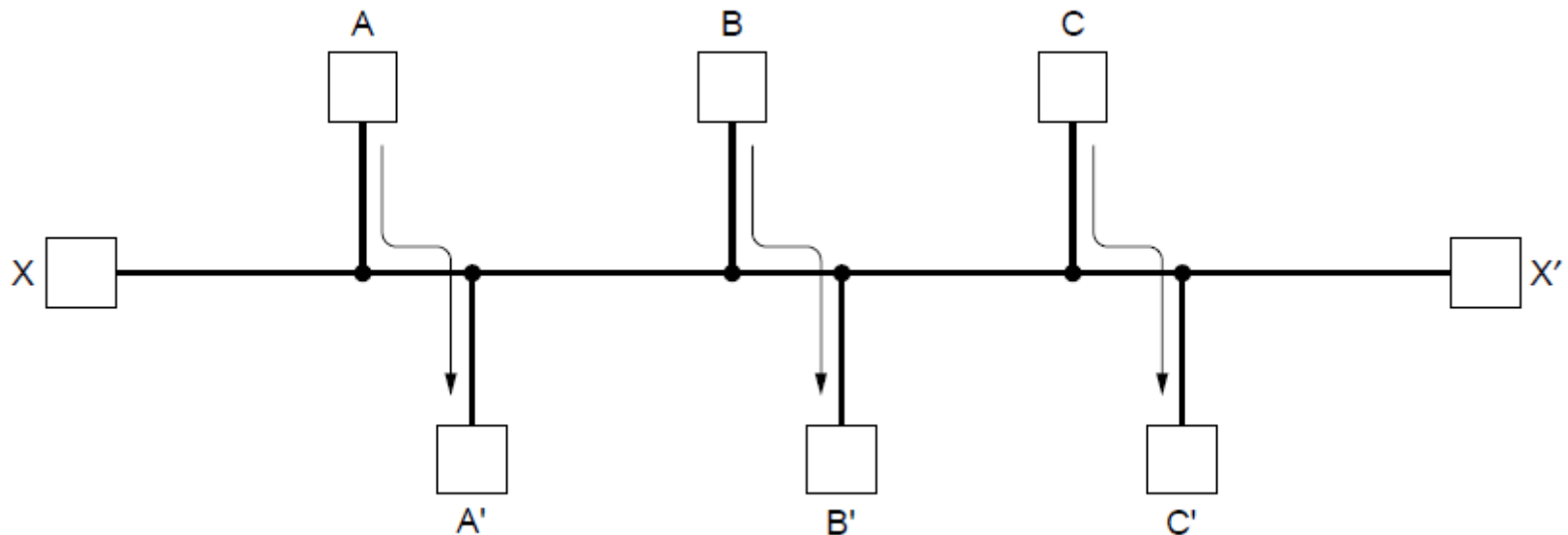
- Routers need to know two things
  - Which router to use to reach a destination prefix
  - Which outgoing interface to use to reach that router



- just how routers reach each other
  - How you knows how to forward packets toward z

# Fairness vs. Efficiency

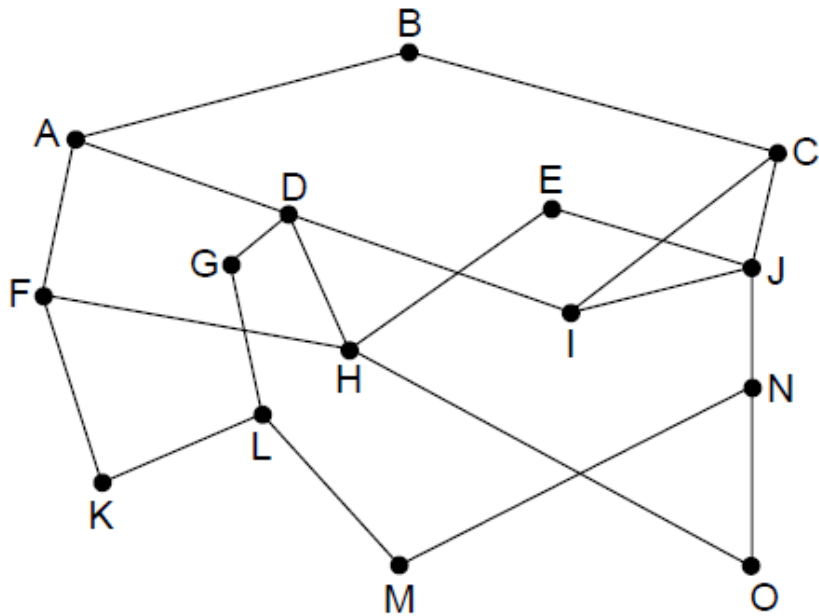
Network with a conflict between fairness and efficiency.



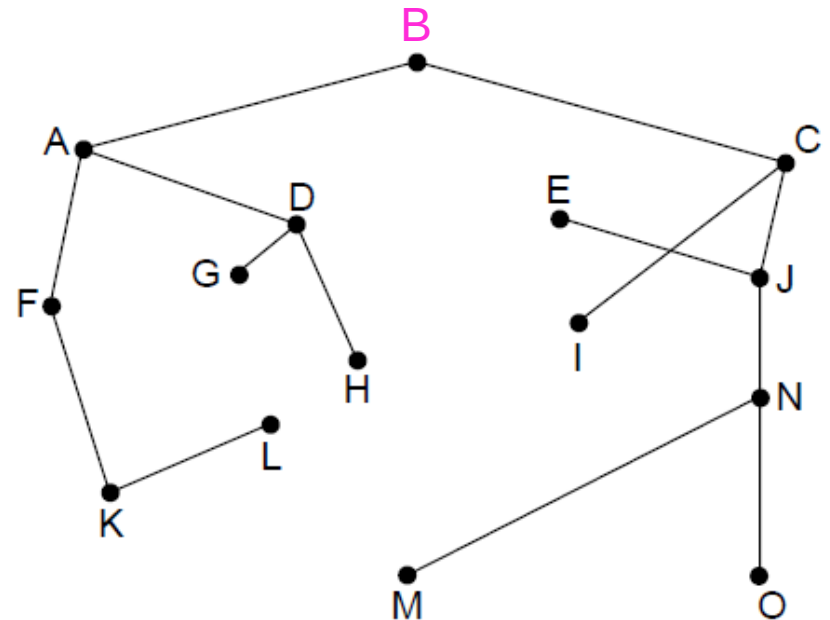


# The Optimality Principle

Each portion of a best path is also a best path;  
the union of them to a router is a tree  
called the sink tree



Network



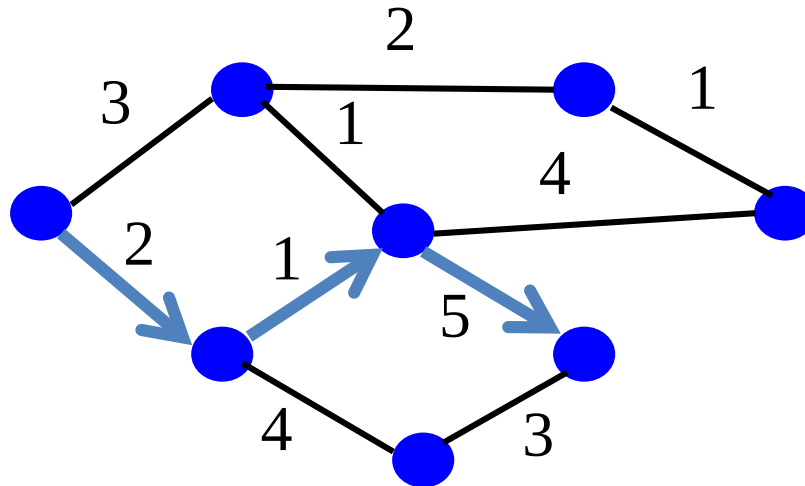
Sink tree of best paths to router B

# Computing the Shortest Paths

(assuming you already know the topology)

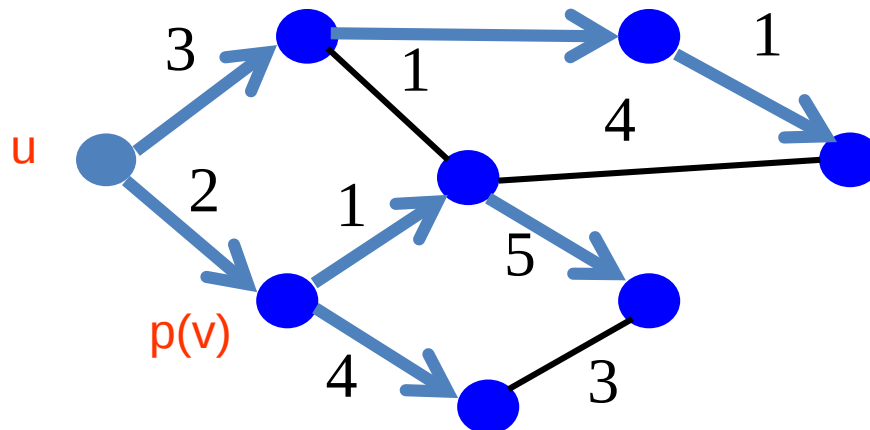
# Shortest-Path Routing

- Path-selection model
  - Destination-based
  - Load-insensitive (e.g., static link weights)
  - Minimum hop count or sum of link weights



# Shortest-Path Problem

- Given: network topology with link costs
  - $c(x,y)$ : link cost from node  $x$  to node  $y$
  - Infinity if  $x$  and  $y$  are not direct neighbors
- Compute: least-cost paths to all nodes
  - From a given source  $u$  to all other nodes
  - $p(v)$ : predecessor node along path from source to  $v$



# Dijkstra's Shortest-Path Algorithm

- Iterative algorithm
  - After  $k$  iterations, know least-cost path to  $k$  nodes
- **S**: nodes whose least-cost path definitively known
  - Initially, **S** = {**u**} where  $u$  is the source node
  - Add one node to  $S$  in each iteration
- **D(v)**: current cost of path from source to node  $v$ 
  - Initially, **D(v)** = **c(u,v)** for all nodes  $v$  adjacent to  $u$
  - ... and **D(v)** =  $\infty$  for all other nodes  $v$
  - Continually update  $D(v)$  as shorter paths are learned

# Dijkstra's Algorithm

1 *Initialization:*

2  $S = \{u\}$

3 for all nodes  $v$

4 if ( $v$  is adjacent to  $u$ )

5  $D(v) = c(u,v)$

6 else  $D(v) = \infty$

7

8 *Loop*

9 find  $w$  not in  $S$  with the smallest  $D(w)$

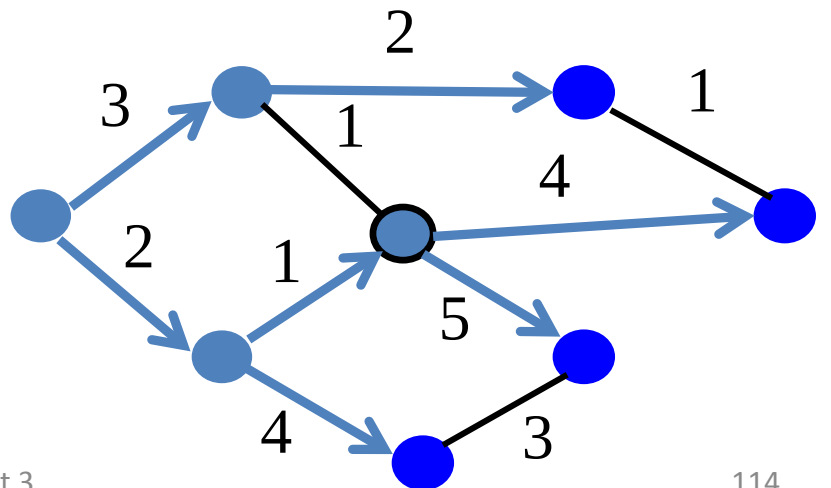
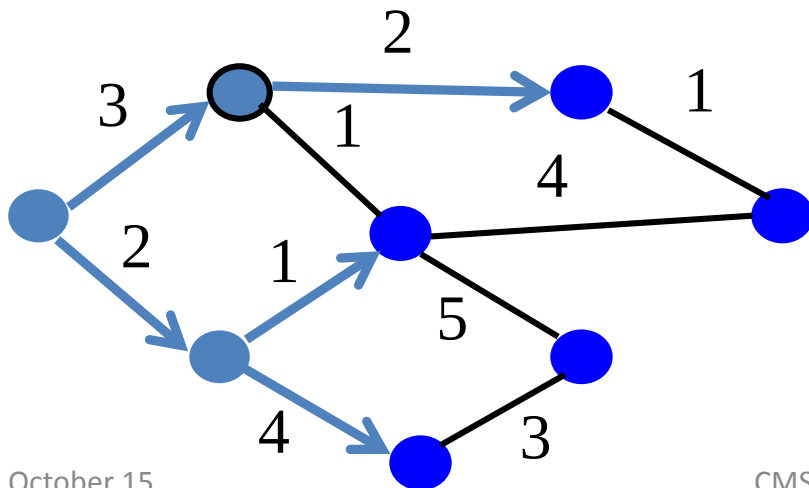
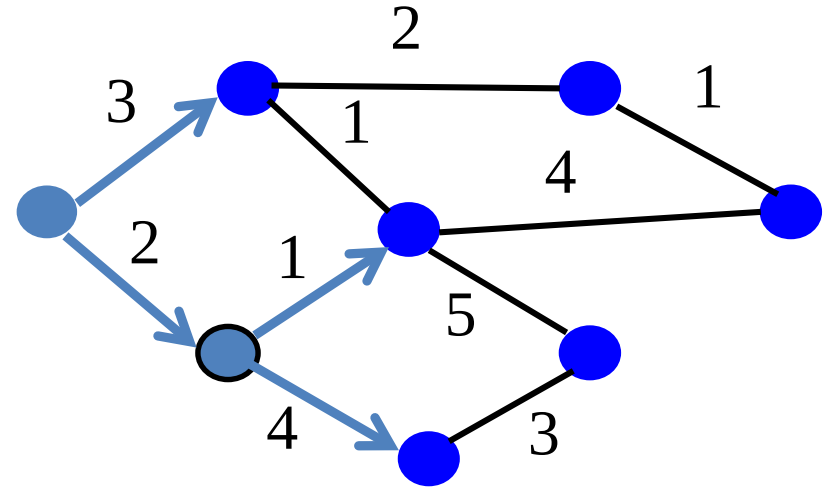
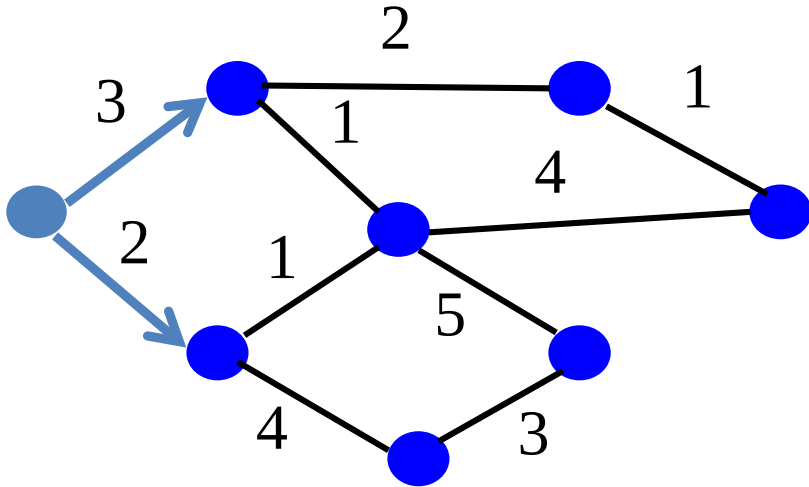
10 add  $w$  to  $S$

11 update  $D(v)$  for all  $v$  adjacent to  $w$  and not in  $S$ :

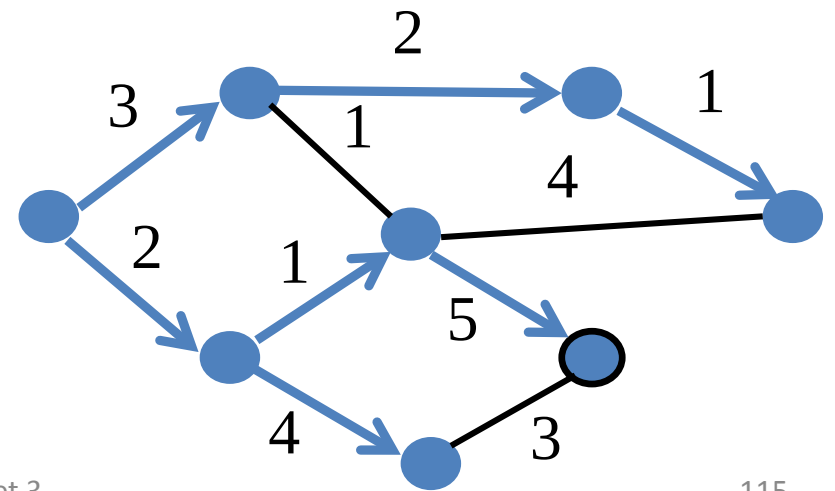
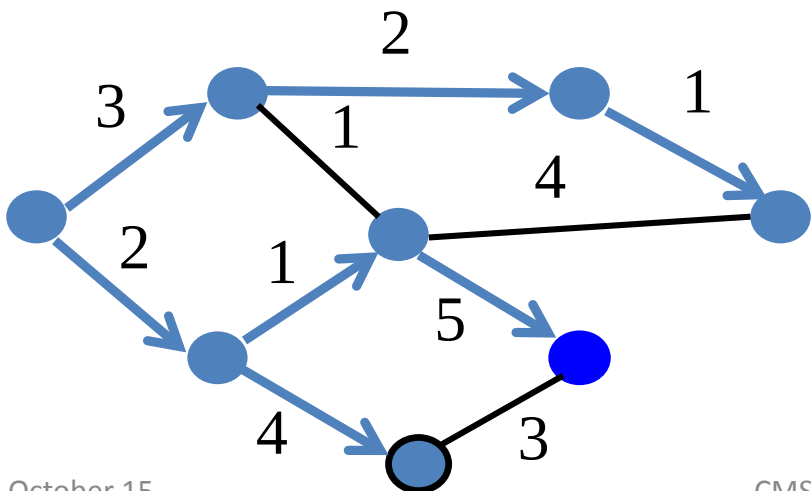
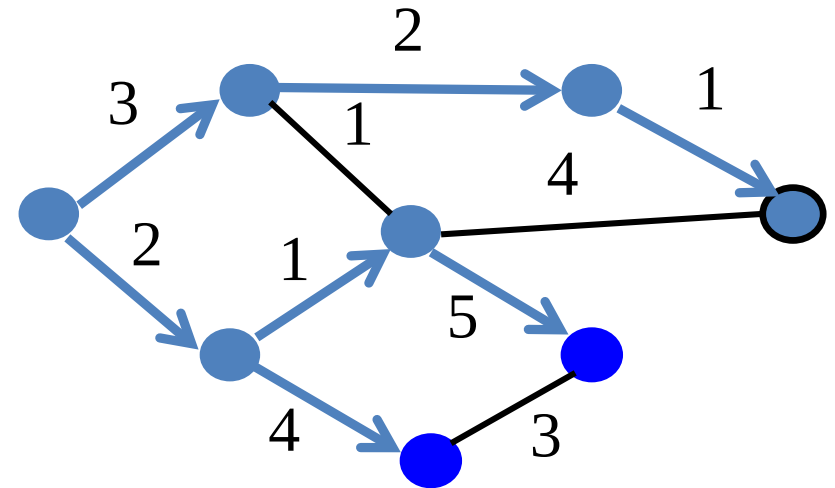
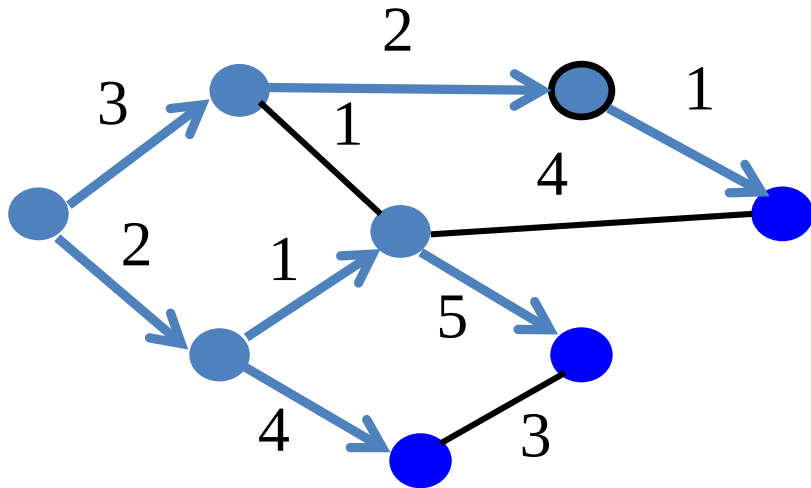
12  $D(v) = \min\{D(v), D(w) + c(w,v)\}$

13 *until all nodes in  $S$*

# Dijkstra's Algorithm Example

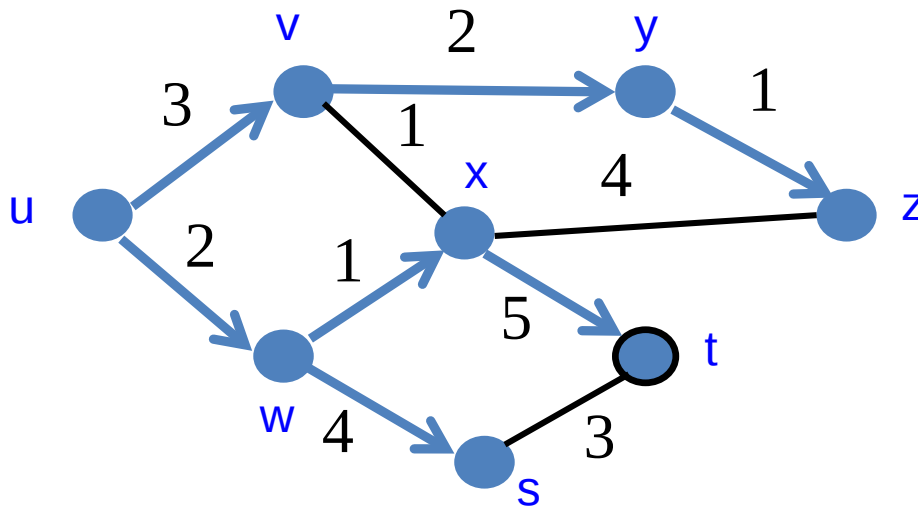


# Dijkstra's Algorithm Example



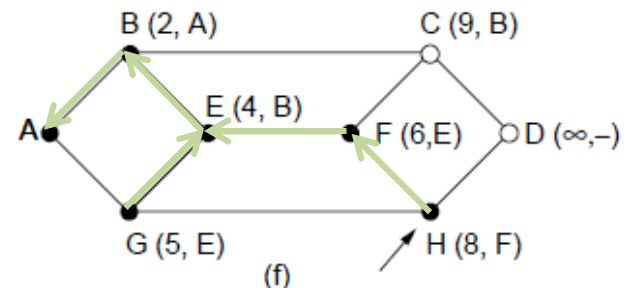
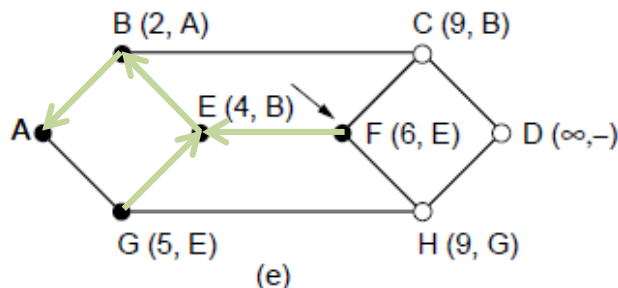
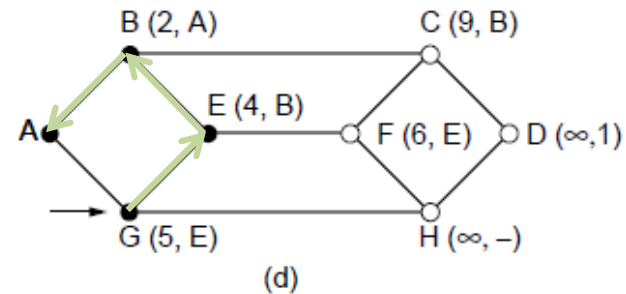
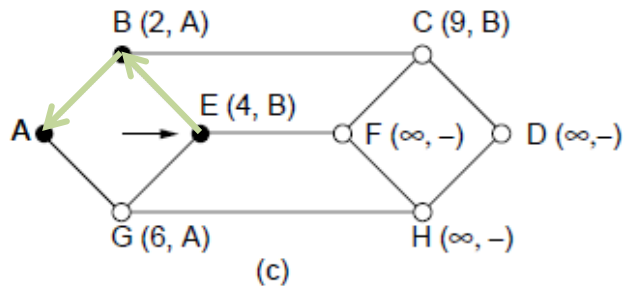
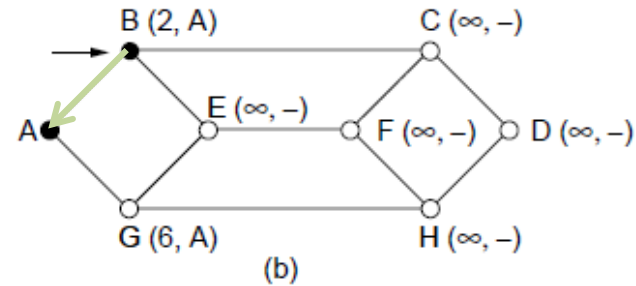
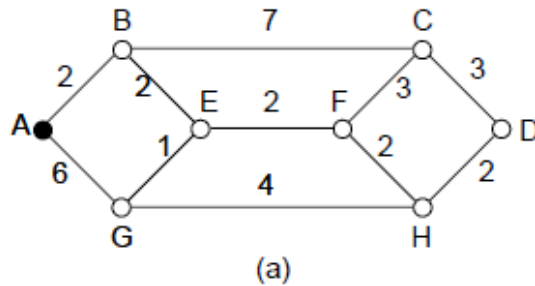
# Shortest-Path Tree

- Shortest-path tree from u
- Forwarding table at u



|   | link  |
|---|-------|
| v | (u,v) |
| w | (u,w) |
| x | (u,w) |
| y | (u,v) |
| z | (u,v) |
| s | (u,w) |
| t | (u,w) |

# Shortest Path Algorithm (2)



A network and first five steps in computing the shortest paths from A to D. Pink arrows show the sink tree so far.

# Shortest path

```
#define MAX_NODES 1024          /* maximum number of nodes */
#define INFINITY 1000000000     /* a number larger than every maximum path */
int n, dist[MAX_NODES][MAX_NODES]; /* dist[i][j] is the distance from i to j */

void shortest_path(int s, int t, int path[])
{ struct state {                /* the path being worked on */
    int predecessor;            /* previous node */
    int length;                 /* length from source to this node */
    enum {permanent, tentative} label; /* label state */
} state[MAX_NODES];

int i, k, min;
struct state *p;

for (p = &state[0]; p < &state[n]; p++) { /* initialize state */
    p->predecessor = -1;
    p->length = INFINITY;
    p->label = tentative;
}
state[t].length = 0; state[t].label = permanent;
k = t;                                /* k is the initial working node */
```

Start with the sink,  
all other nodes are  
unreachable

Dijkstra's algorithm to compute the shortest path through a graph.

# Shortest path (2)

```
do {
    for (i = 0; i < n; i++)
        if (dist[k][i] != 0 && state[i].label == tentative) {
            if (state[k].length + dist[k][i] < state[i].length) {
                state[i].predecessor = k;
                state[i].length = state[k].length + dist[k][i];
            }
        }
}
```

Relaxation step.  
Lower distance to  
nodes linked to  
newest member of  
the sink tree

```
/* Find the tentatively labeled node with the smallest label. */
k = 0; min = INFINITY;
for (i = 0; i < n; i++)
    if (state[i].label == tentative && state[i].length < min) {
        min = state[i].length;
        k = i;
    }
state[k].label = permanent;
} while (k != s);
```

Find the lowest  
distance, add it to  
the sink tree, and  
repeat until done

```
/* Copy the path into the output array. */
i = 0; k = s;
do {path[i++] = k; k = state[k].predecessor; } while (k >= 0);
```

└─ Insert a algorithm to compute the shortest path through a graph.

# Learning the Topology

(by the routers talking among themselves)

# Link-State Routing

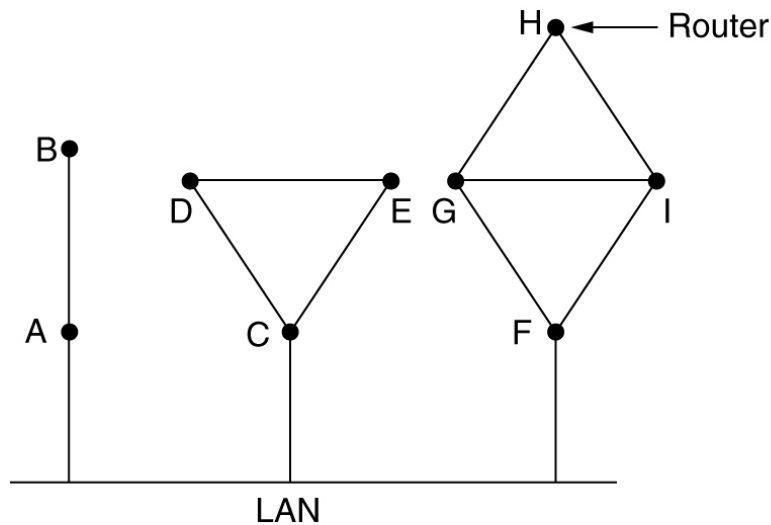
- Each router keeps track of its incident links
  - Whether the link is up or down
  - The cost on the link
- Each router broadcasts the link state
  - To give every router a complete view of the graph
- Each router runs Dijkstra's algorithm
  - To compute the shortest paths
  - ... and construct the forwarding table
- Example protocols
  - Open Shortest Path First (OSPF)
  - Intermediate System – Intermediate System (IS-IS)

# Link State Routing

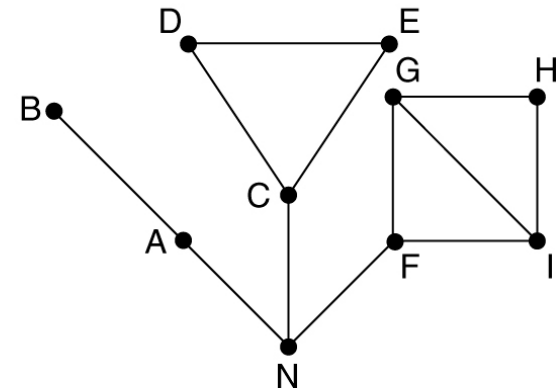
Each router must do the following:

1. Discover its neighbors, learn their network address.
2. Measure the delay or cost to each of its neighbors.
3. Construct a packet telling all it has just learned.
4. Send this packet to all other routers.
5. Compute the shortest path to every other router.

# Learning about the Neighbors



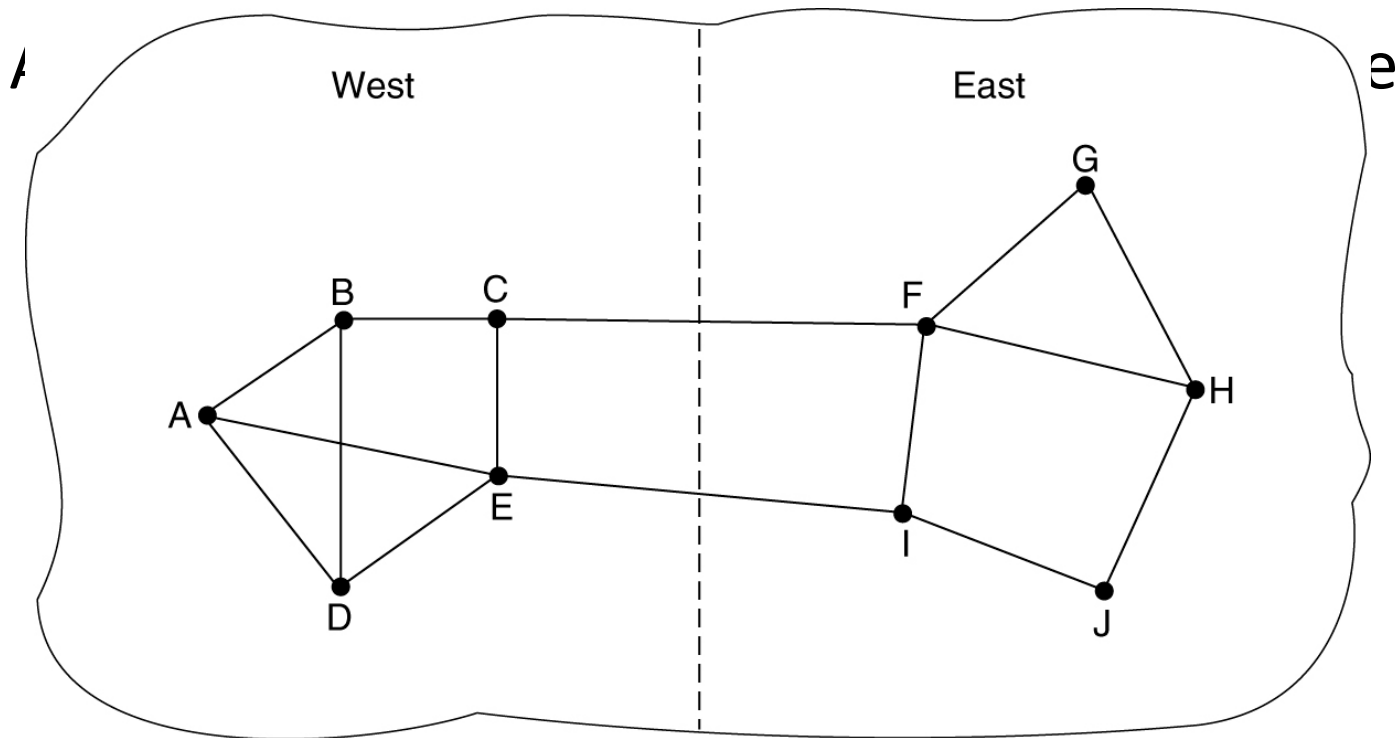
(a)



(b)

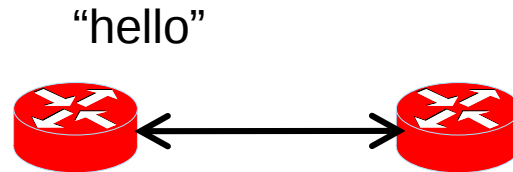
(a) Nine routers and a LAN. (b) A graph model of (a).

# Measuring Line Cost



# Detecting Topology Changes

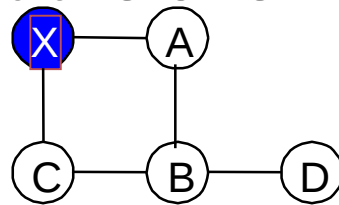
- Beaconing
  - Periodic “hello” messages in both directions
  - Detect a failure after a few missed “hellos”



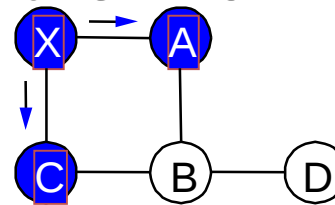
- Performance trade-offs
  - Detection speed
  - Overhead on link bandwidth and CPU
  - Likelihood of false detection

# Broadcasting the Link State

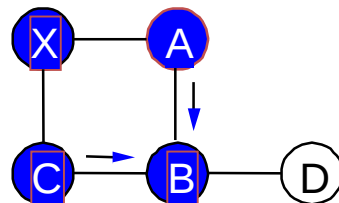
- Flooding
  - Node sends link-state information out its links
  - And then the next node sends out all of its links
  - ... except the one where the information arrived



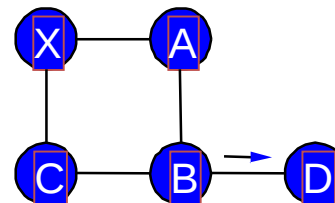
(a)



(b)



(c)

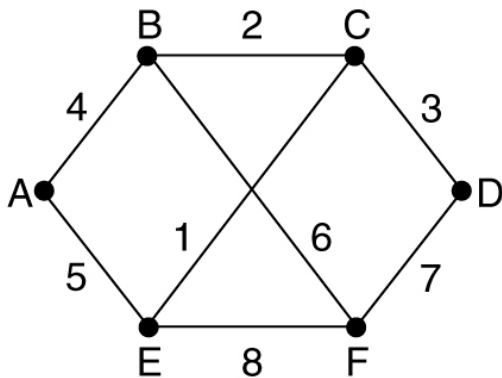


(d)

# Broadcasting the Link State

- Reliable flooding
  - Ensure all nodes receive link-state information
  - ... and that they use the latest version
- Challenges
  - Packet loss
  - Out-of-order arrival
- Solutions
  - Acknowledgments and retransmissions
  - Sequence numbers
  - Time-to-live for each packet

# Building Link State Packets



(a)

|      |   | Link |   | State |   | Packets |   |
|------|---|------|---|-------|---|---------|---|
| A    |   | B    |   | C     |   | D       |   |
| Seq. |   | Seq. |   | Seq.  |   | Seq.    |   |
| Age  |   | Age  |   | Age   |   | Age     |   |
| B    | 4 | A    | 4 | B     | 2 | C       | 3 |
| E    | 5 | C    | 2 | D     | 3 | F       | 7 |
|      |   | F    | 6 | E     | 1 |         |   |

| E    |   | F    |   |
|------|---|------|---|
| Seq. |   | Seq. |   |
| Age  |   | Age  |   |
| A    | 5 | B    | 6 |
| C    | 1 | D    | 7 |
| F    | 8 | E    | 8 |

(b)

(a) A subnet. (b) The link state packets for this subnet.

# Distributing the Link State Packets

The packet buffer for router B in the previous slide

| Source | Seq. | Age | Send flags |   |   | ACK flags |   |   | Data |
|--------|------|-----|------------|---|---|-----------|---|---|------|
|        |      |     | A          | C | F | A         | C | F |      |
| A      | 21   | 60  | 0          | 1 | 1 | 1         | 0 | 0 |      |
| F      | 21   | 60  | 1          | 1 | 0 | 0         | 0 | 1 |      |
| E      | 21   | 59  | 0          | 1 | 0 | 1         | 0 | 1 |      |
| C      | 20   | 60  | 1          | 0 | 1 | 0         | 1 | 0 |      |
| D      | 21   | 59  | 1          | 0 | 0 | 0         | 1 | 1 |      |

# When to Initiate Flooding

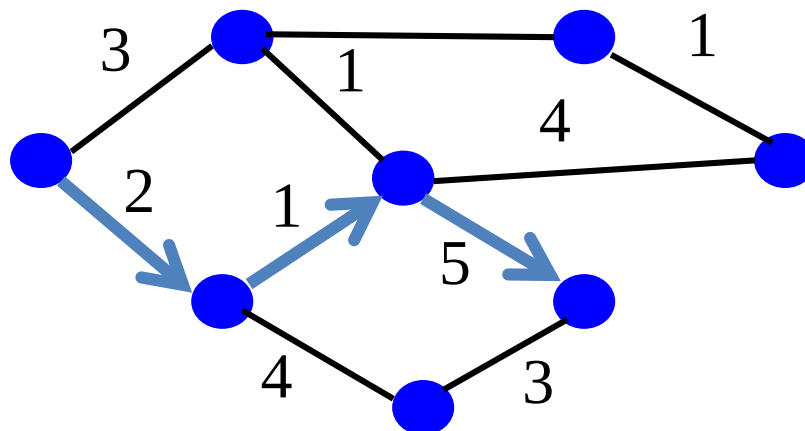
- Topology change
  - Link or node failure
  - Link or node recovery
- Configuration change
  - Link cost change
- Periodically
  - Refresh the link-state information
  - Typically (say) 30 minutes
  - Corrects for possible corruption of the data

# When the Routers Disagree

(during transient periods)

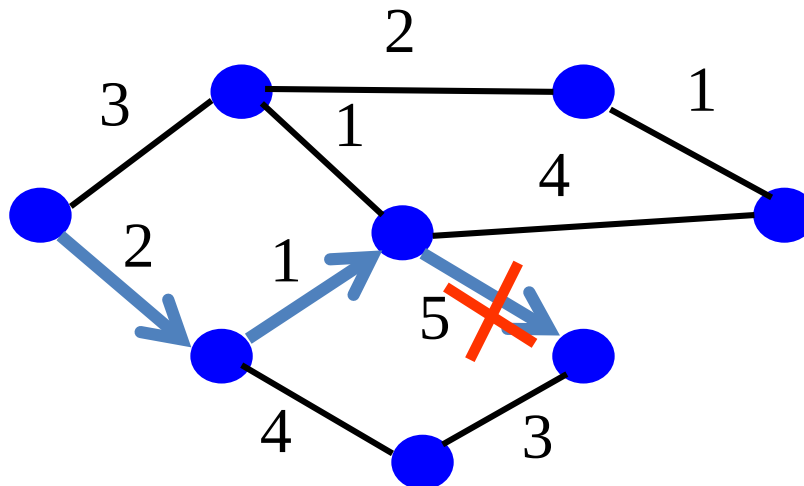
# Convergence

- Getting consistent routing information to all nodes
  - E.g., all nodes having the same link-state database
- Consistent forwarding after convergence
  - All nodes have the same link-state database
  - All nodes forward packets on shortest paths
  - The next router on the path forwards to the next hop



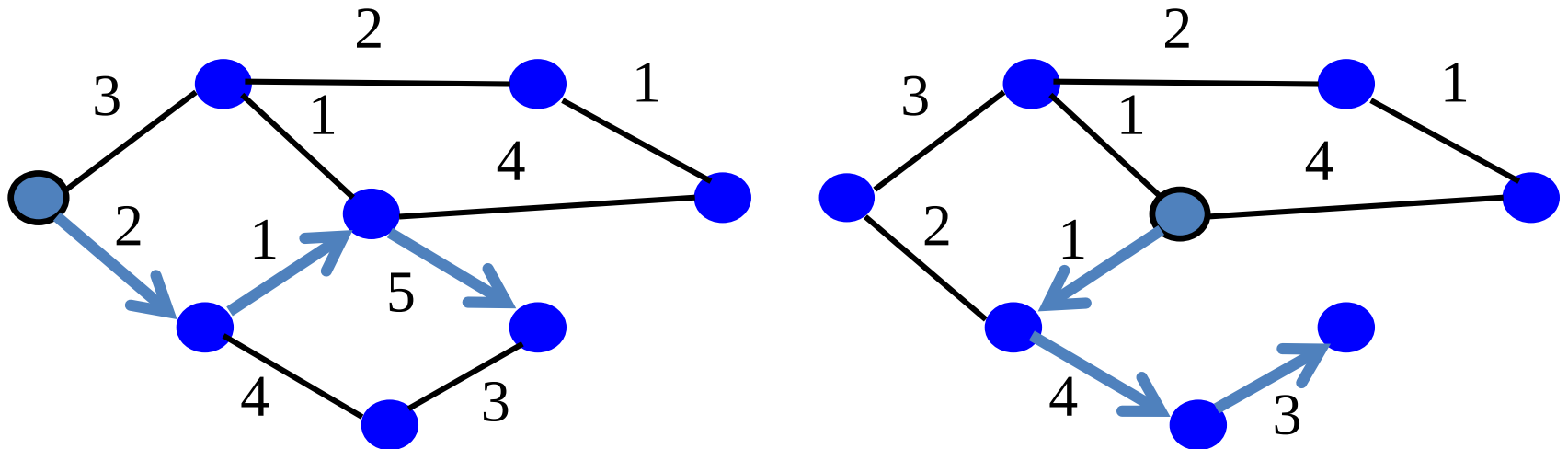
# Transient Disruptions

- Detection delay
  - A node does not detect a failed link immediately
  - ... and forwards data packets into a “blackhole”
  - Depends on timeout for detecting lost hellos



# Transient Disruptions

- Inconsistent link-state database
  - Some routers know about failure before others
  - The shortest paths are no longer consistent
  - Can cause transient forwarding loops



# Convergence Delay

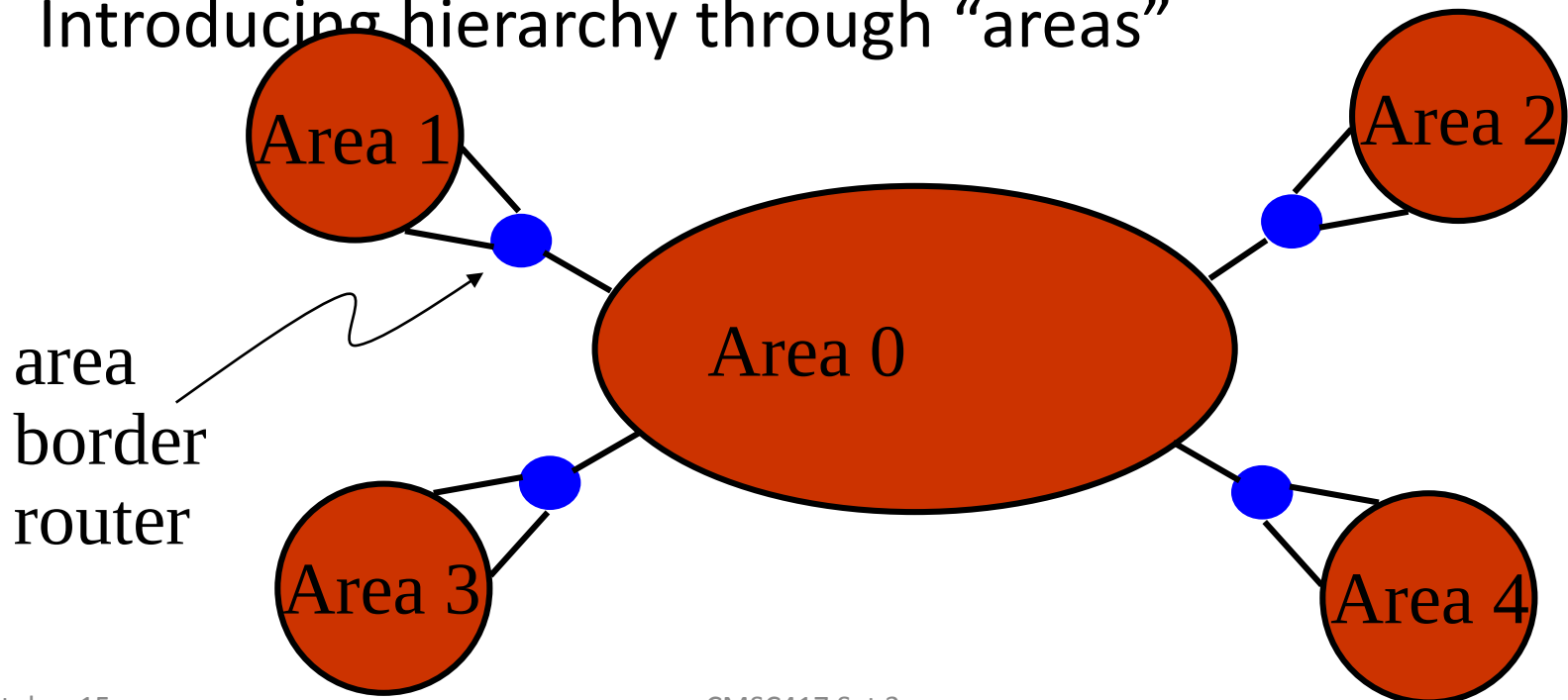
- Sources of convergence delay
  - Detection latency
  - Flooding of link-state information
  - Shortest-path computation
  - Creating the forwarding table
- Performance during convergence period
  - Lost packets due to blackholes and TTL expiry
  - Looping packets consuming resources
  - Out-of-order packets reaching the destination
- Very bad for VoIP, online gaming, and video

# Reducing Convergence Delay

- Faster detection
  - Smaller hello timers
  - Link-layer technologies that can detect failures
- Faster flooding
  - Flooding immediately
  - Sending link-state packets with high-priority
- Faster computation
  - Faster processors on the routers
  - Incremental Dijkstra's algorithm
- Faster forwarding-table update
  - Data structures supporting incremental updates

# Scaling Link-State Routing

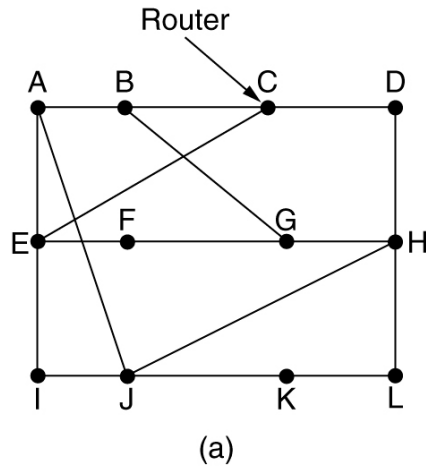
- Overhead of link-state routing
  - Flooding link-state packets throughout the network
  - Running Dijkstra's shortest-path algorithm
- Introducing hierarchy through “areas”



# Some Properties

- Routing is a distributed algorithm
  - React to changes in the topology
  - Compute the paths through the network
- Shortest-path link state routing
  - Flood link weights throughout the network
  - Compute shortest paths as a sum of link weights
  - Forward packets on next hop in the shortest path
- Convergence process
  - Changing from one topology to another
  - Transient periods of inconsistency across routers

# Distance Vector Routing



New estimated delay from J

| To | A  | I  | H  | K  | Line |
|----|----|----|----|----|------|
| A  | 0  | 24 | 20 | 21 | 8 A  |
| B  | 12 | 36 | 31 | 28 | 20 A |
| C  | 25 | 18 | 19 | 36 | 28 I |
| D  | 40 | 27 | 8  | 24 | 20 H |
| E  | 14 | 7  | 30 | 22 | 17 I |
| F  | 23 | 20 | 19 | 40 | 30 I |
| G  | 18 | 31 | 6  | 31 | 18 H |
| H  | 17 | 20 | 0  | 19 | 12 H |
| I  | 21 | 0  | 14 | 22 | 10 I |
| J  | 9  | 11 | 7  | 10 | 0 -  |
| K  | 24 | 22 | 22 | 0  | 6 K  |
| L  | 29 | 33 | 9  | 9  | 15 K |

|               |                |                |               |
|---------------|----------------|----------------|---------------|
| JA delay is 8 | JI delay is 10 | JH delay is 12 | JK delay is 6 |
|---------------|----------------|----------------|---------------|

Vectors received from J's four neighbors

New routing table for J

(b)

(a) A subnet. (b) Input from A, I, H, K, and the new routing table for J.

# Distance Vector Algorithm

- $c(x,v)$  = cost for direct link from  $x$  to  $v$ 
  - Node  $x$  maintains costs of direct links  $c(x,v)$
- $D_x(y)$  = estimate of least cost from  $x$  to  $y$ 
  - Node  $x$  maintains distance vector  $\mathbf{D}_x = [D_x(y): y \in N]$
- Node  $x$  maintains its neighbors' distance vectors
  - For each neighbor  $v$ ,  $x$  maintains  $\mathbf{D}_v = [D_v(y): y \in N]$
- Each node  $v$  periodically sends  $D_v$  to its neighbors
  - And neighbors update their own distance vectors
  - $D_x(y) \leftarrow \min_v \{c(x,v) + D_v(y)\}$  for each node  $y \in N$
- Over time, the distance vector  $D_x$  converges

# Distance Vector Algorithm

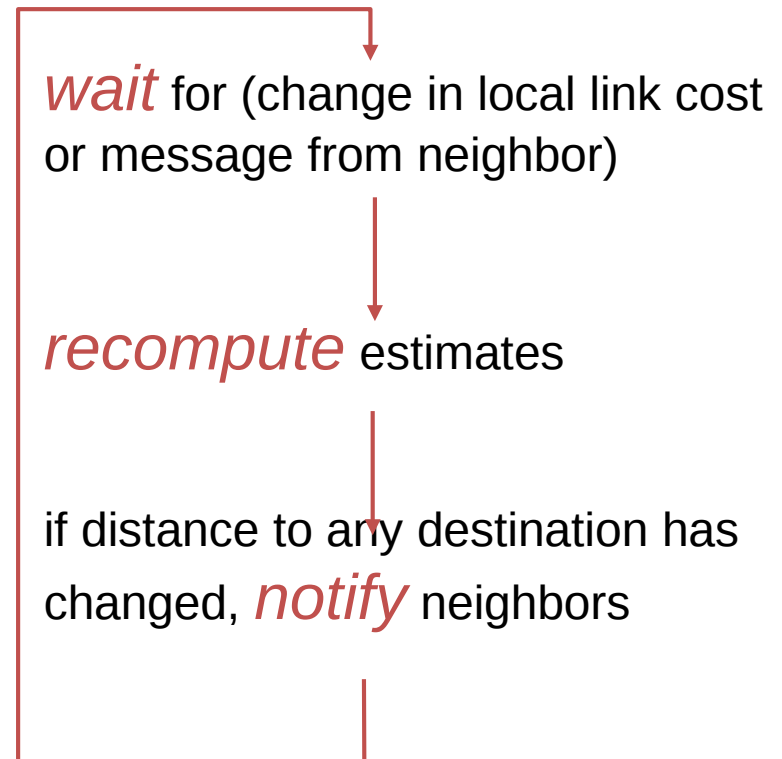
**Iterative, asynchronous:** each local iteration caused by:

- Local link cost change
- Distance vector update message from neighbor

**Distributed:**

- Each node notifies neighbors *only* when its DV changes
- Neighbors then notify their neighbors if necessary

**Each node:**



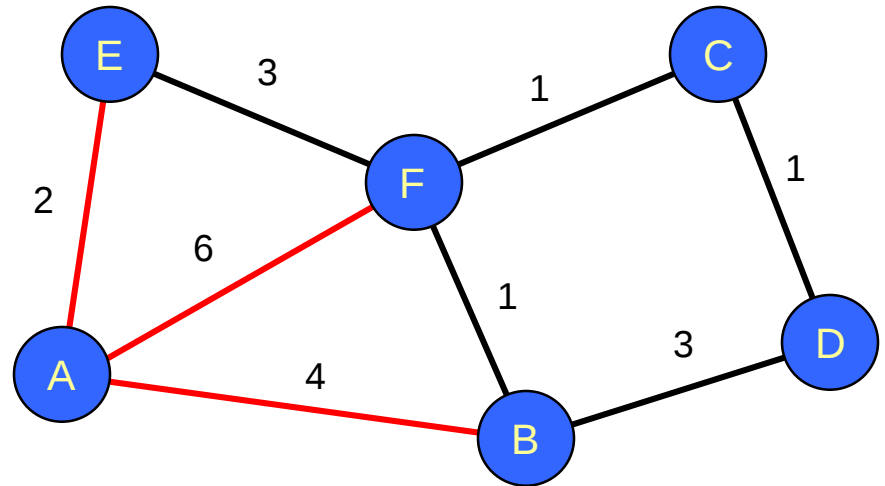
# Distance Vector Example: Step 1

Optimum 1-hop paths

| Table for A |          |     | Table for B |          |     |
|-------------|----------|-----|-------------|----------|-----|
| Dst         | Cst      | Hop | Dst         | Cst      | Hop |
| A           | 0        | A   | A           | 4        | A   |
| B           | 4        | B   | B           | 0        | B   |
| C           | $\infty$ | —   | C           | $\infty$ | —   |
| D           | $\infty$ | —   | D           | 3        | D   |
| E           | 2        | E   | E           | $\infty$ | —   |
| F           | 6        | F   | F           | 1        | F   |

| Table for C |          |     | Table for D |          |     | Table for E |          |     | Table for F |          |     |
|-------------|----------|-----|-------------|----------|-----|-------------|----------|-----|-------------|----------|-----|
| Dst         | Cst      | Hop | Dst         | Cst      | Hop | Dst         | Cst      | Hop | Dst         | Cst      | Hop |
| A           | $\infty$ | —   | A           | $\infty$ | —   | A           | 2        | A   | A           | 6        | A   |
| B           | $\infty$ | —   | B           | 3        | B   | B           | $\infty$ | —   | B           | 1        | B   |
| C           | 0        | C   | C           | 1        | C   | C           | $\infty$ | —   | C           | 1        | C   |
| D           | 1        | D   | D           | 0        | D   | D           | $\infty$ | —   | D           | $\infty$ | —   |
| E           | $\infty$ | —   | E           | $\infty$ | —   | E           | 0        | E   | E           | 3        | E   |
| F           | 1        | F   | F           | $\infty$ | —   | F           | 3        | F   | F           | 0        | F   |



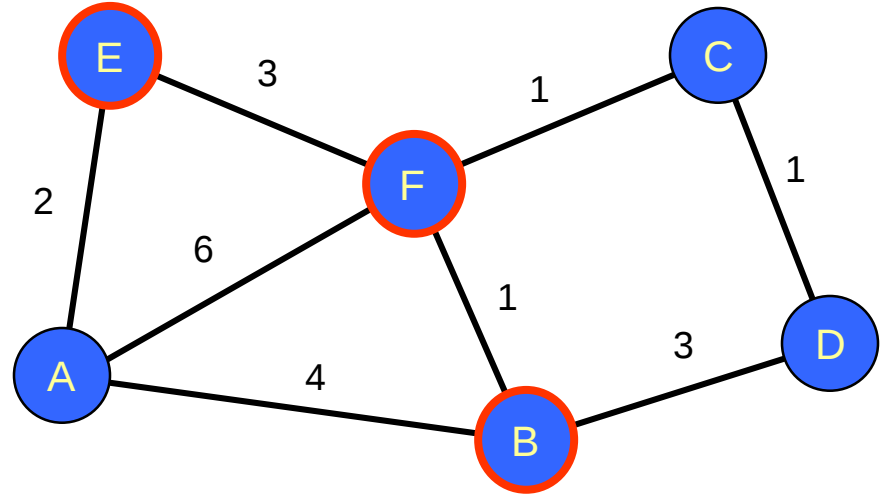
# Distance Vector Example: Step 2

Optimum 2-hop paths

| Table for A |     |     | Table for B |     |     |
|-------------|-----|-----|-------------|-----|-----|
| Dst         | Cst | Hop | Dst         | Cst | Hop |
| A           | 0   | A   | A           | 4   | A   |
| B           | 4   | B   | B           | 0   | B   |
| C           | 7   | F   | C           | 2   | F   |
| D           | 7   | B   | D           | 3   | D   |
| E           | 2   | E   | E           | 4   | F   |
| F           | 5   | E   | F           | 1   | F   |

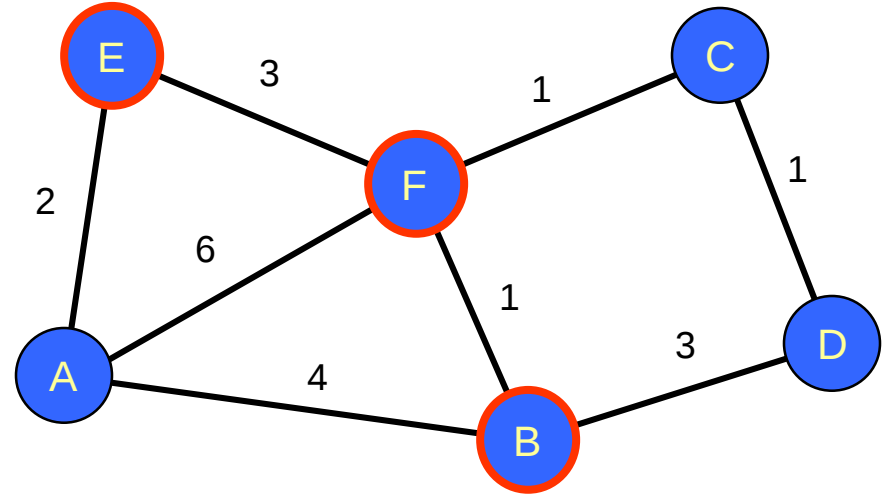
| Table for C |     |     | Table for D |          |     | Table for E |          |     | Table for F |     |     |
|-------------|-----|-----|-------------|----------|-----|-------------|----------|-----|-------------|-----|-----|
| Dst         | Cst | Hop | Dst         | Cst      | Hop | Dst         | Cst      | Hop | Dst         | Cst | Hop |
| A           | 7   | F   | A           | 7        | B   | A           | 2        | A   | A           | 5   | B   |
| B           | 2   | F   | B           | 3        | B   | B           | 4        | F   | B           | 1   | B   |
| C           | 0   | C   | C           | 1        | C   | C           | 4        | F   | C           | 1   | C   |
| D           | 1   | D   | D           | 0        | D   | D           | $\infty$ | —   | D           | 2   | C   |
| E           | 4   | F   | E           | $\infty$ | —   | E           | 0        | E   | E           | 3   | E   |
| F           | 1   | F   | F           | 2        | C   | F           | 3        | F   | F           | 0   | F   |



# Distance Vector Example: Step 3

Optimum 3-hop paths

| Table for A |     |     | Table for B |     |     |
|-------------|-----|-----|-------------|-----|-----|
| Dst         | Cst | Hop | Dst         | Cst | Hop |
| A           | 0   | A   | A           | 4   | A   |
| B           | 4   | B   | B           | 0   | B   |
| C           | 6   | E   | C           | 2   | F   |
| D           | 7   | B   | D           | 3   | D   |
| E           | 2   | E   | E           | 4   | F   |
| F           | 5   | E   | F           | 1   | F   |

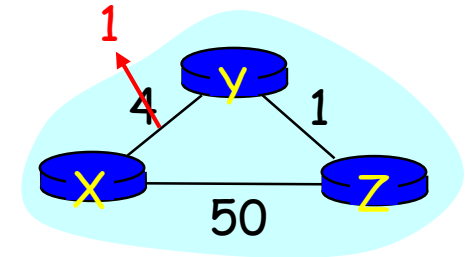


| Table for C |     |     | Table for D |     |     | Table for E |     |     | Table for F |     |     |
|-------------|-----|-----|-------------|-----|-----|-------------|-----|-----|-------------|-----|-----|
| Dst         | Cst | Hop | Dst         | Cst | Hop | Dst         | Cst | Hop | Dst         | Cst | Hop |
| A           | 6   | F   | A           | 7   | B   | A           | 2   | A   | A           | 5   | B   |
| B           | 2   | F   | B           | 3   | B   | B           | 4   | F   | B           | 1   | B   |
| C           | 0   | C   | C           | 1   | C   | C           | 4   | F   | C           | 1   | C   |
| D           | 1   | D   | D           | 0   | D   | D           | 5   | F   | D           | 2   | C   |
| E           | 4   | F   | E           | 5   | C   | E           | 0   | E   | E           | 3   | E   |
| F           | 1   | F   | F           | 2   | C   | F           | 3   | F   | F           | 0   | F   |

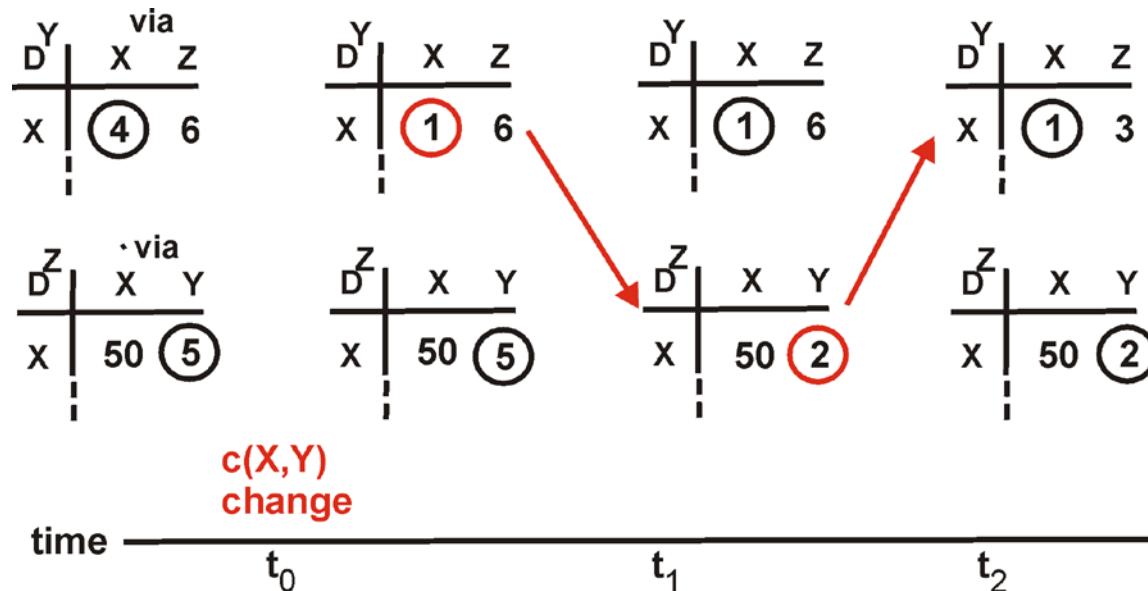
# Distance Vector: Link Cost Changes

## Link cost changes:

- Node detects local link cost change
- Updates the distance table
- If cost change in least cost path, notify neighbors



"good news travels fast"

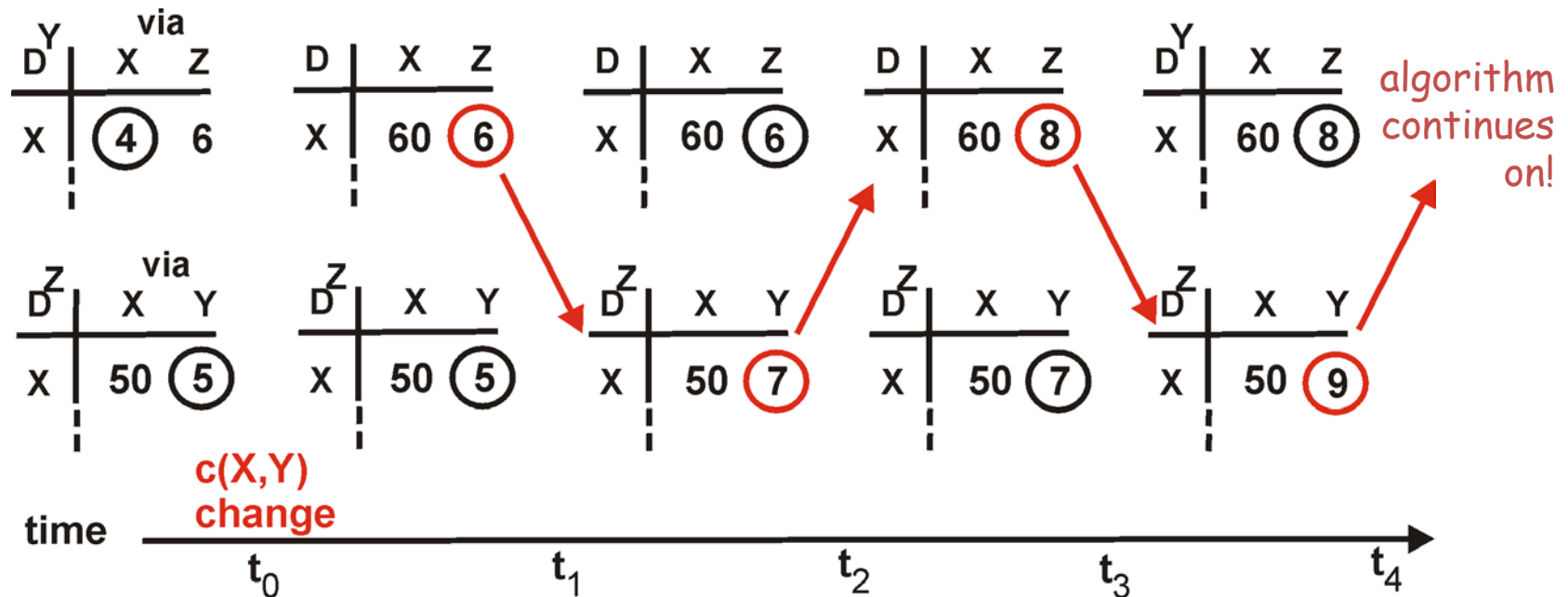
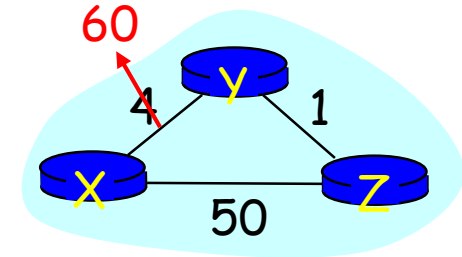


algorithm terminates

# Distance Vector: Link Cost Changes

## Link cost changes:

- Good news travels fast
- Bad news travels slow - “count to infinity” problem!



# Distance Vector Routing

| A | B | C | D | E |                   |
|---|---|---|---|---|-------------------|
| • | • | • | • | • | Initially         |
|   | • | • | • | • |                   |
|   | 1 | • | • | • | After 1 exchange  |
|   | 1 | 2 | • | • | After 2 exchanges |
|   | 1 | 2 | 3 | • | After 3 exchanges |
|   | 1 | 2 | 3 | 4 | After 4 exchanges |

(a)

| A | B | C | D | E |                   |
|---|---|---|---|---|-------------------|
| • | • | • | • | • | Initially         |
|   | 1 | 2 | 3 | 4 |                   |
|   | 3 | 2 | 3 | 4 | After 1 exchange  |
|   | 3 | 4 | 3 | 4 | After 2 exchanges |
|   | 5 | 4 | 5 | 4 | After 3 exchanges |
|   | 5 | 6 | 5 | 6 | After 4 exchanges |
|   | 7 | 6 | 7 | 6 | After 5 exchanges |
|   | 7 | 8 | 7 | 8 | After 6 exchanges |
|   |   | ⋮ |   |   |                   |
|   | • | • | • | • |                   |

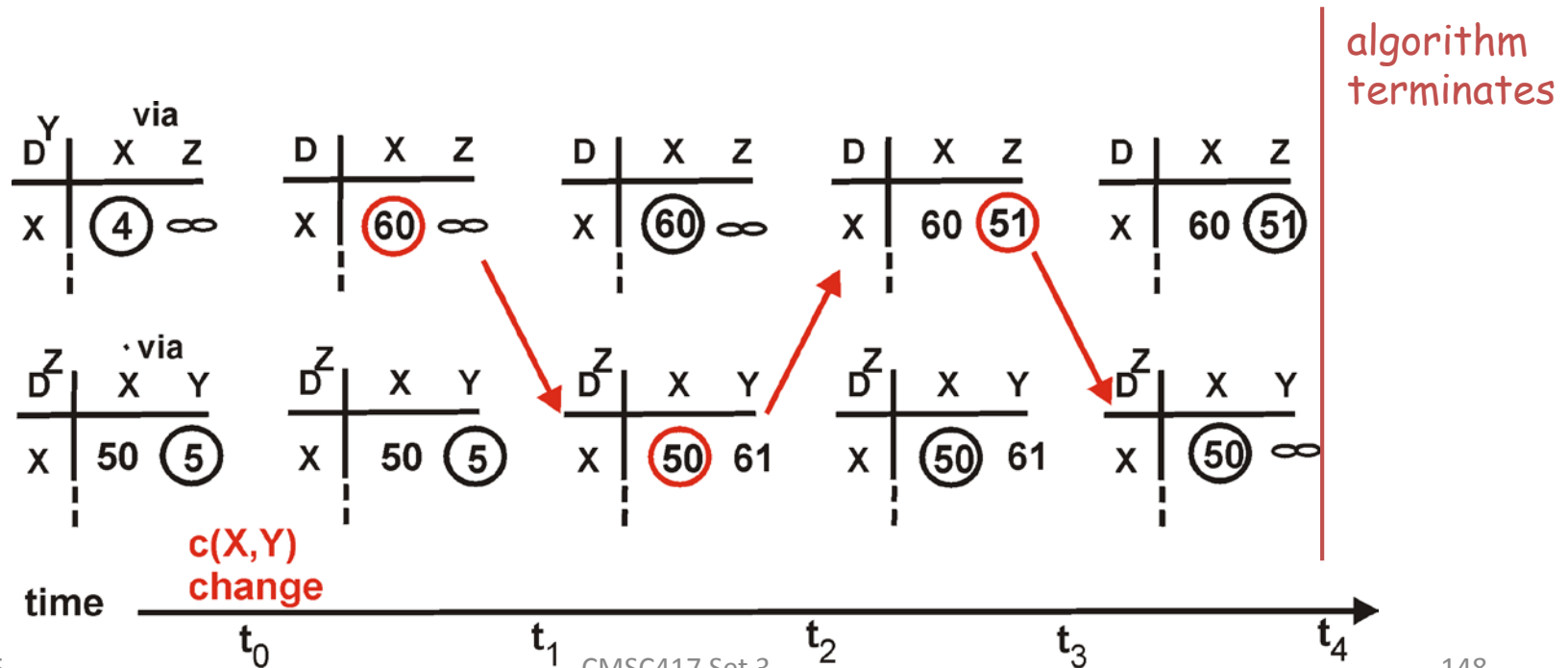
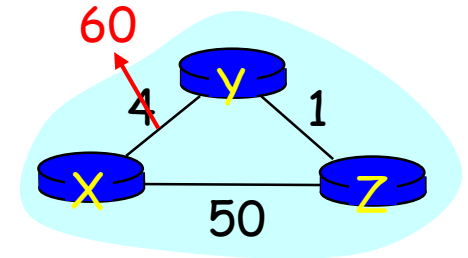
(b)

The count-to-infinity problem.

# Distance Vector: Poison Reverse

If Z routes through Y to get to X :

- Z tells Y its (Z's) distance to X is infinite (so Y won't route to X via Z)
- Still, can have problems when more than 2 routers are involved



# Routing Information Protocol (RIP)

- Distance vector protocol
  - Nodes send distance vectors every 30 seconds
  - ... or, when an update causes a change in routing
- Link costs in RIP
  - All links have cost 1
  - Valid distances of 1 through 15
  - ... with 16 representing infinity
  - Small “infinity” → smaller “counting to infinity” problem
- RIP is limited to fairly small networks
  - E.g., used in some campus networks

# Comparison of LS and DV Routing

## Message complexity

- LS: with  $n$  nodes,  $E$  links,  $O(nE)$  messages sent
- DV: exchange between neighbors only

## Speed of Convergence

- LS: relatively fast
- DV: convergence time varies
  - May be routing loops
  - Count-to-infinity problem

**Robustness:** what happens if router malfunctions?

### LS:

- Node can advertise incorrect *link* cost
- Each node computes only its *own* table

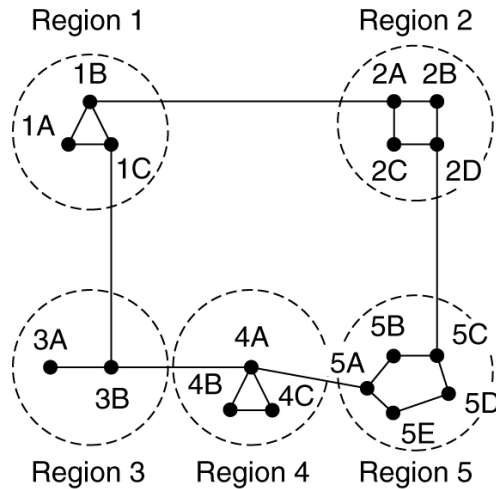
### DV:

- DV node can advertise incorrect *path* cost
- Each node's table used by others (error propagates)

# Similarities of LS and DV Routing

- Shortest-path routing
  - Metric-based, using link weights
  - Routers share a common view of how good a path is
- As such, commonly used *inside* an organization
  - RIP and OSPF are mostly used as *intradomain* protocols
  - E.g., Princeton uses RIP, and AT&T uses OSPF
- But the Internet is a “network of networks”
  - How to stitch the many networks together?
  - When networks may not have common goals
  - ... and may not want to share information

# Hierarchical Routing



(a)

Full table for 1A

| Dest. | Line | Hops |
|-------|------|------|
| 1A    | —    | —    |
| 1B    | 1B   | 1    |
| 1C    | 1C   | 1    |
| 2A    | 1B   | 2    |
| 2B    | 1B   | 3    |
| 2C    | 1B   | 3    |
| 2D    | 1B   | 4    |
| 3A    | 1C   | 3    |
| 3B    | 1C   | 2    |
| 4A    | 1C   | 3    |
| 4B    | 1C   | 4    |
| 4C    | 1C   | 4    |
| 5A    | 1C   | 4    |
| 5B    | 1C   | 5    |
| 5C    | 1B   | 5    |
| 5D    | 1C   | 6    |
| 5E    | 1C   | 5    |

(b)

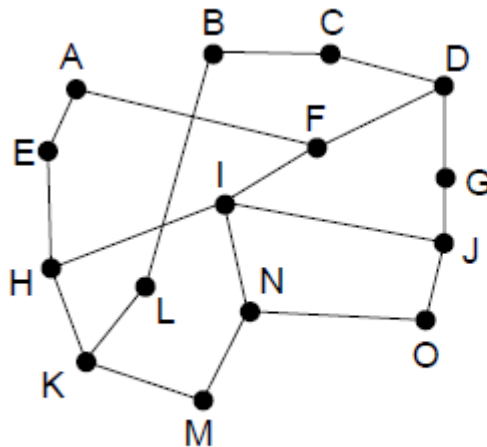
Hierarchical table for 1A

| Dest. | Line | Hops |
|-------|------|------|
| 1A    | —    | —    |
| 1B    | 1B   | 1    |
| 1C    | 1C   | 1    |
| 2     | 1B   | 2    |
| 3     | 1C   | 2    |
| 4     | 1C   | 3    |
| 5     | 1C   | 4    |

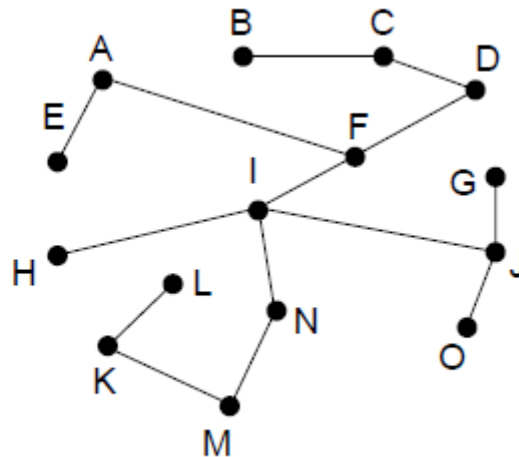
(c)

# Broadcast Routing

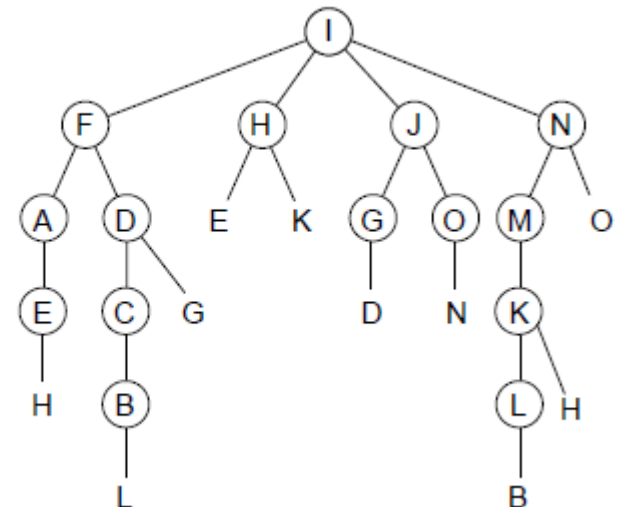
- Broadcast sends a packet to all nodes
  - RPF (Reverse Path Forwarding): send broadcast received on the link to the source out all remaining links
  - Alternatively, can build and use sink trees at all nodes



Network



Sink tree for I is  
efficient broadcast



RPF from I is larger  
than sink tree



- 

A graph with 10 vertices and 7 edges. The vertices are labeled with '1' or 'S'. The edges connect the vertices in a way that forms a single connected component. The graph is a tree with 10 vertices and 9 edges, but one edge is highlighted in red.

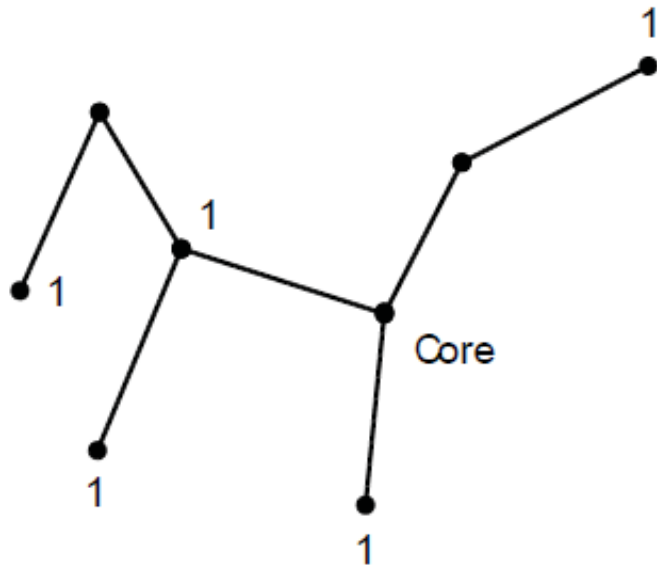
A graph with 10 vertices and 5 edges. The vertices are labeled with degrees: 2, 2, 2, 2, 2, 2, 2, 2, 2, 2. The edges are labeled with weights: 2, 2, 2, 2, 2. The graph is a cycle of length 5 with 5 additional vertices connected to the cycle vertices.

154

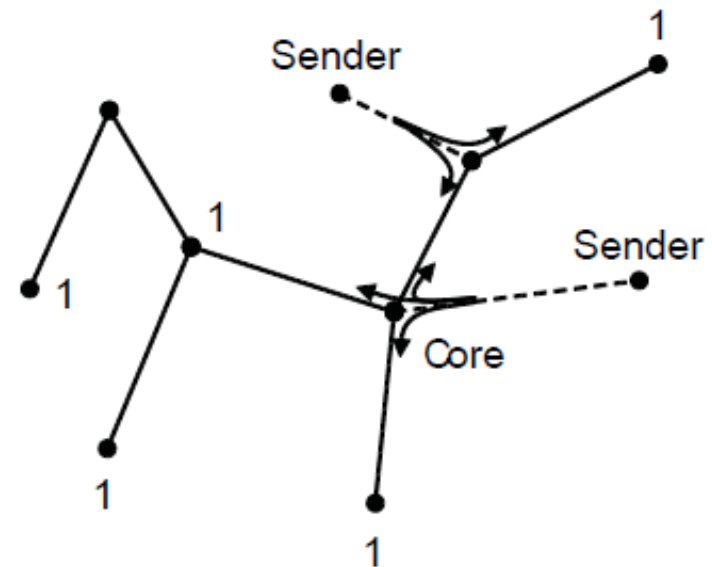


# Multicast Routing (2) – Sparse Case

- CBT (Core-Based Tree) uses a single tree to multicast
  - Tree is the sink tree from core node to group members



Sink tree from core to group 1

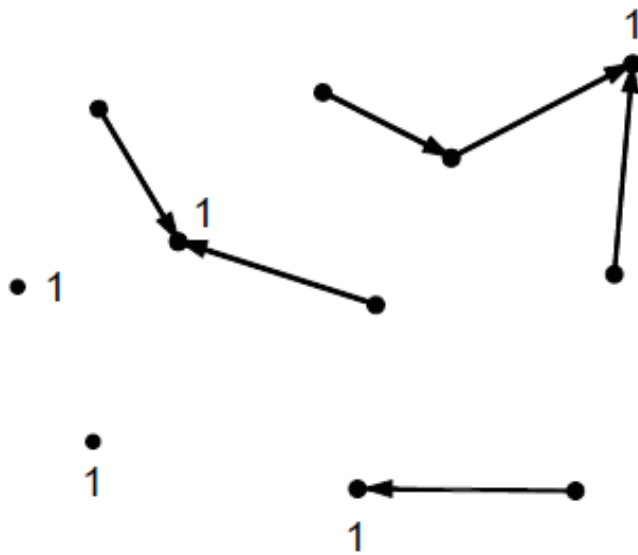


Multicast is send to the core then down when it reaches the sink tree

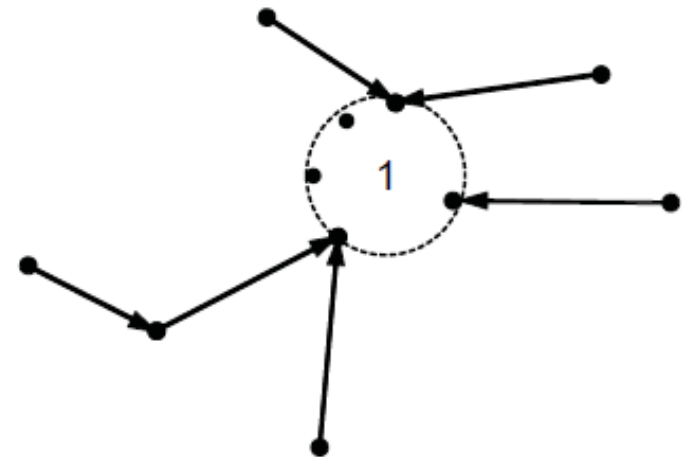


# Anycast Routing

Anycast sends a packet to one (nearest) group member



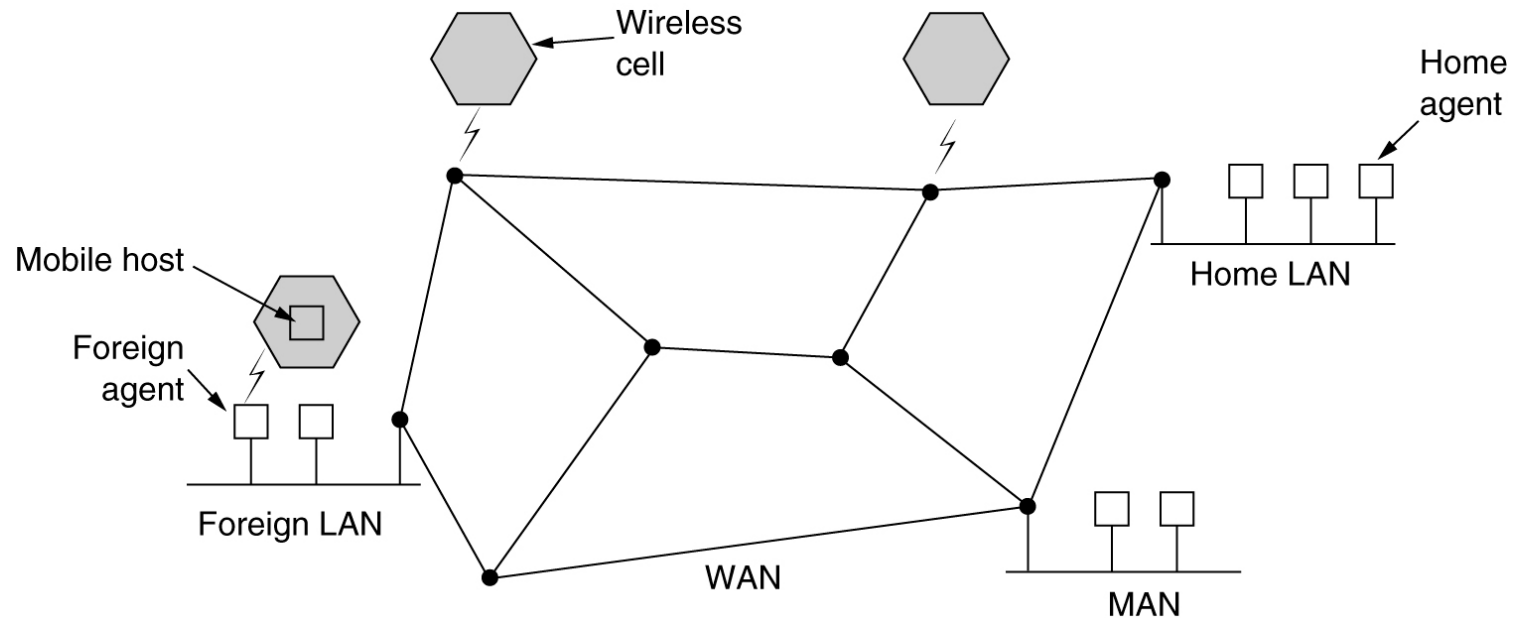
Anycast routes to group 1



Apparent topology of sink tree to "node" 1

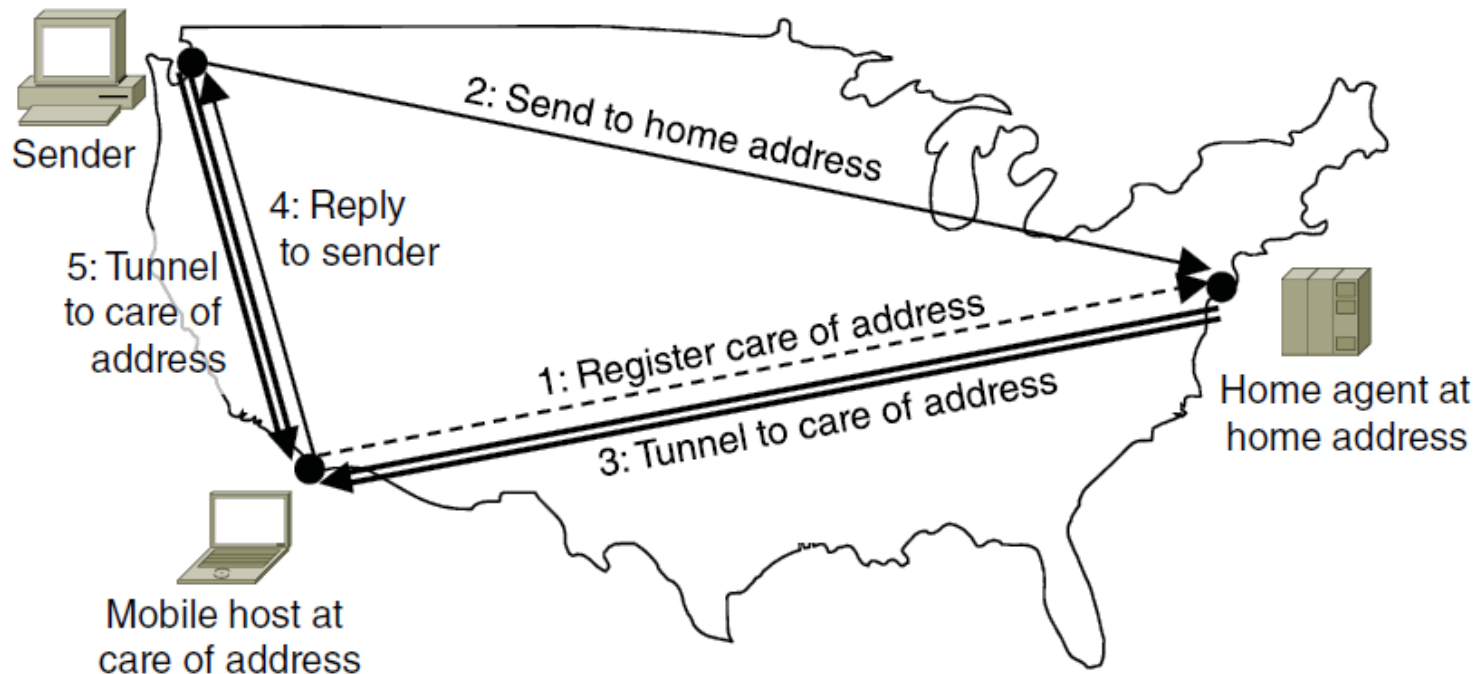
# Routing for Mobile Hosts

A WAN to which LANs, MANs, and wireless cells are attached.



# Routing for Mobile Hosts

- Mobile hosts can be reached via a home agent
  - Fixed home agent tunnels packets to reach the mobile host; reply can optimize path for subsequent packets
  - No changes to routers or fixed hosts



# Routing in Ad Hoc Networks

Possibilities when the routers are mobile:

1. Military vehicles on battlefield.

- No infrastructure.

2. A fleet of ships at sea.

- All moving all the time

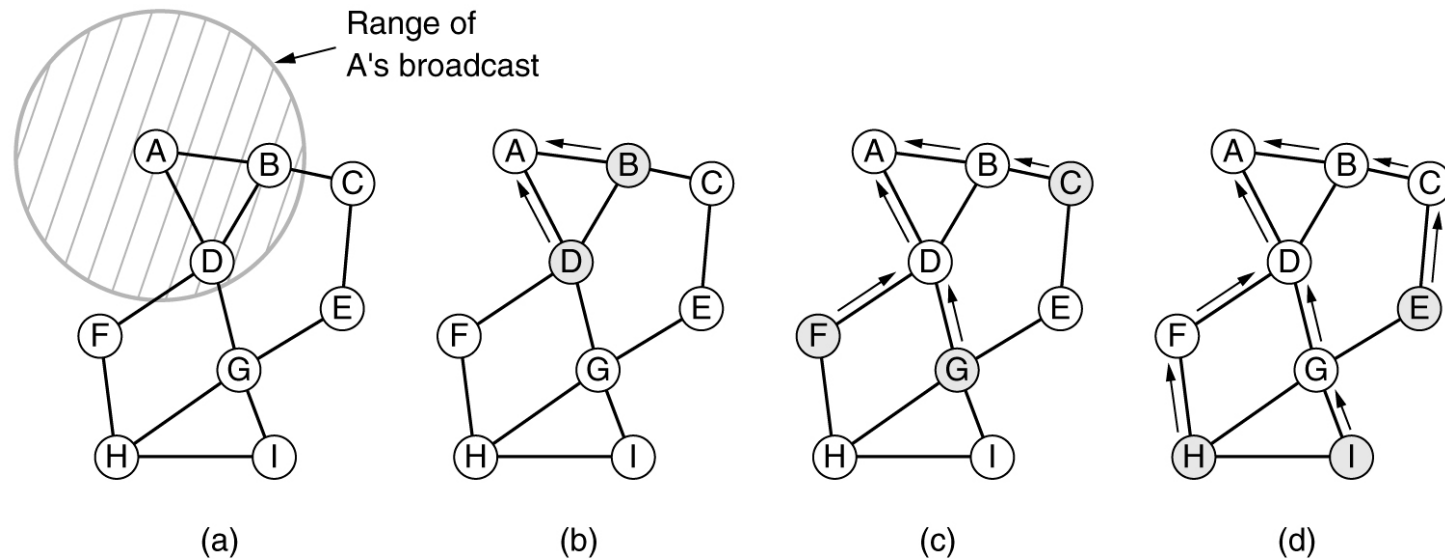
3. Emergency works at earthquake .

- The infrastructure destroyed.

4. A gathering of people with notebook computers.

- In an area lacking 802.11.

# Route Discovery



- (a) Range of A's broadcast.
- (b) After B and D have received A's broadcast.
- (c) After C, F, and G have received A's broadcast.
- (d) After E, H, and I have received A's broadcast.

Shaded nodes are new recipients. Arrows show possible reverse routes.

# Route Discovery (2)

| Source address | Request ID | Destination address | Source sequence # | Dest. sequence # | Hop count |
|----------------|------------|---------------------|-------------------|------------------|-----------|
|----------------|------------|---------------------|-------------------|------------------|-----------|

Format of a ROUTE REQUEST packet.

# Route Discovery (3)

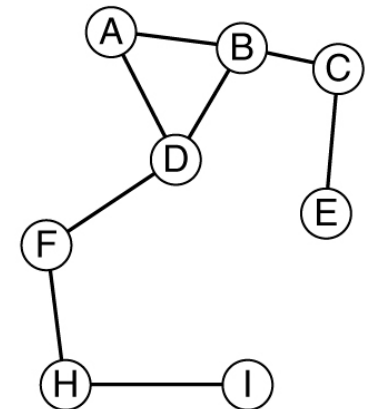
|                   |                        |                           |              |          |
|-------------------|------------------------|---------------------------|--------------|----------|
| Source<br>address | Destination<br>address | Destination<br>sequence # | Hop<br>count | Lifetime |
|-------------------|------------------------|---------------------------|--------------|----------|

Format of a ROUTE REPLY packet.

# Route Maintenance

| Dest. | Next hop | Distance | Active neighbors | Other fields |
|-------|----------|----------|------------------|--------------|
| A     | A        | 1        | F, G             |              |
| B     | B        | 1        | F, G             |              |
| C     | B        | 2        | F                |              |
| E     | G        | 2        |                  |              |
| F     | F        | 1        | A, B             |              |
| G     | G        | 1        | A, B             |              |
| H     | F        | 2        | A, B             |              |
| I     | G        | 2        | A, B             |              |

(a)

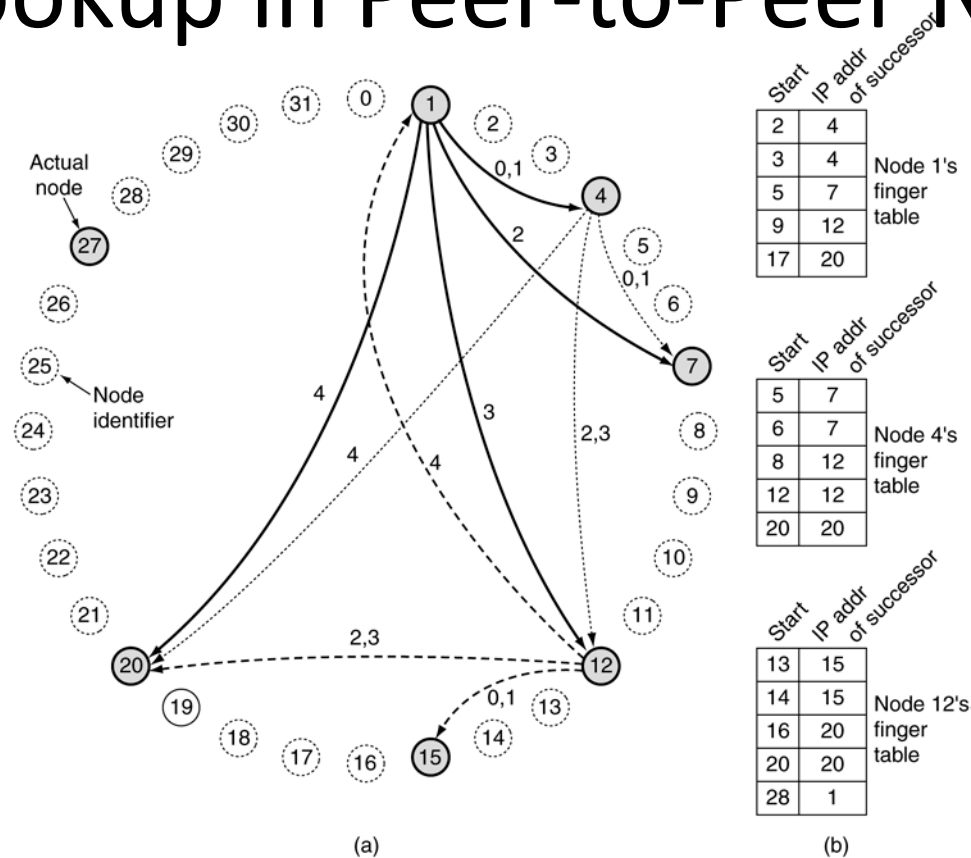


(b)

(a) D's routing table before G goes down.

(b) The graph after G has gone down.

# Node Lookup in Peer-to-Peer Networks

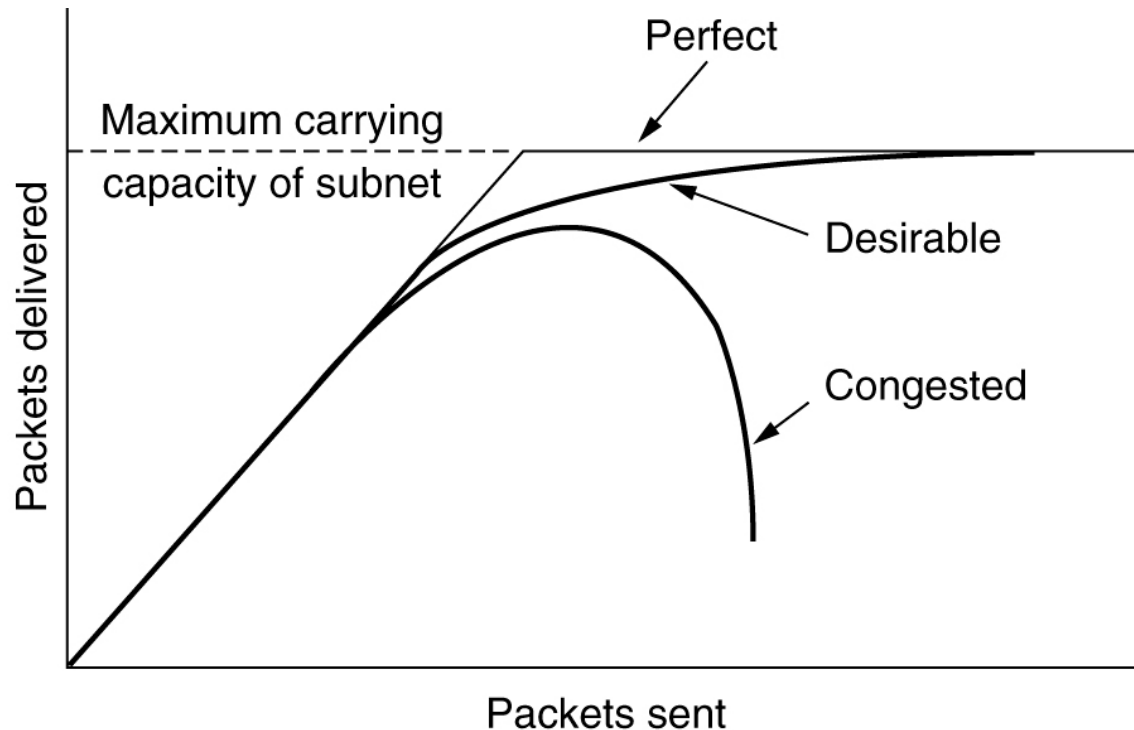


- (a) A set of 32 node identifiers arranged in a circle. The shaded ones correspond to actual machines. The arcs show the fingers from nodes 1, 4, and 12. The labels on the arcs are the table indices.
- (b) Examples of the finger tables.

# Congestion Control Algorithms

- General Principles of Congestion Control
- Congestion Prevention Policies
- Congestion Control in Virtual-Circuit Subnets
- Congestion Control in Datagram Subnets
- Load Shedding
- Jitter Control

# Congestion

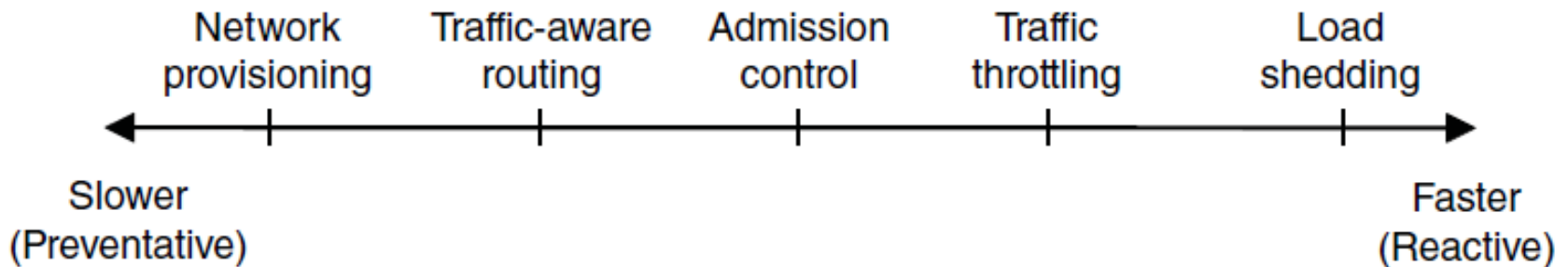


When too much traffic is offered, congestion sets in and performance degrades sharply.

# Congestion Control (3) – Approaches

Network must do its best with the offered load

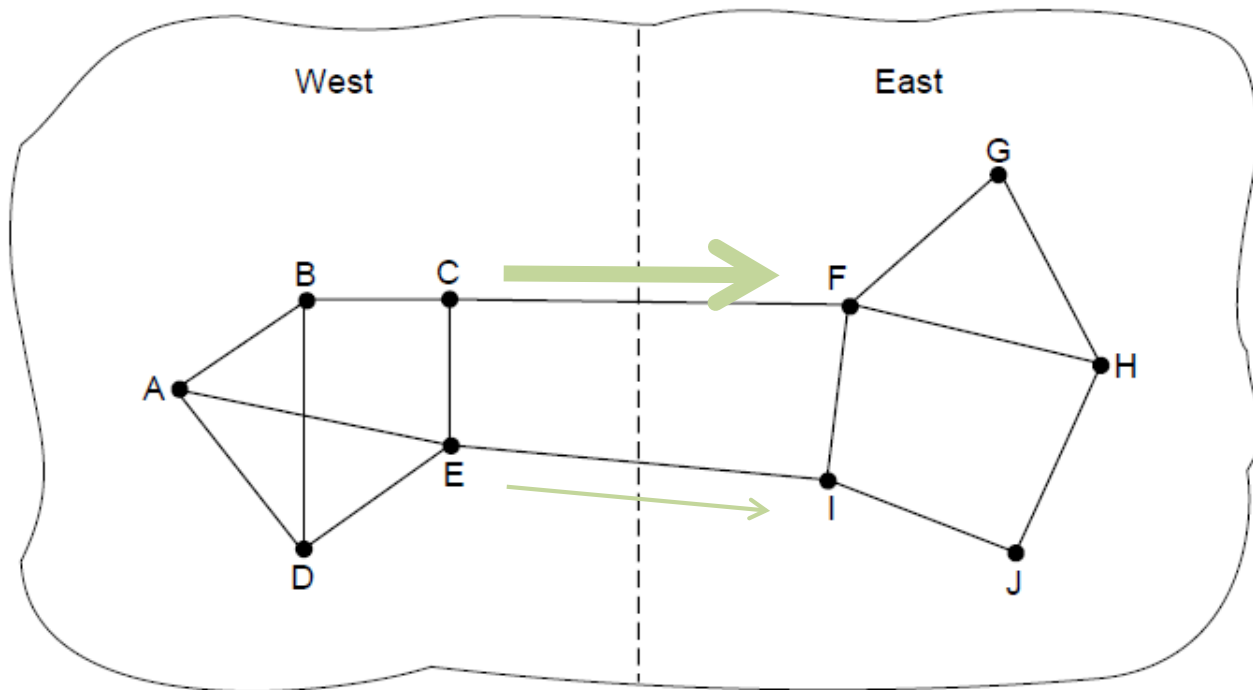
- Different approaches at different timescales
- Nodes should also reduce offered load (Transport)



# Traffic-Aware Routing

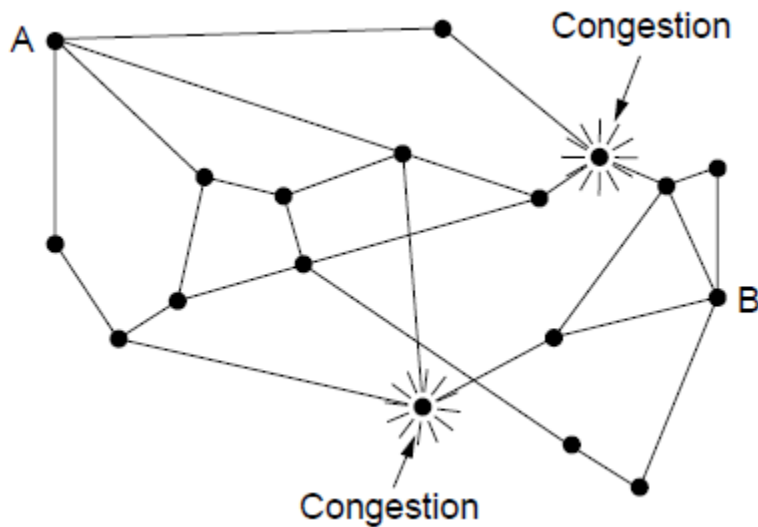
Choose routes depending on traffic, not just topology

- E.g., use *EI* for West-to-East traffic if *CF* is loaded
- But take care to avoid oscillations

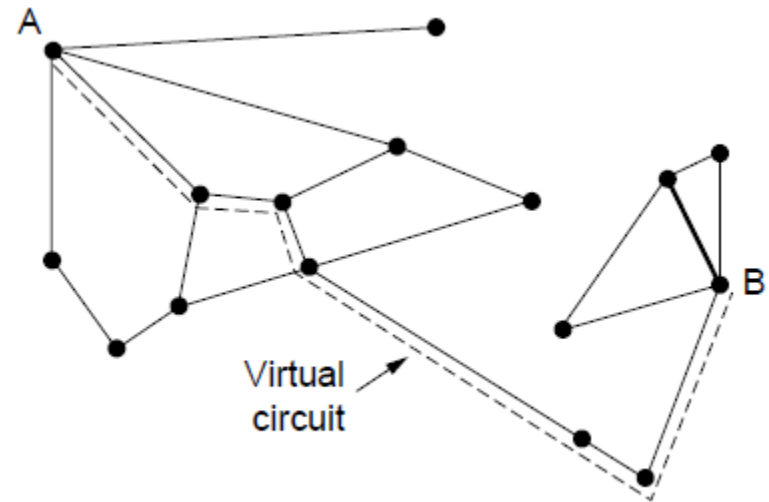


# Admission Control

- Admission control allows a new traffic load only if the network has sufficient capacity, e.g., with virtual circuits
  - Can combine with looking for an uncongested route



Network with some congested nodes

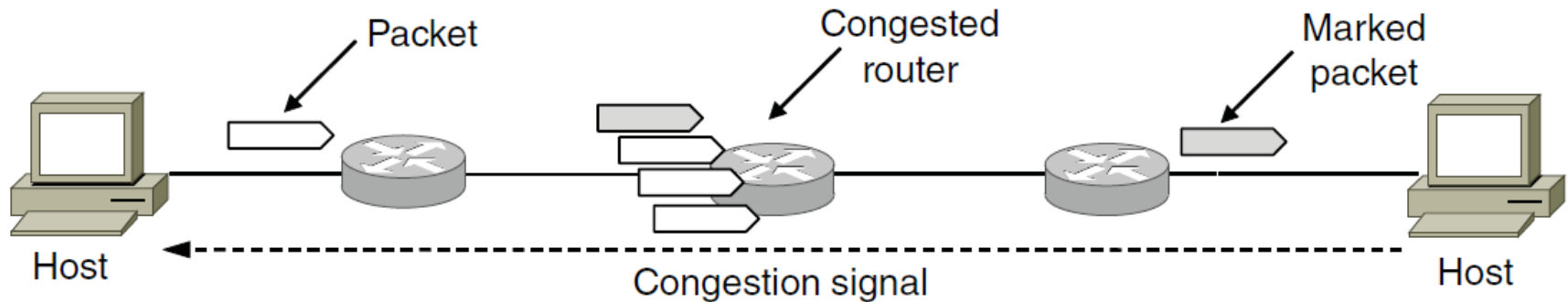


Uncongested portion and route AB around congestion

# Traffic Throttling

Congested routers signal hosts to slow down traffic

- ECN (Explicit Congestion Notification) marks packets and receiver returns signal to sender

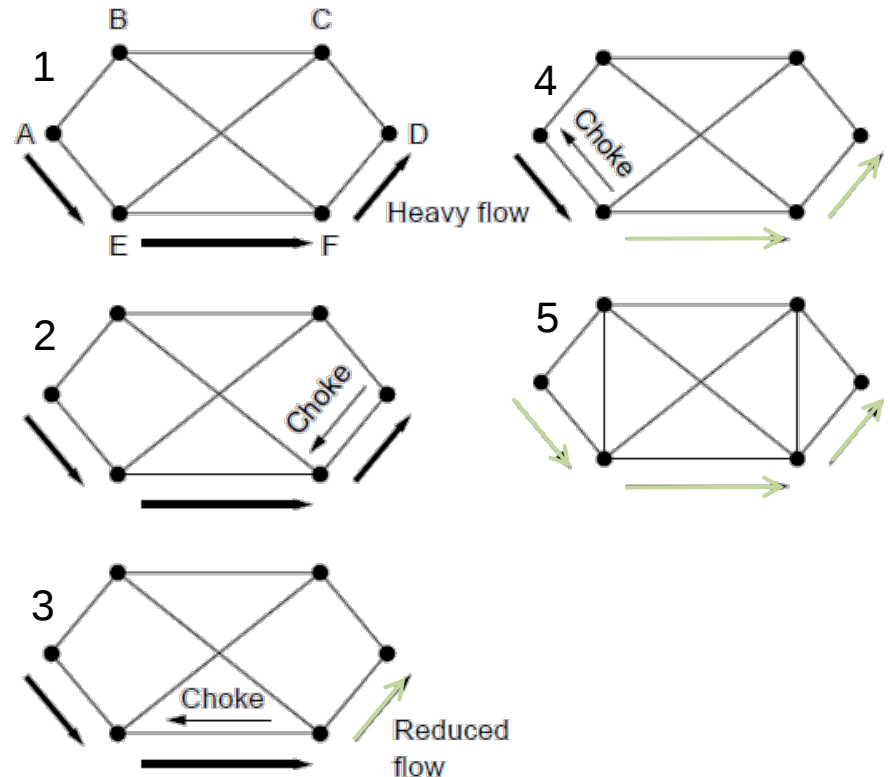


# Load Shedding (1)

When all else fails,  
network will drop  
packets (shed load)

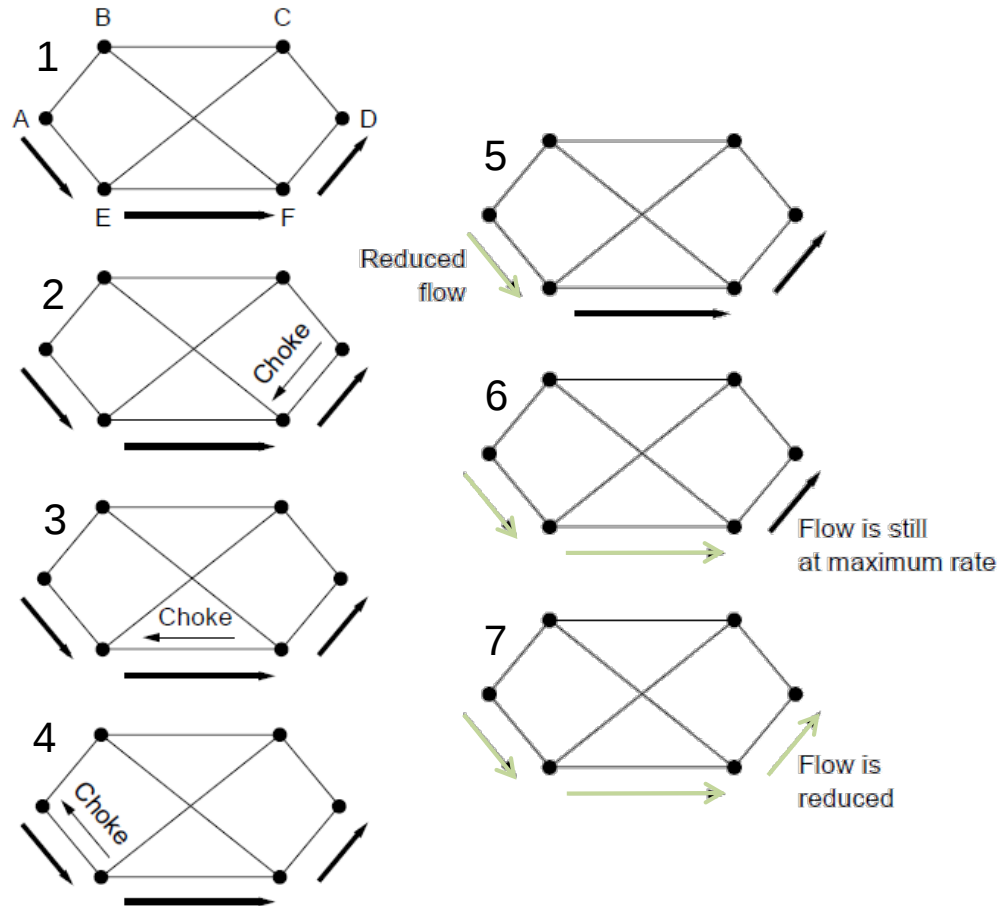
Can be done end-to-  
end or link-by-link

Link-by-link (right)  
produces rapid relief



# Load Shedding (2)

End-to-end (right)  
takes longer to  
have an effect, but  
can better target  
the cause of  
congestion



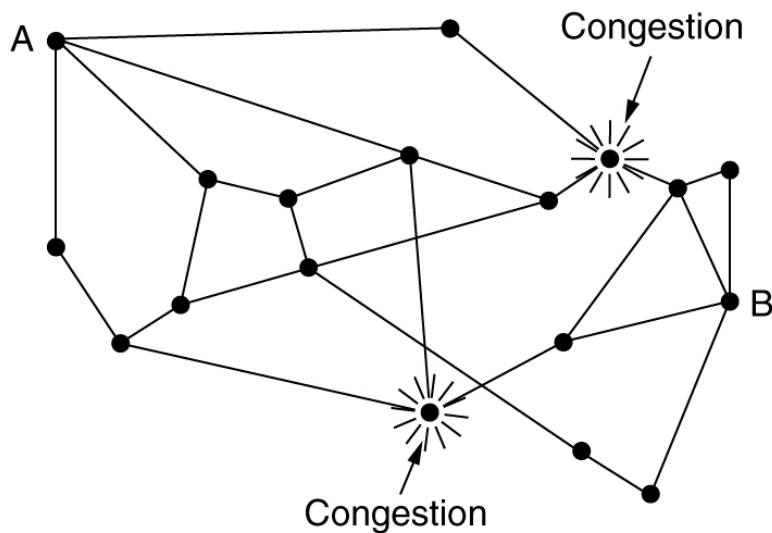
# General Principles of Congestion Control

1. Monitor the system .
  - detect when and where congestion occurs.
2. Pass information to where action can be taken.
3. Adjust system operation to correct the problem.

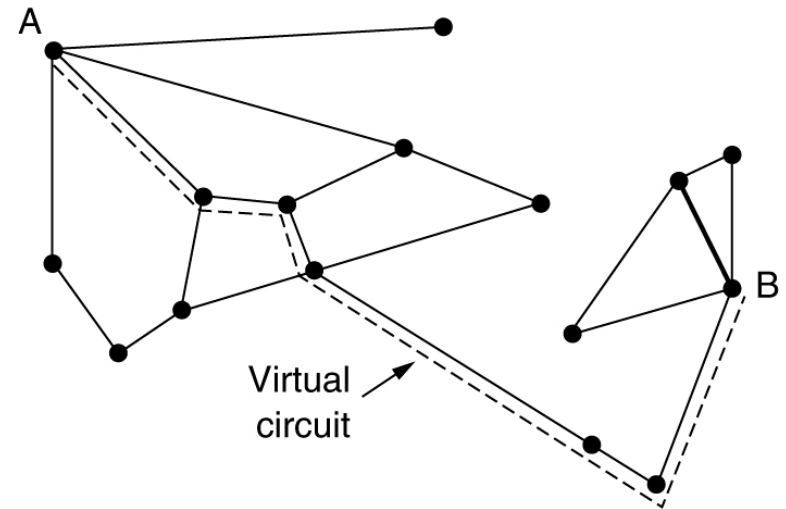
# Congestion Prevention Policies

| Layer     | Policies                                                                                                                                                                                                                                           |
|-----------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Transport | <ul style="list-style-type: none"><li>• Retransmission policy</li><li>• Out-of-order caching policy</li><li>• Acknowledgement policy</li><li>• Flow control policy</li><li>• Timeout determination</li></ul>                                       |
| Network   | <ul style="list-style-type: none"><li>• Virtual circuits versus datagram inside the subnet</li><li>• Packet queueing and service policy</li><li>• Packet discard policy</li><li>• Routing algorithm</li><li>• Packet lifetime management</li></ul> |
| Data link | <ul style="list-style-type: none"><li>• Retransmission policy</li><li>• Out-of-order caching policy</li><li>• Acknowledgement policy</li><li>• Flow control policy</li></ul>                                                                       |

# Congestion Control in Virtual-Circuit Subnets



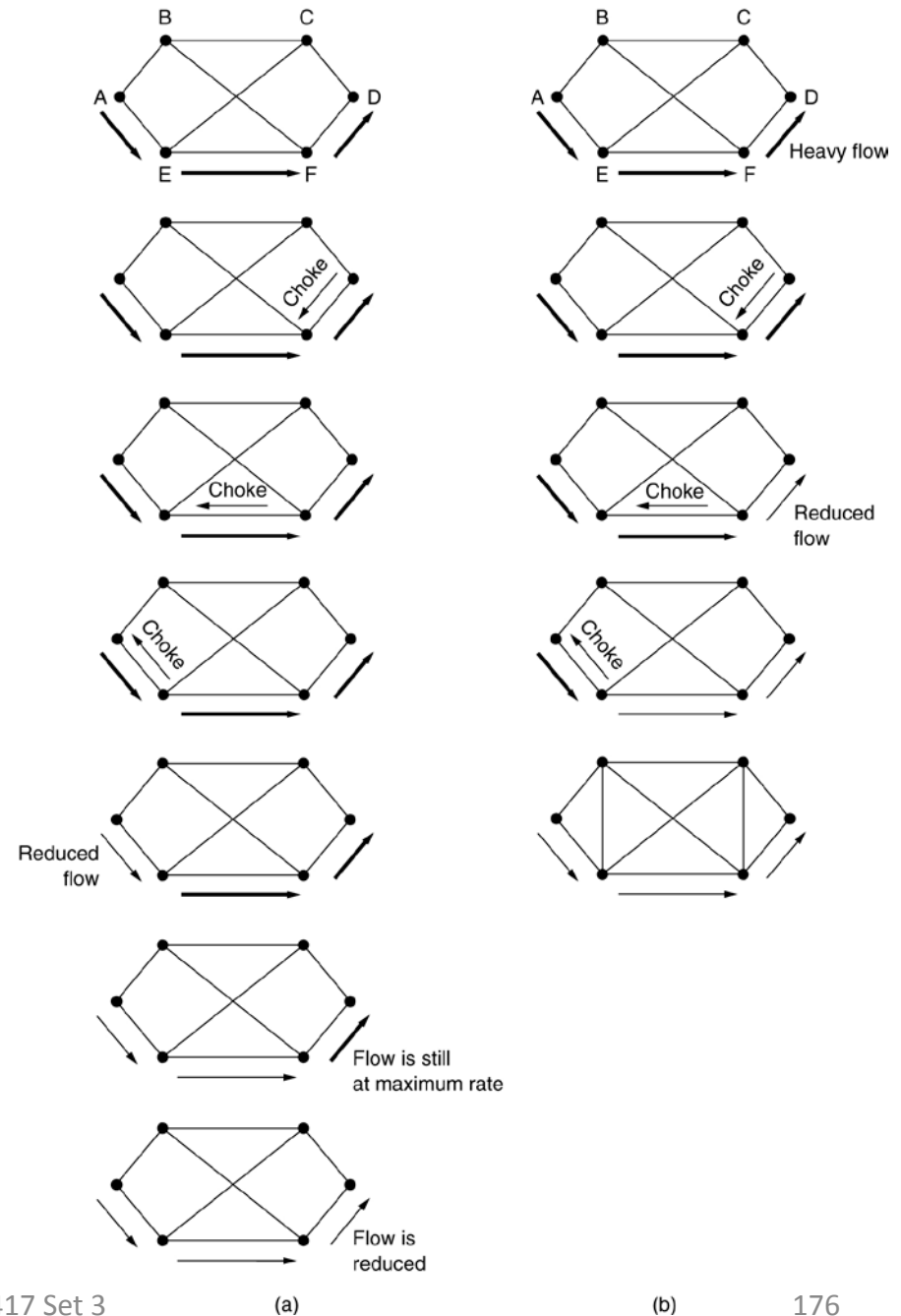
(a)



(b)

(a) A congested subnet. (b) A redrawn subnet, eliminates congestion and a virtual circuit from A to B.

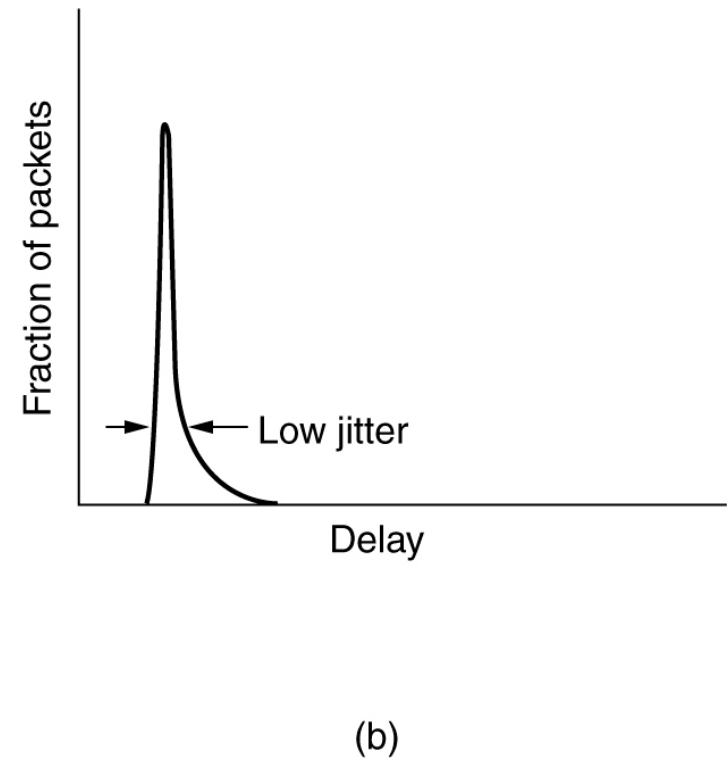
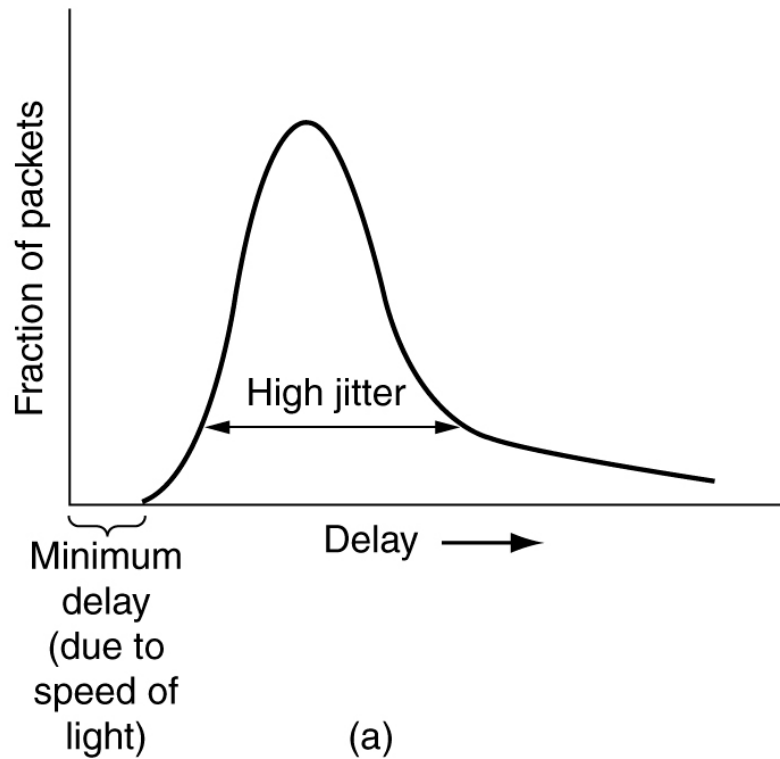
# Hop-by-Hop Choke Packets



(a) A choke packet that affects only the source.

(b) A choke packet that affects each hop it passes through.

# Jitter Control



(a) High jitter. (b) Low jitter.

# Quality of Service

- Application requirements
- Traffic shaping
- Packet scheduling
- Admission control
- Integrated services
- Differentiated services

# Application Requirements (1)

Different applications care about different properties

| Application       | Bandwidth | Delay  | Jitter | Loss   |
|-------------------|-----------|--------|--------|--------|
| Email             | Low       | Low    | Low    | Medium |
| File sharing      | High      | Low    | Low    | Medium |
| Web access        | Medium    | Medium | Low    | Medium |
| Remote login      | Low       | Medium | Medium | Medium |
| Audio on demand   | Low       | Low    | High   | Low    |
| Video on demand   | High      | Low    | High   | Low    |
| Telephony         | Low       | High   | High   | Low    |
| Videoconferencing | High      | High   | High   | Low    |

ed

“High” means a demanding requirement, e.g., low delay



# Application Requirements (2)

Network provides service with different kinds of QoS (Quality of Service) to meet application requirements

| Network Service                 | Application       |
|---------------------------------|-------------------|
| Constant bit rate               | Telephony         |
| Real-time variable bit rate     | Videoconferencing |
| Non-real-time variable bit rate | Streaming a movie |
| Available bit rate              | File transfer     |

Example of QoS categories from ATM networks

# Categories of QoS and Examples

## 1.Constant bit rate

- Telephony

## 2.Real-time variable bit rate

- Compressed videoconferencing

## 3.Non-real-time variable bit rate

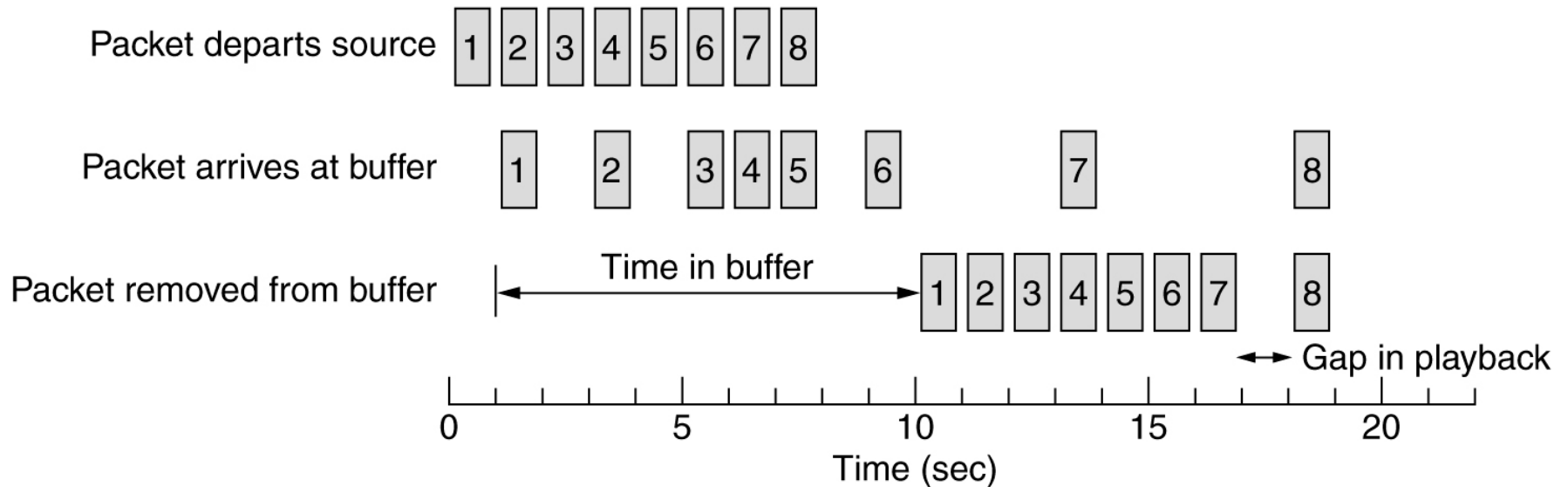
- Watching a movie on demand

## 4.Available bit rate

- File transfer

# Buffering

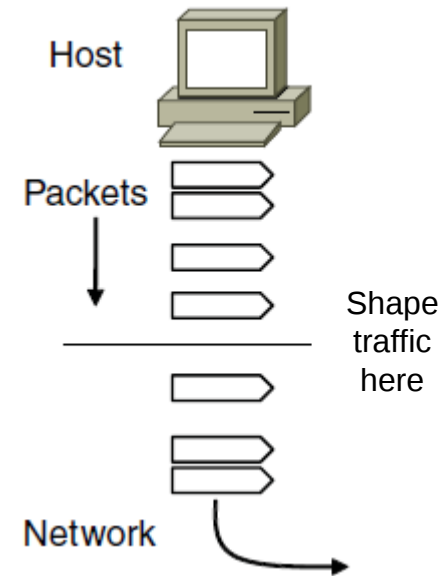
Smoothing the output stream by buffering packets.



# Traffic Shaping (1)

Traffic shaping  
regulates the average  
rate and burstiness of  
data entering the  
network

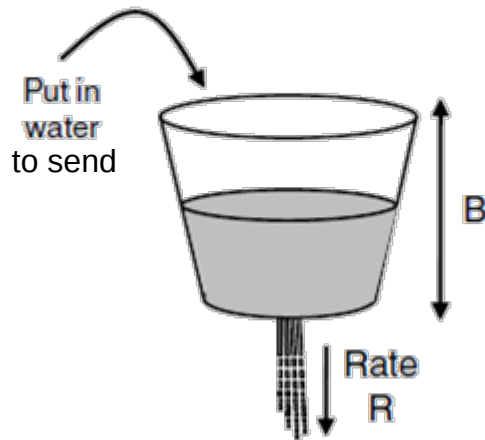
- Lets us make  
guarantees



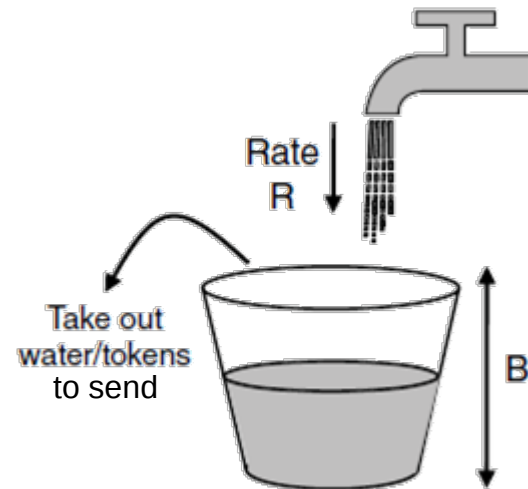
# Traffic Shaping (2)

Token/Leaky bucket limits both the average rate ( $R$ ) and short-term burst ( $B$ ) of traffic

- For token, bucket size is  $B$ , water enters at rate  $R$  and is removed to send; opposite for leaky.

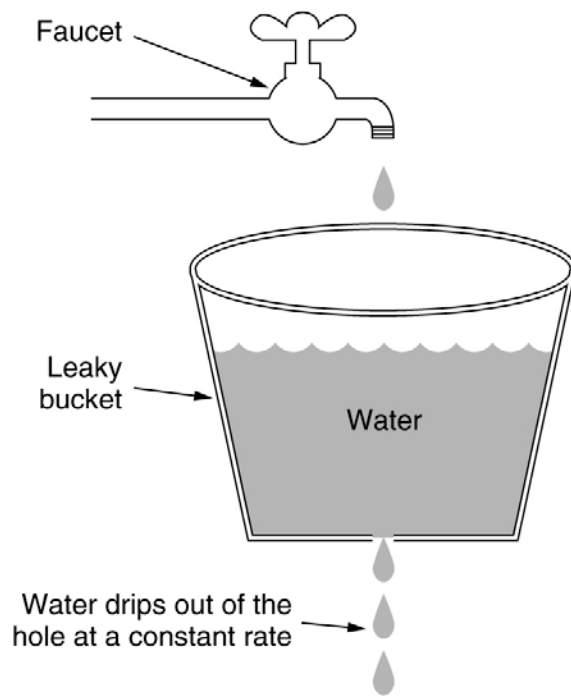


Leaky bucket  
(need not full to send)

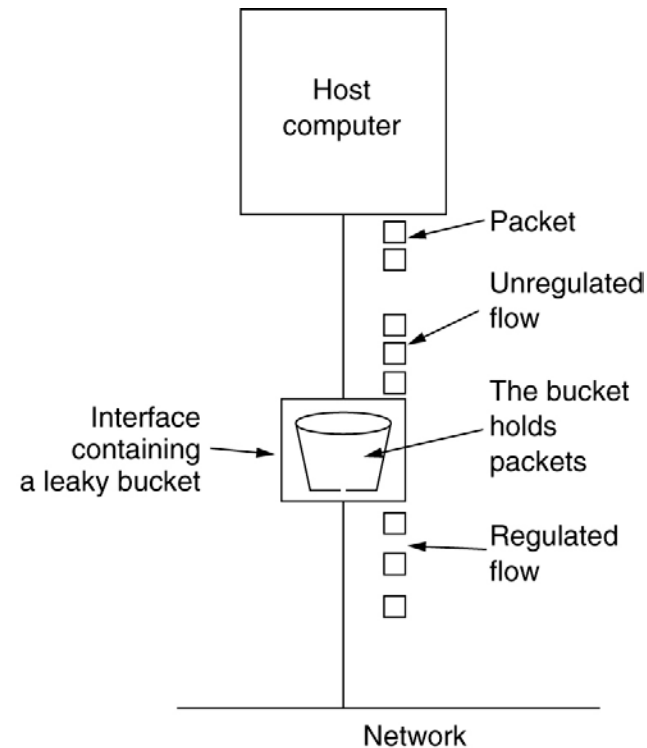


Token bucket  
(need some water to send)

# The Leaky Bucket Algorithm



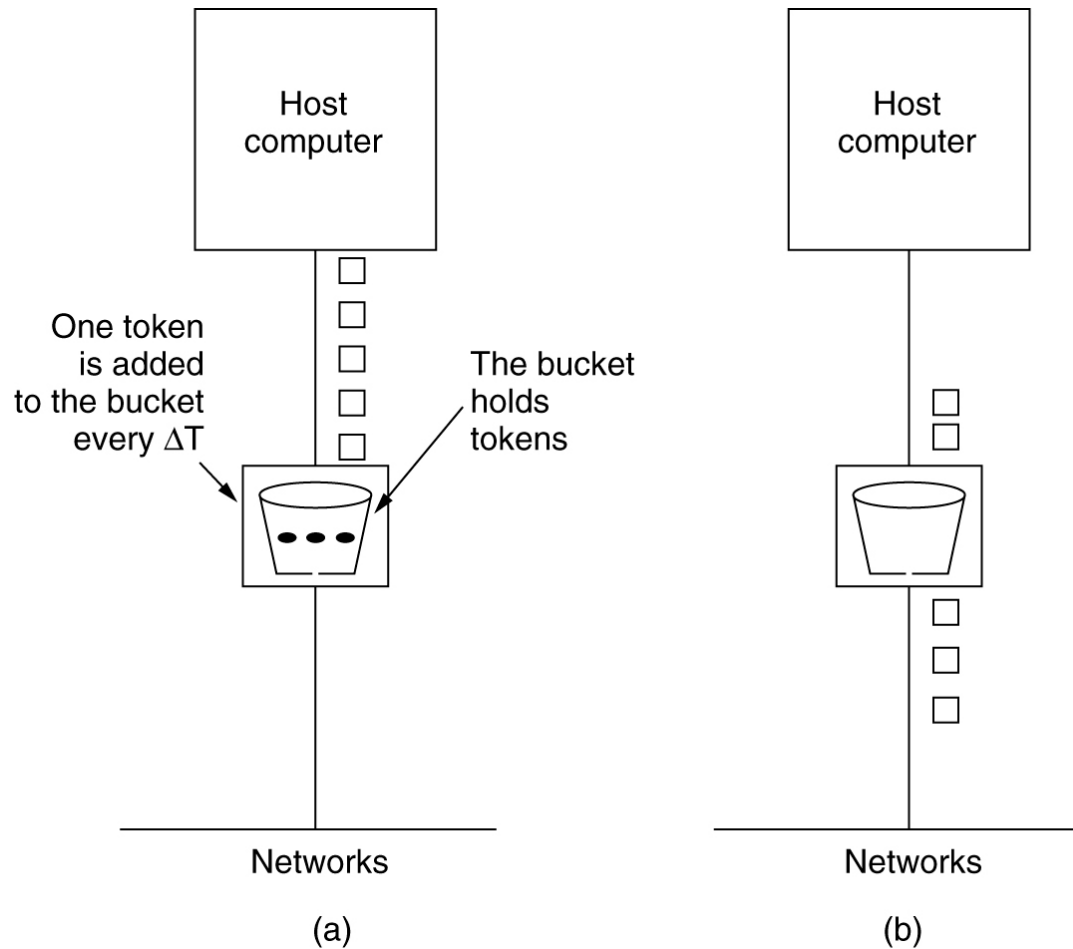
(a)



(b)

(a) A leaky bucket with water. (b) a leaky bucket with packets.

# The Token Bucket Algorithm



(a) Before. (b) After.

# Traffic Shaping (3)

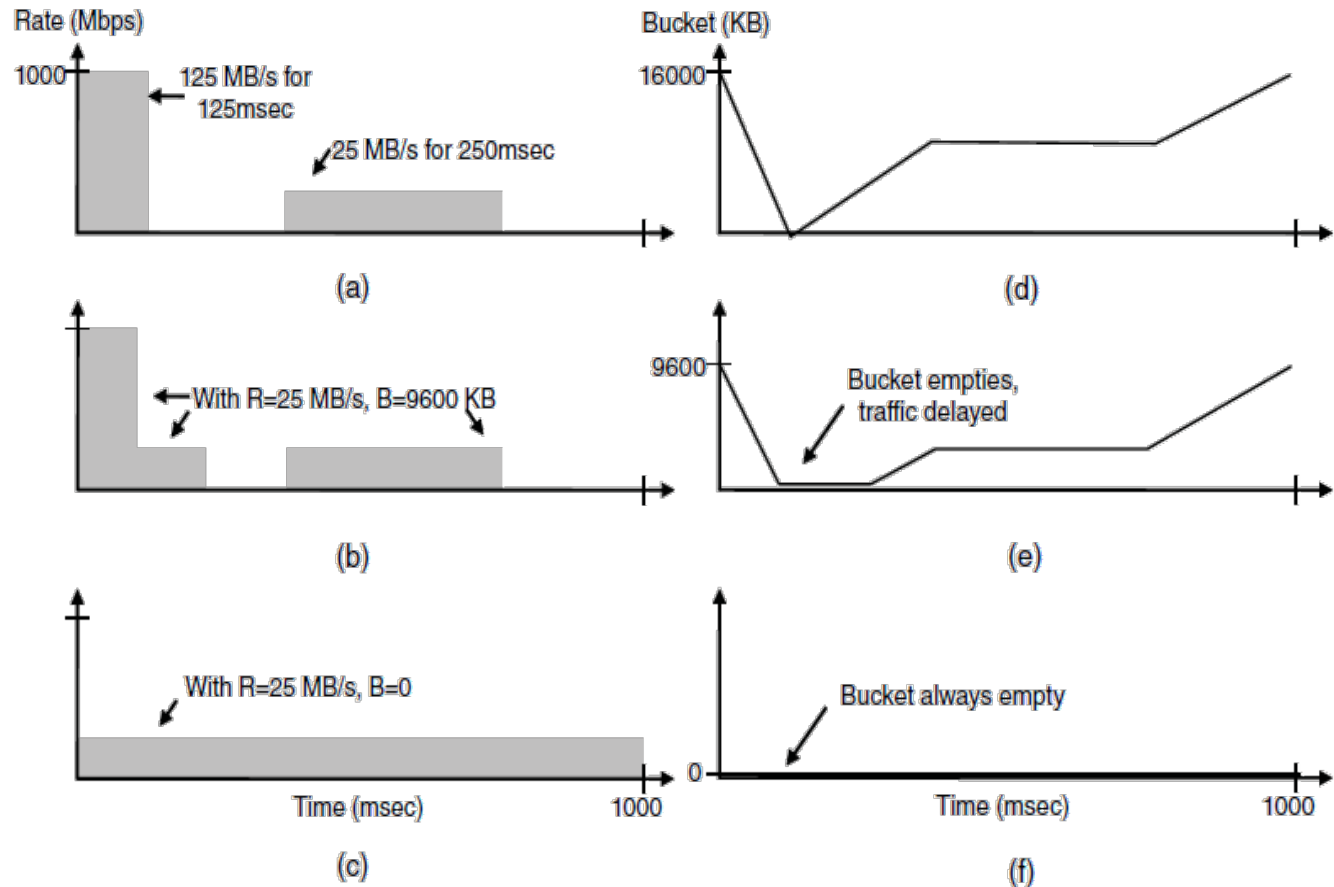
Host traffic  
 $R=200$  Mbps  
 $B=16000$  KB



Shaped by  
 $R=200$  Mbps  
 $B=9600$  KB



Shaped by  
 $R=200$  Mbps  
 $B=0$  KB



Smaller bucket size delays traffic and reduces burstiness

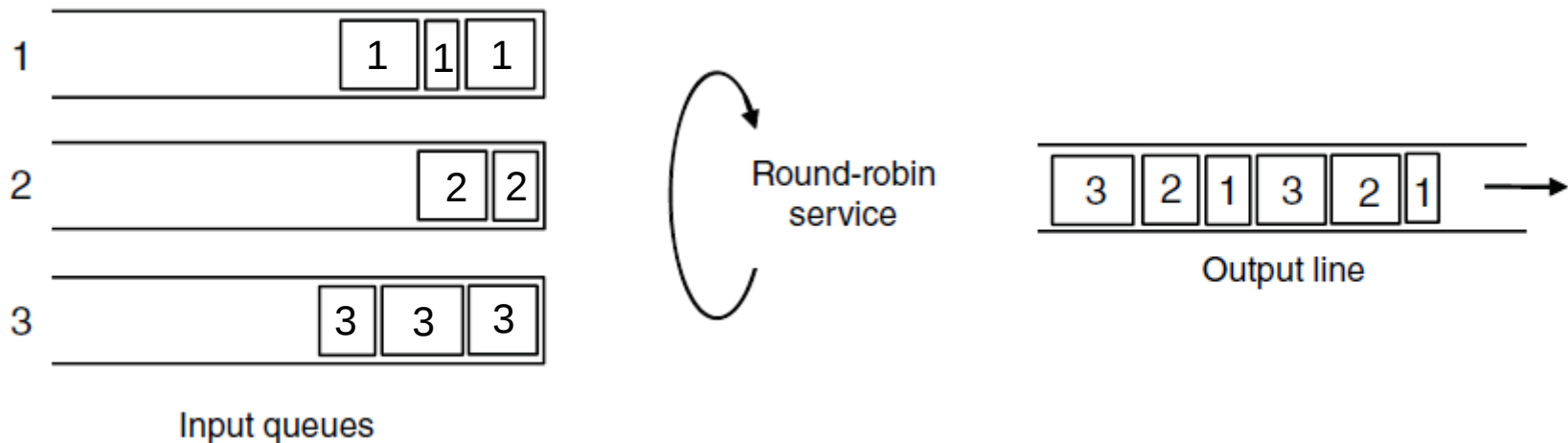
# Packet Scheduling (1)

Kinds of resources can potentially be reserved for different flows:

1. Bandwidth.
2. Buffer space.
3. CPU cycles.

# Packet Scheduling (1)

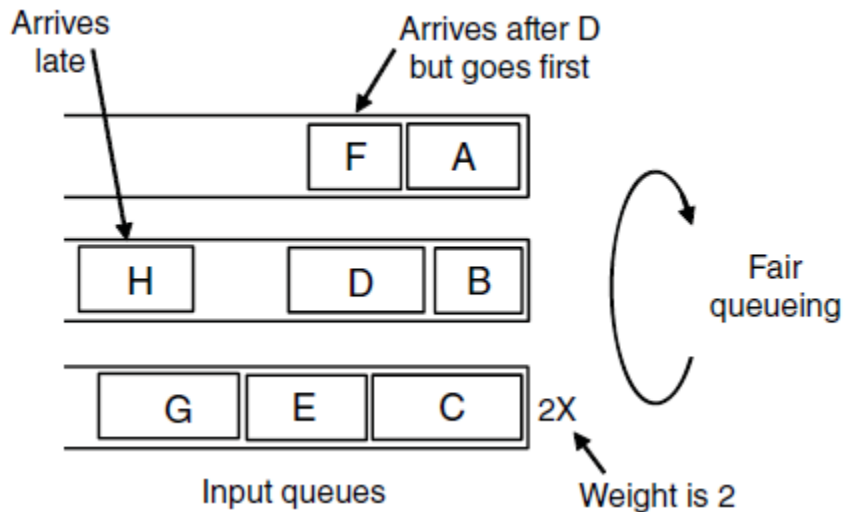
Packet scheduling divides router/link resources among traffic flows with alternatives to FIFO (First In First Out)



Example of round-robin queuing

# Packet Scheduling (2)

Fair Queueing approximates bit-level fairness with different packet sizes; weights change target levels



Packets may be sent out of arrival order

| Packet | Arrival time | Length | Finish time | Output order |
|--------|--------------|--------|-------------|--------------|
| A      | 0            | 8      | 8           | 1            |
| B      | 5            | 6      | 11          | 3            |
| C      | 5            | 10     | 10          | 2            |
| D      | 8            | 9      | 20          | 7            |
| E      | 8            | 8      | 14          | 4            |
| F      | 10           | 6      | 16          | 5            |
| G      | 11           | 10     | 19          | 6            |
| H      | 20           | 8      | 28          | 8            |

$$F_i = \max(A_i, F_{i-1}) + L_i/W$$

Finish virtual times determine transmission order

# Admission Control (1)

Admission control takes a traffic flow specification and decides whether the network can carry it

- Sets up packet scheduling to meet QoS

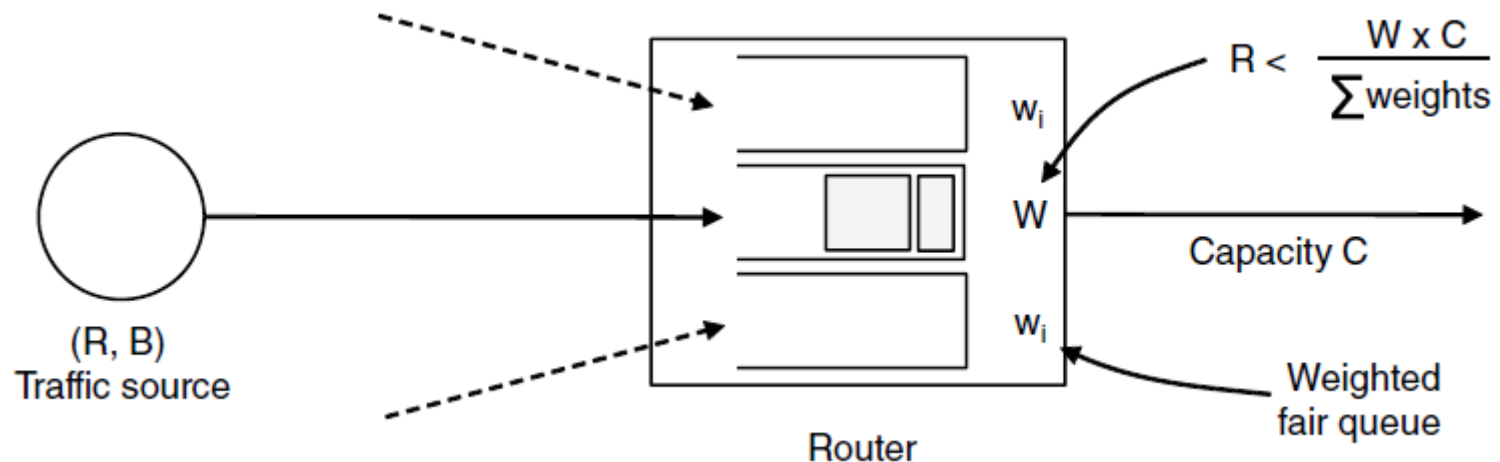
| Parameter           | Unit      |
|---------------------|-----------|
| Token bucket rate   | Bytes/sec |
| Token bucket size   | Bytes     |
| Peak data rate      | Bytes/sec |
| Minimum packet size | Bytes     |
| Maximum packet size | Bytes     |

Example flow specification

# Admission Control (2)

Construction to guarantee bandwidth  $B$  and delay  $D$ :

- Shape traffic source to a  $(R, B)$  token bucket
- Run WFQ with weight  $W$  / all weights  $> R/\text{capacity}$
- Holds for all traffic patterns, all topologies



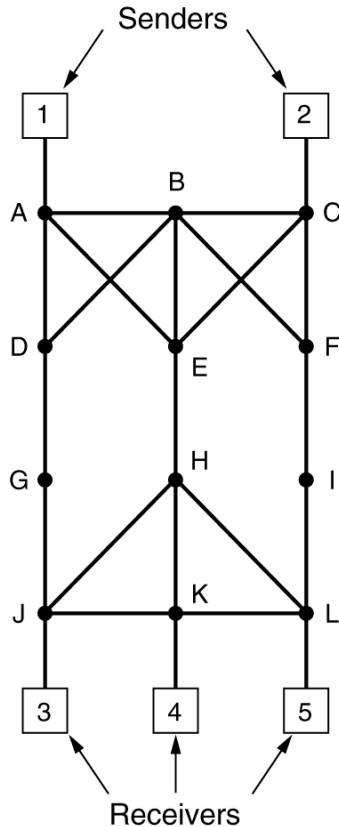
# Integrated Services (1)

Design with QoS for each flow; handles multicast traffic.

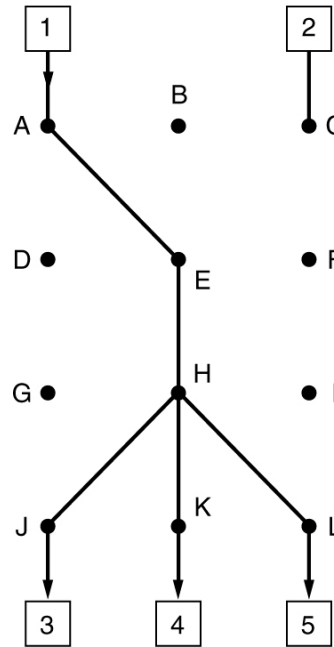
Admission with RSVP (Resource reSerVation Protocol):

- Receiver sends a request back to the sender
- Each router along the way reserves resources
- Routers merge multiple requests for same flow
- Entire path is set up, or reservation not made

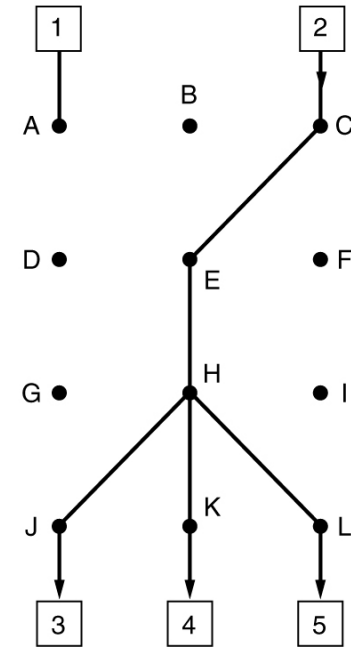
# RSVP-The ReSerVation Protocol



(a)



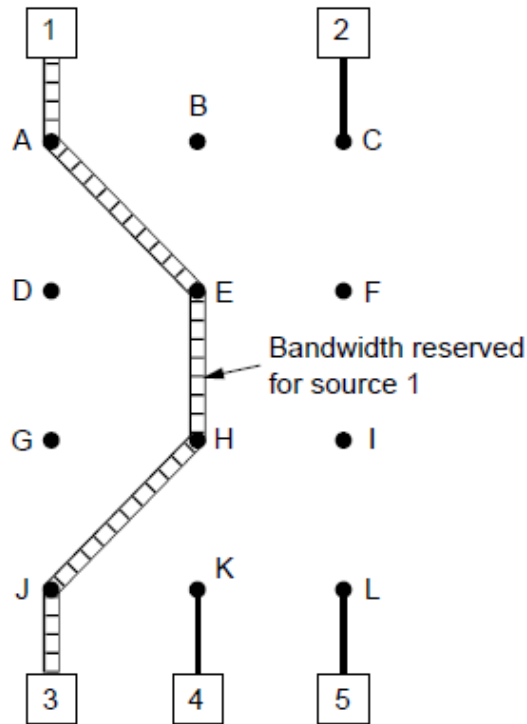
(b)



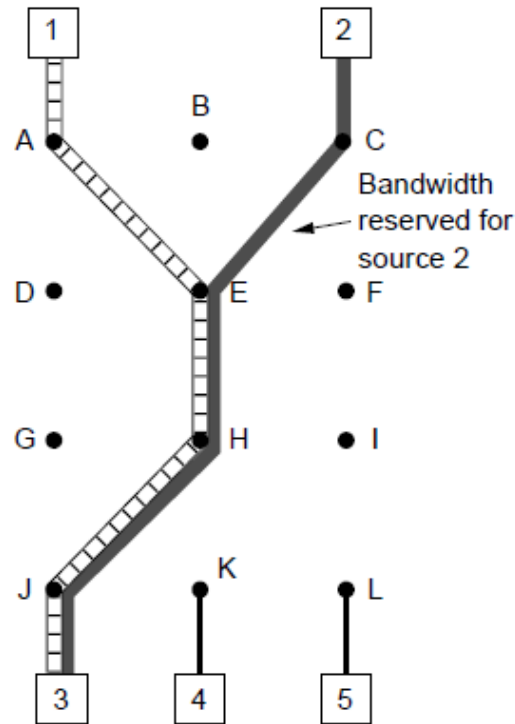
(c)

- (a) A network, (b) The multicast spanning tree for host 1.  
 (c) The multicast spanning tree for host 2.

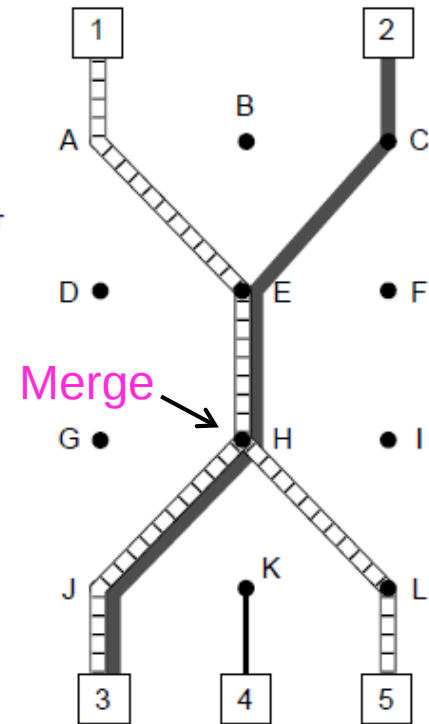
# Integrated Services (2)



R3 reserves flow from S1



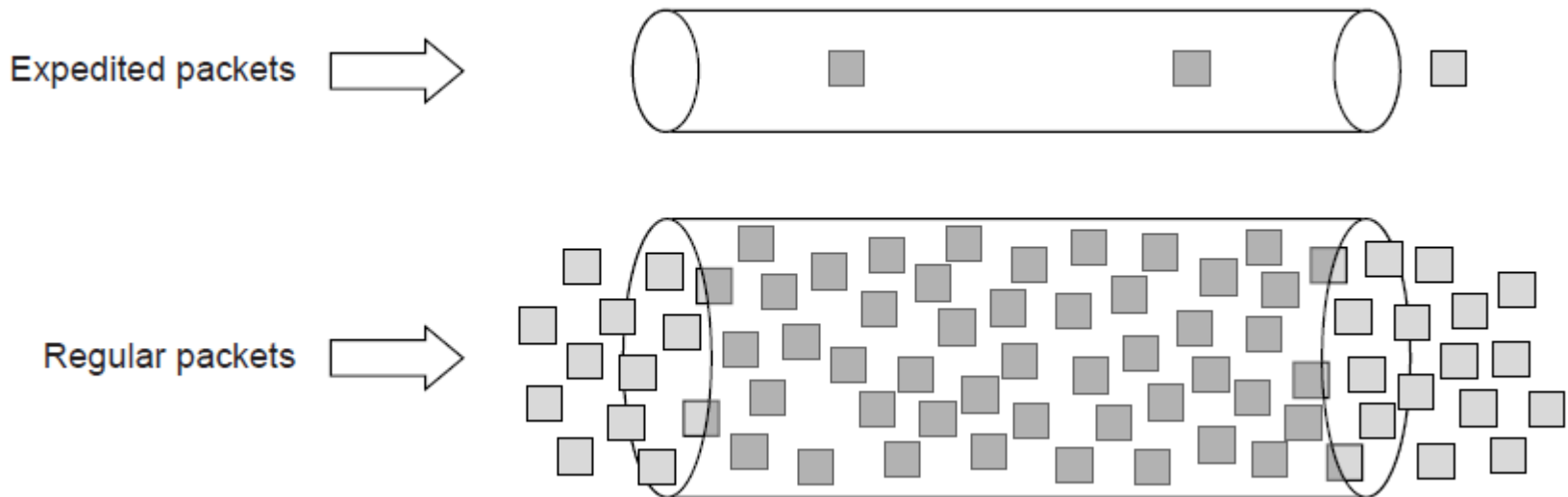
R3 reserves flow from S2



R5 reserves flow from S1; merged with R3 at H

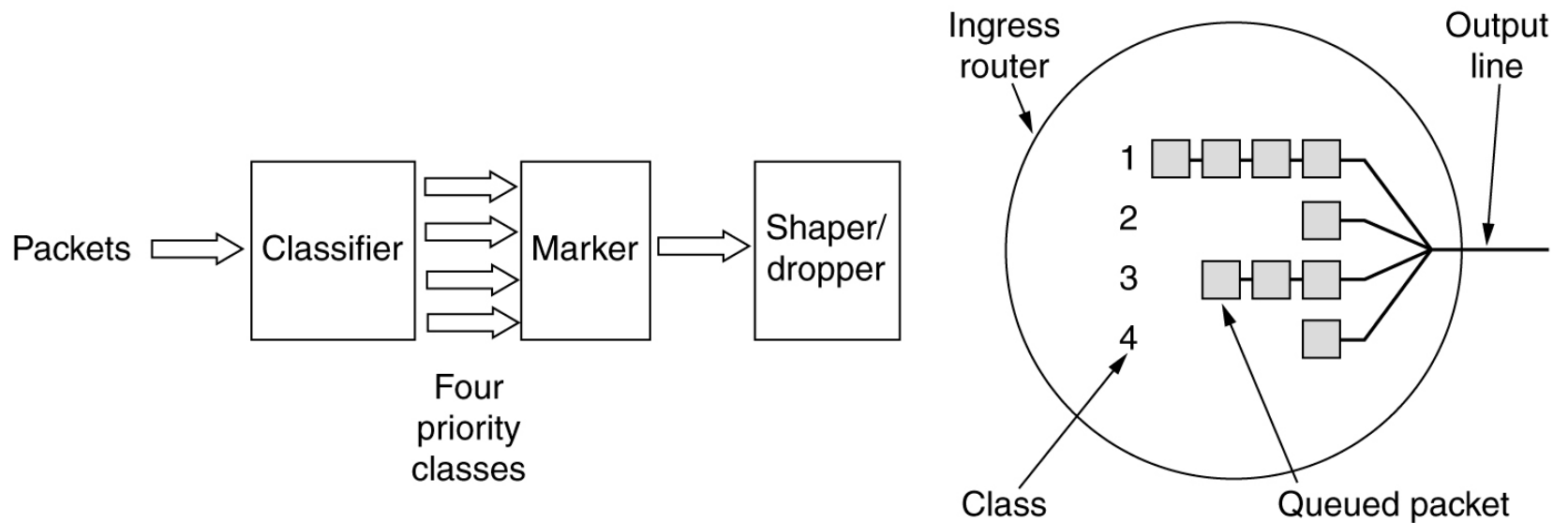
# Differentiated Services (1)

- Design with classes of QoS; customers buy what they want
  - Expedited class is sent in preference to regular class
  - Less expedited traffic but better quality for applications



# Assured Forwarding

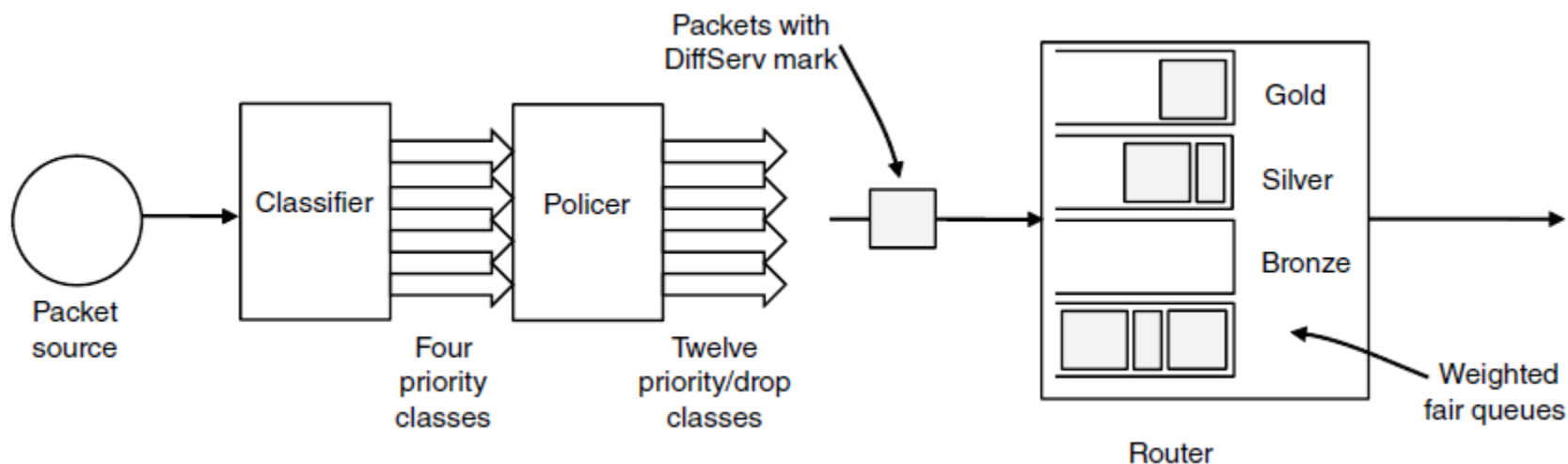
A possible implementation of the data flow for assured forwarding.



# Differentiated Services (2)

## Implementation of DiffServ:

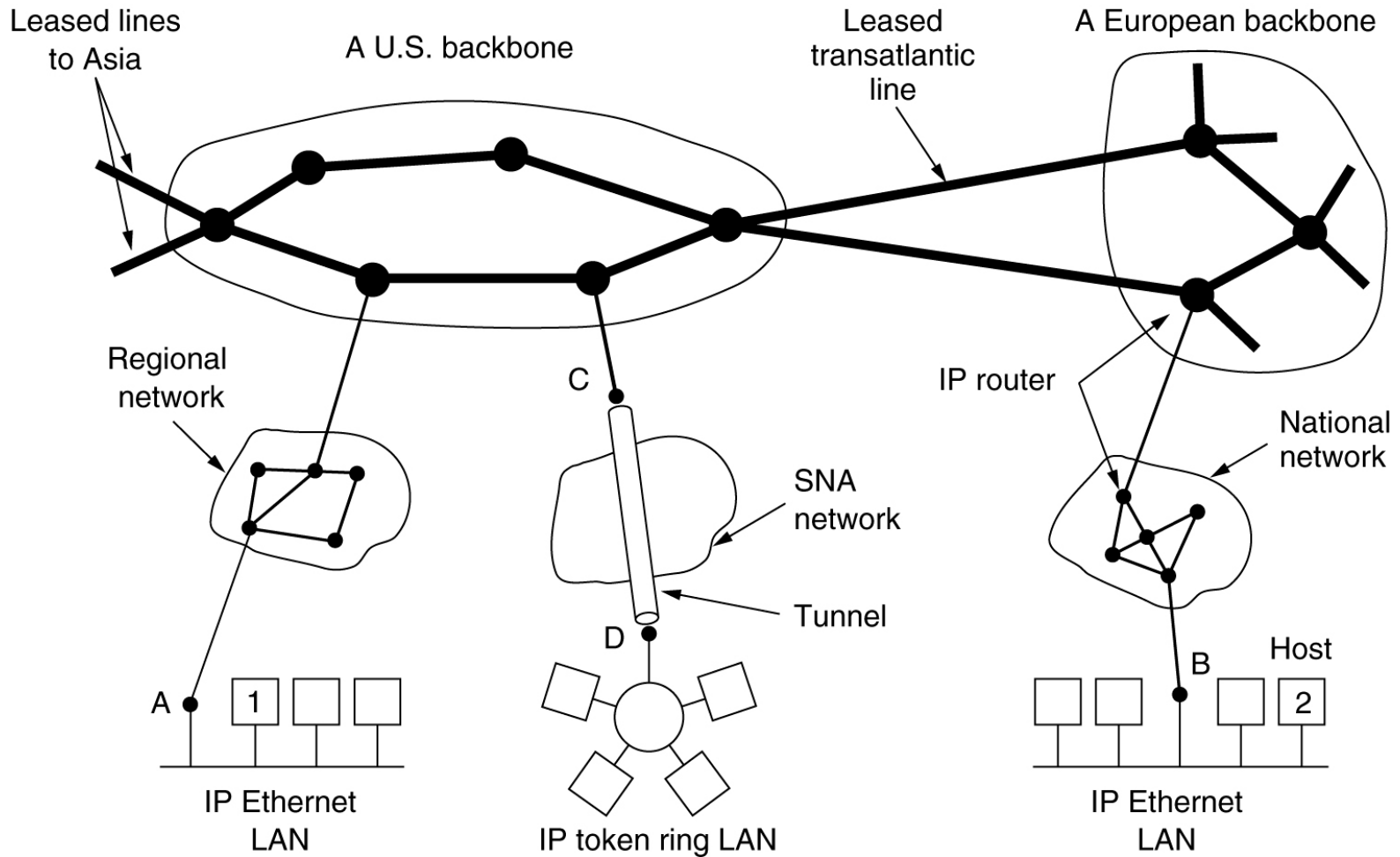
- Customers mark desired class on packet
- ISP shapes traffic to ensure markings are paid for
- Routers use WFQ to give different service levels



# Design Principles for Internet

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.

# Collection of Subnetworks



The Internet is an interconnected collection of many networks.

# The Network Layer in the Internet

- The IP Protocol
- IP Addresses
- Internet Control Protocols
- OSPF – The Interior Gateway Routing Protocol
- BGP – The Exterior Gateway Routing Protocol
- Internet Multicasting
- Mobile IP
- IPv6

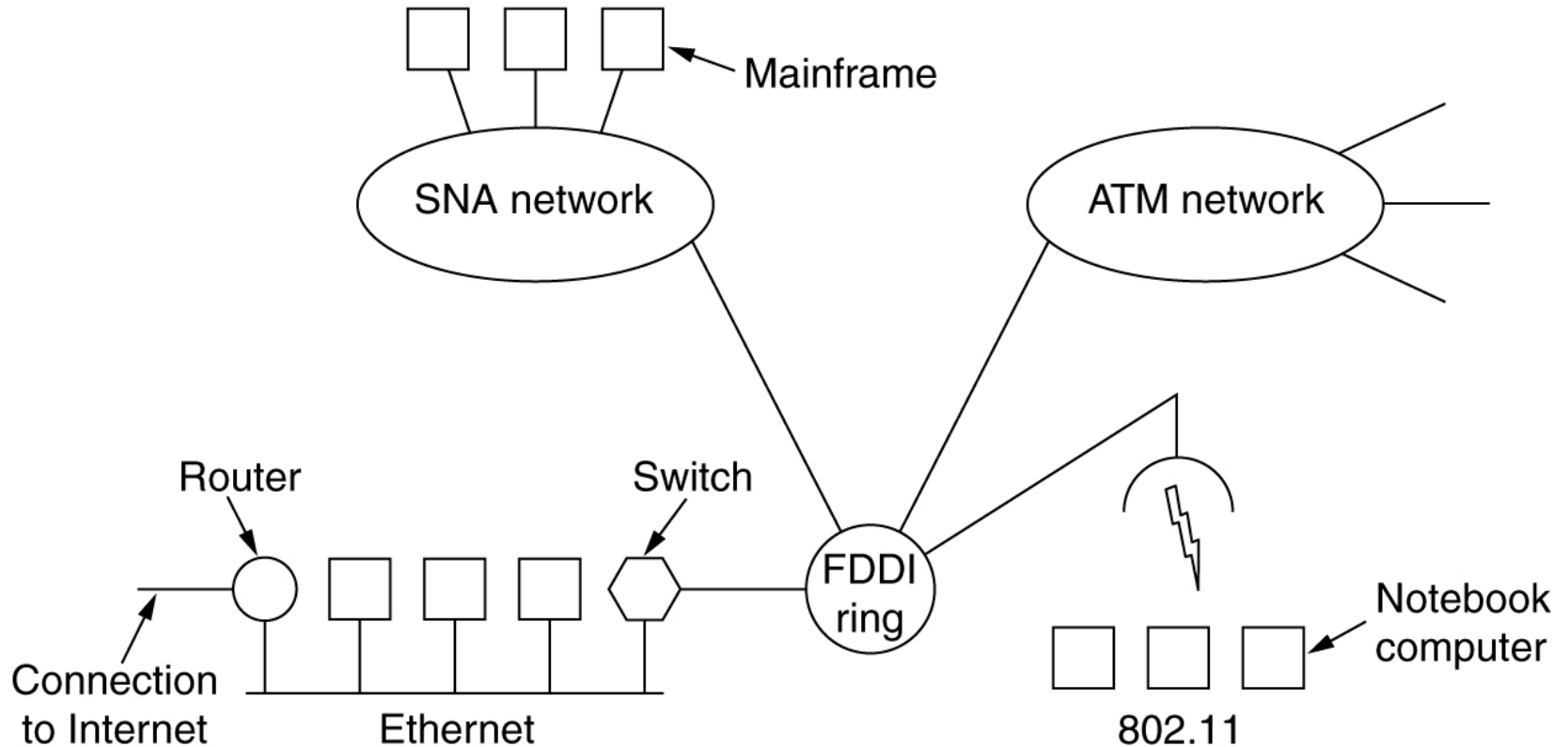
# Internetworking

Internetworking joins multiple, different networks into a single larger network

- How networks differ »
- How networks can be connected »
- Tunneling »
- Internetwork routing »
- Packet fragmentation »

# Connecting Networks

A collection of interconnected networks.



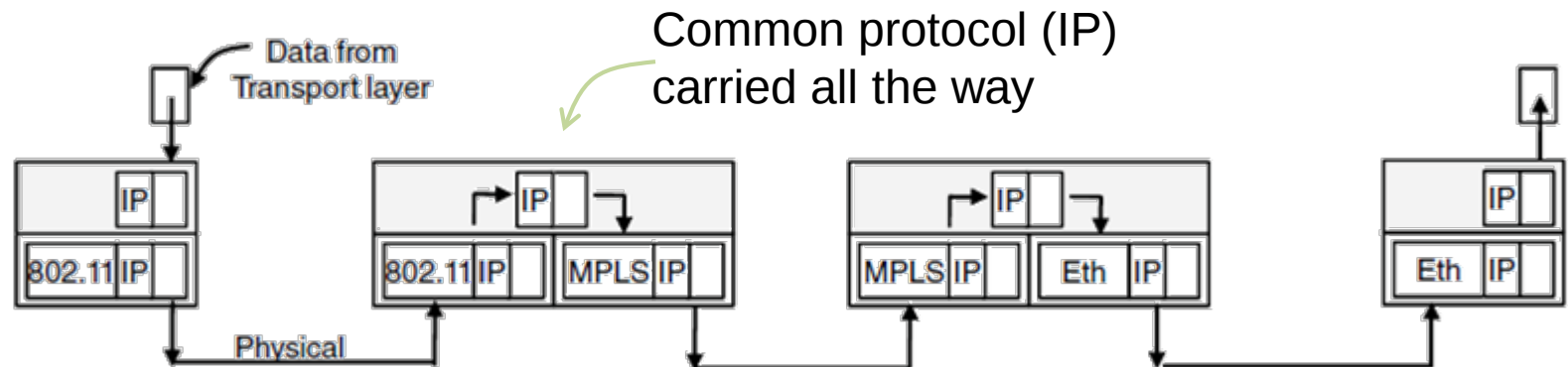
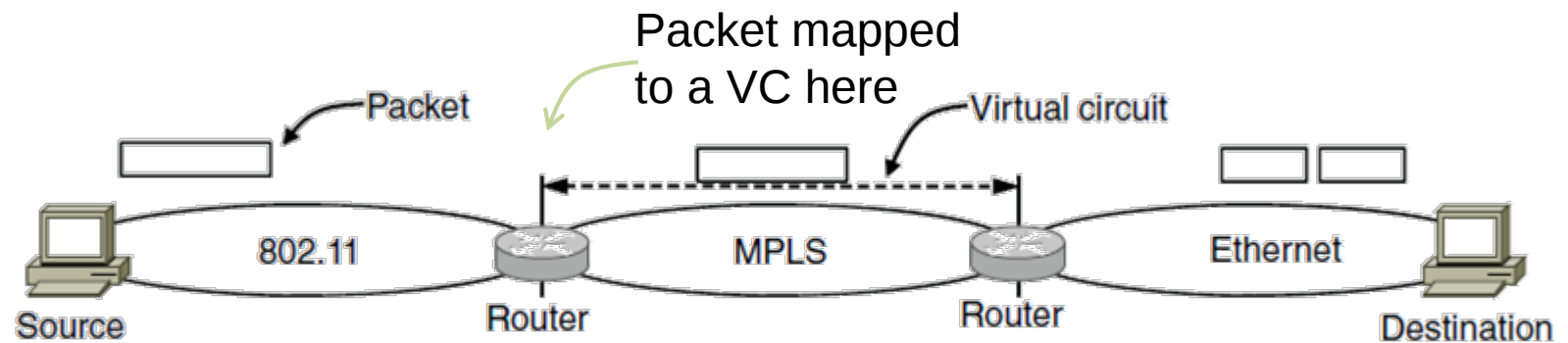
# How Networks Differ

| 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   |

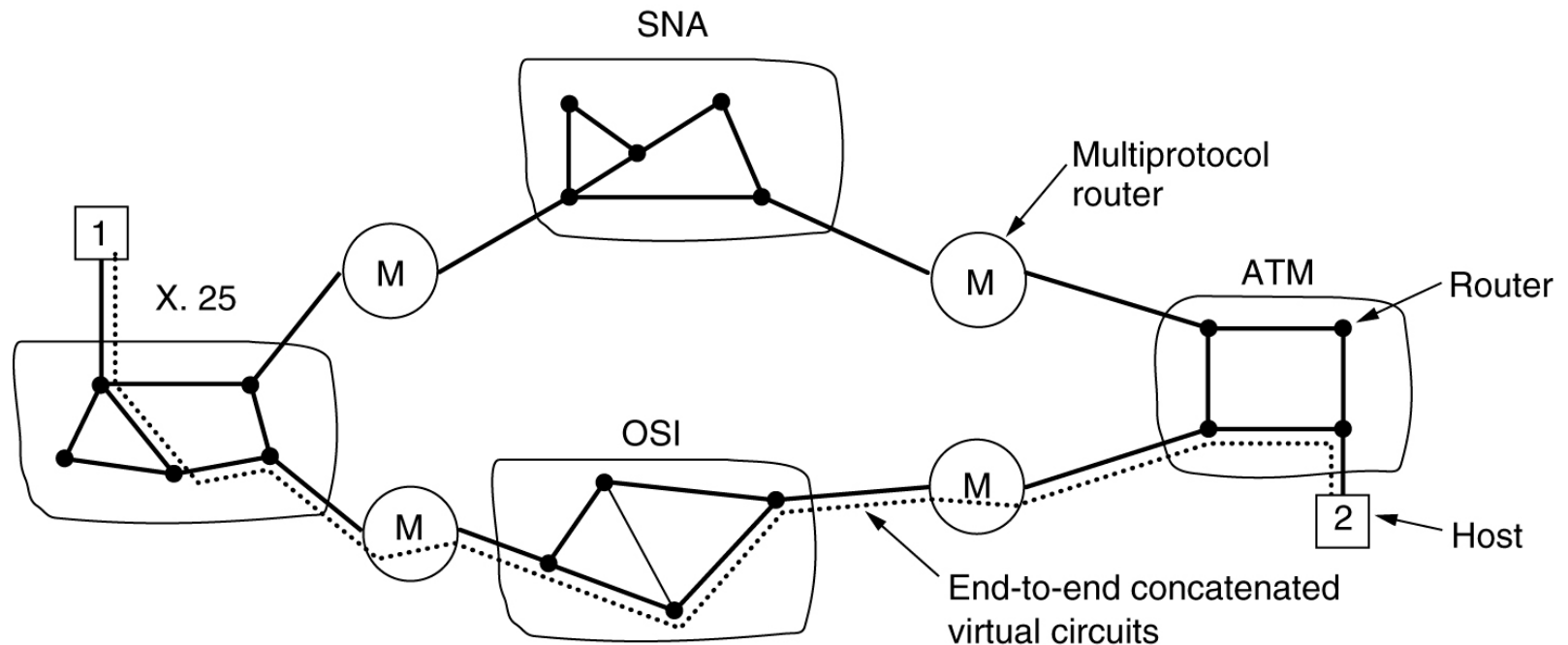
Some of the many ways networks can differ.

# How Networks Can Be Connected

Internetworking based on a common network layer – IP

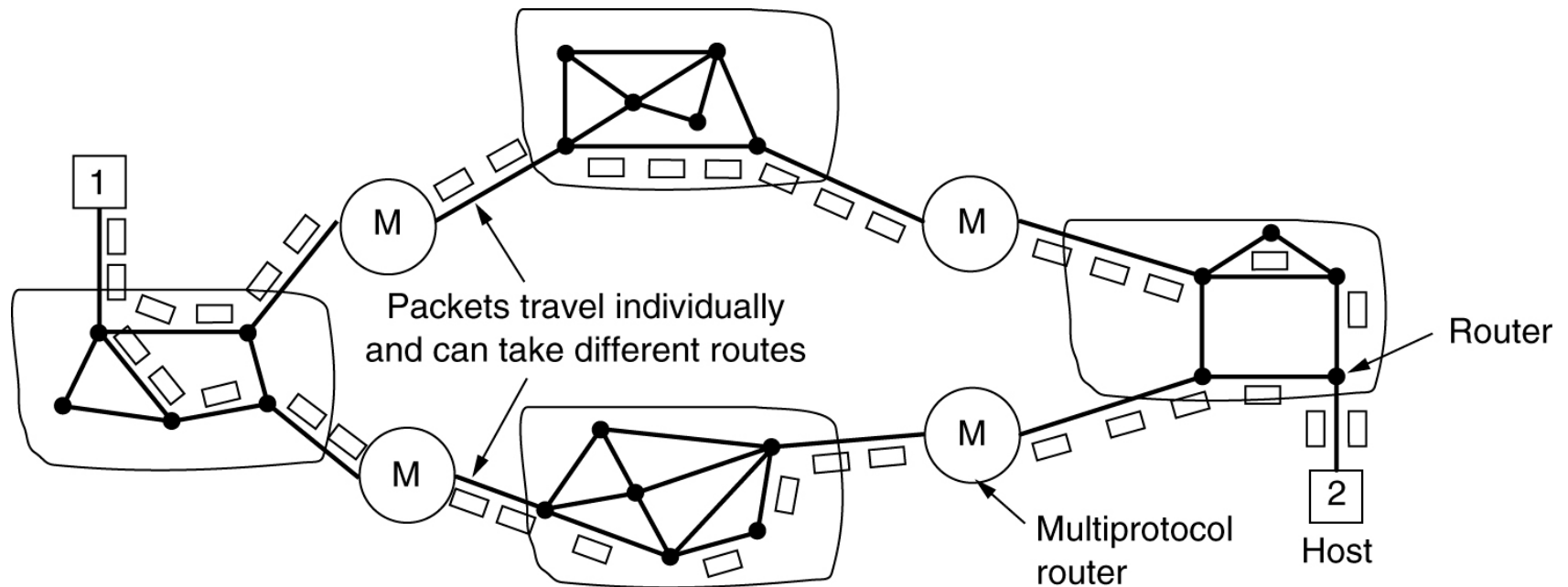


# Concatenated Virtual Circuits



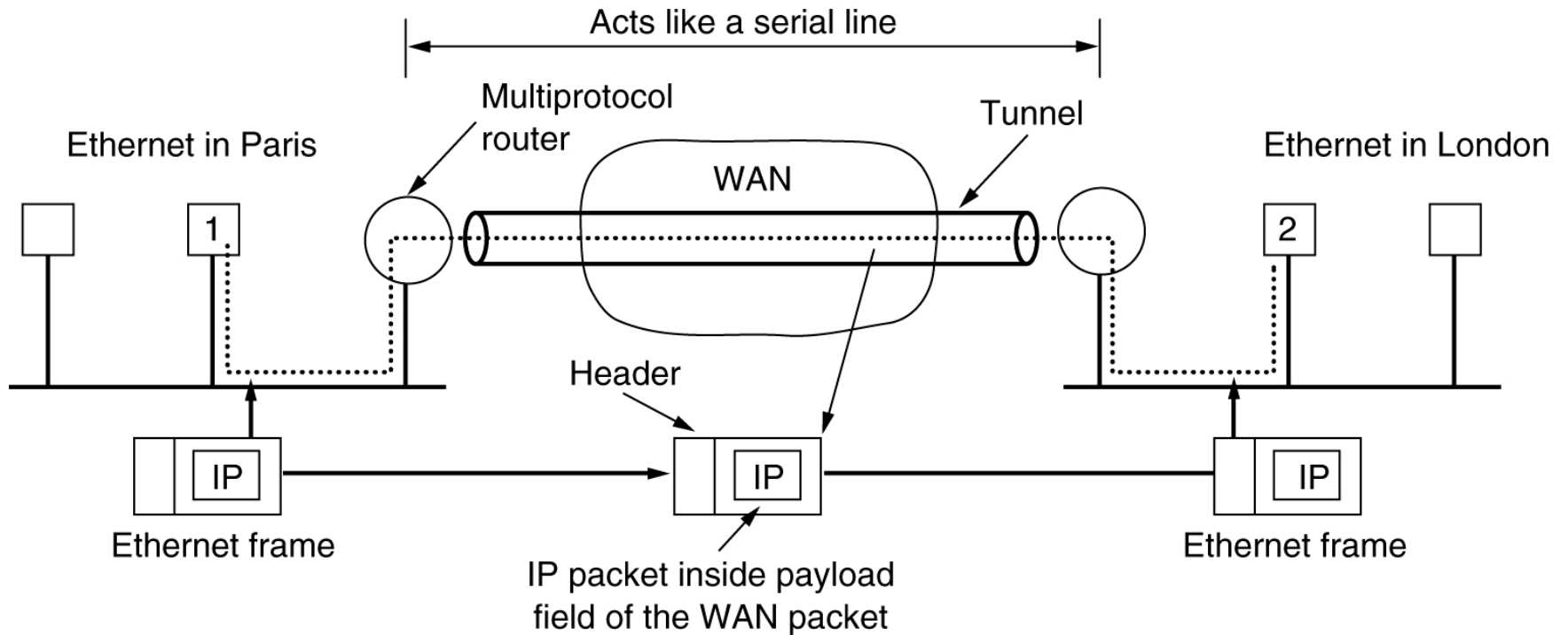
Internetworking using concatenated virtual circuits.

# Connectionless Internetworking



A connectionless internet.

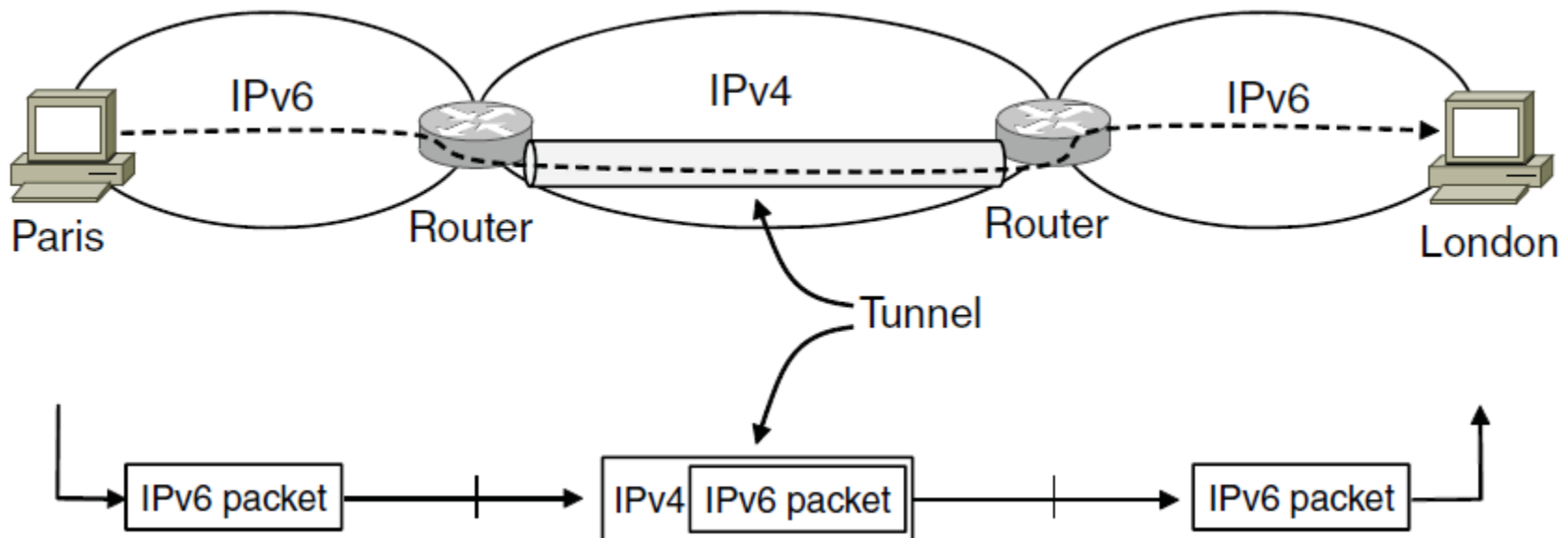
# Tunneling



# Tunneling (1)

Connects two networks through a middle one

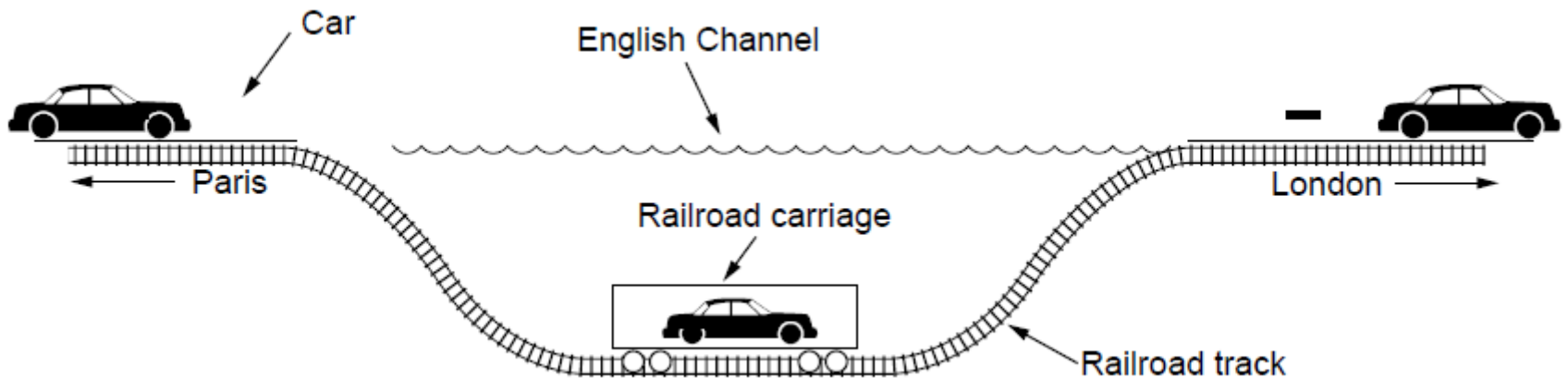
- Packets are encapsulated over the middle



# Tunneling (2)

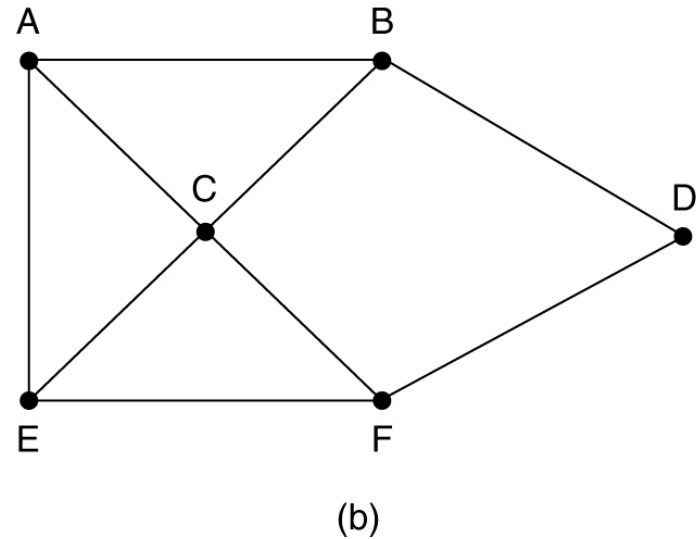
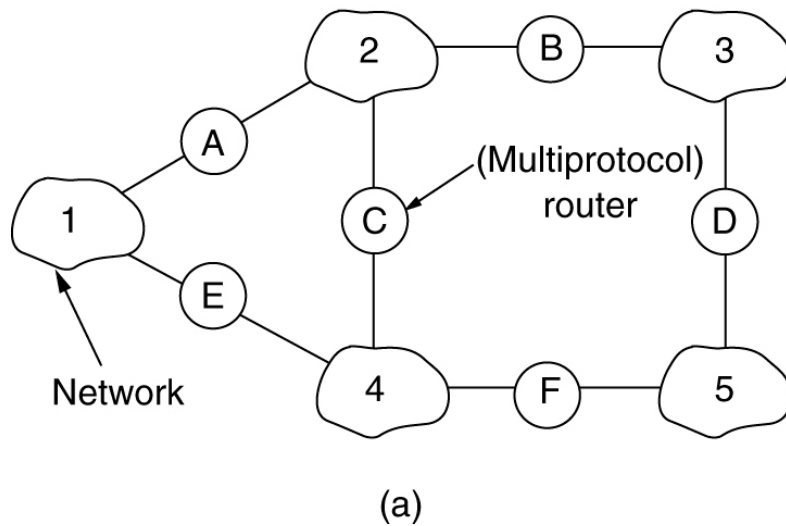
Tunneling analogy:

- tunnel is a link; packet can only enter/exit at ends



# Internetwork Routing

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



# Network Layer in the Internet (1)

- IP Version 4 »
- IP Addresses »
- IP Version 6 »
- Internet Control Protocols »
- Label Switching and MPLS »
- OSPF—An Interior Gateway Routing Protocol »
- BGP—The Exterior Gateway Routing Protocol »
- Internet Multicasting »
- Mobile IP »



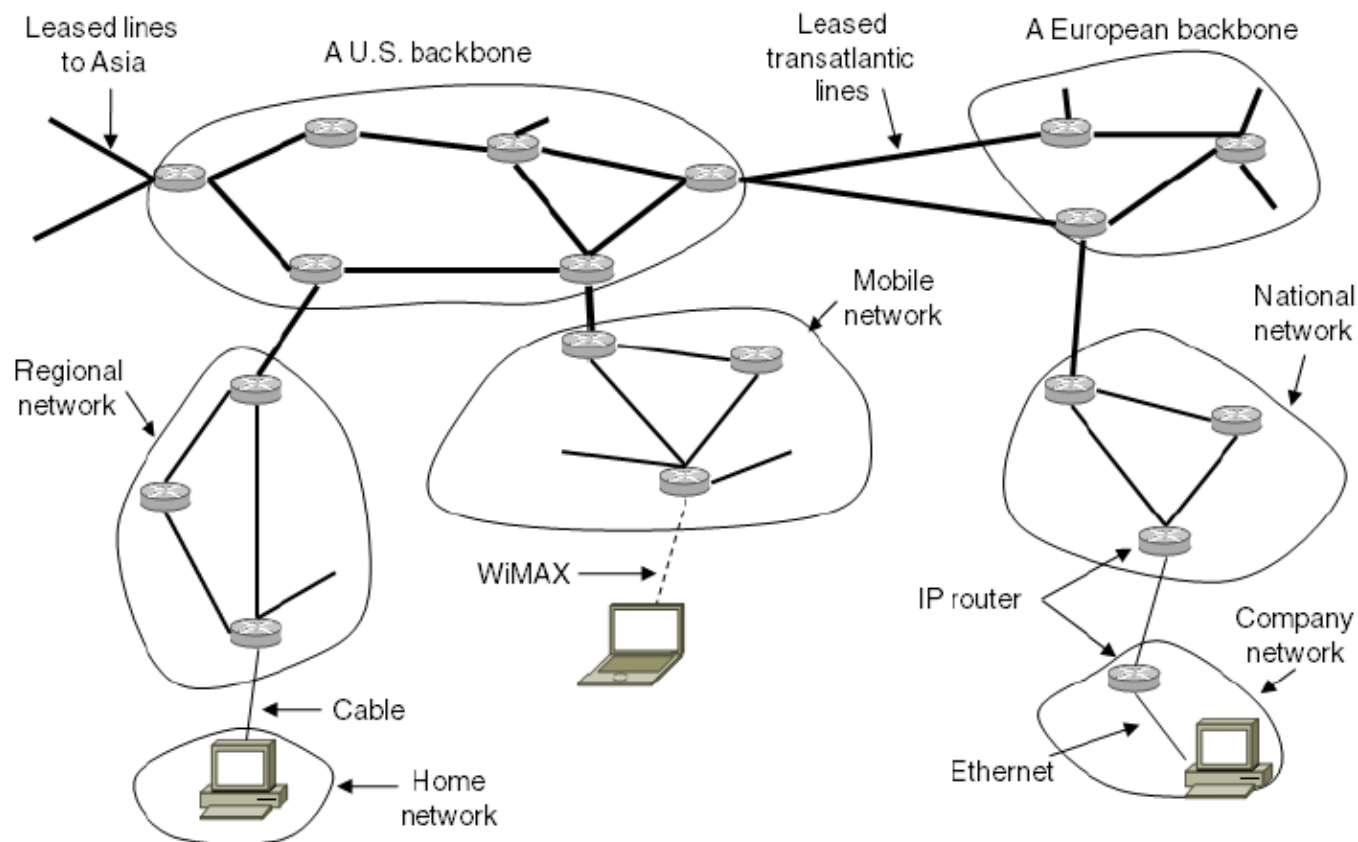
# Network Layer in the Internet (2)

IP has been shaped by guiding principles:

- Make sure it works
- Keep it simple
- Make clear choices
- Exploit modularity
- Expect heterogeneity
- Avoid static options and parameters
- Look for good design (not perfect)
- Strict sending, tolerant receiving
- Think about scalability
- Consider performance and cost

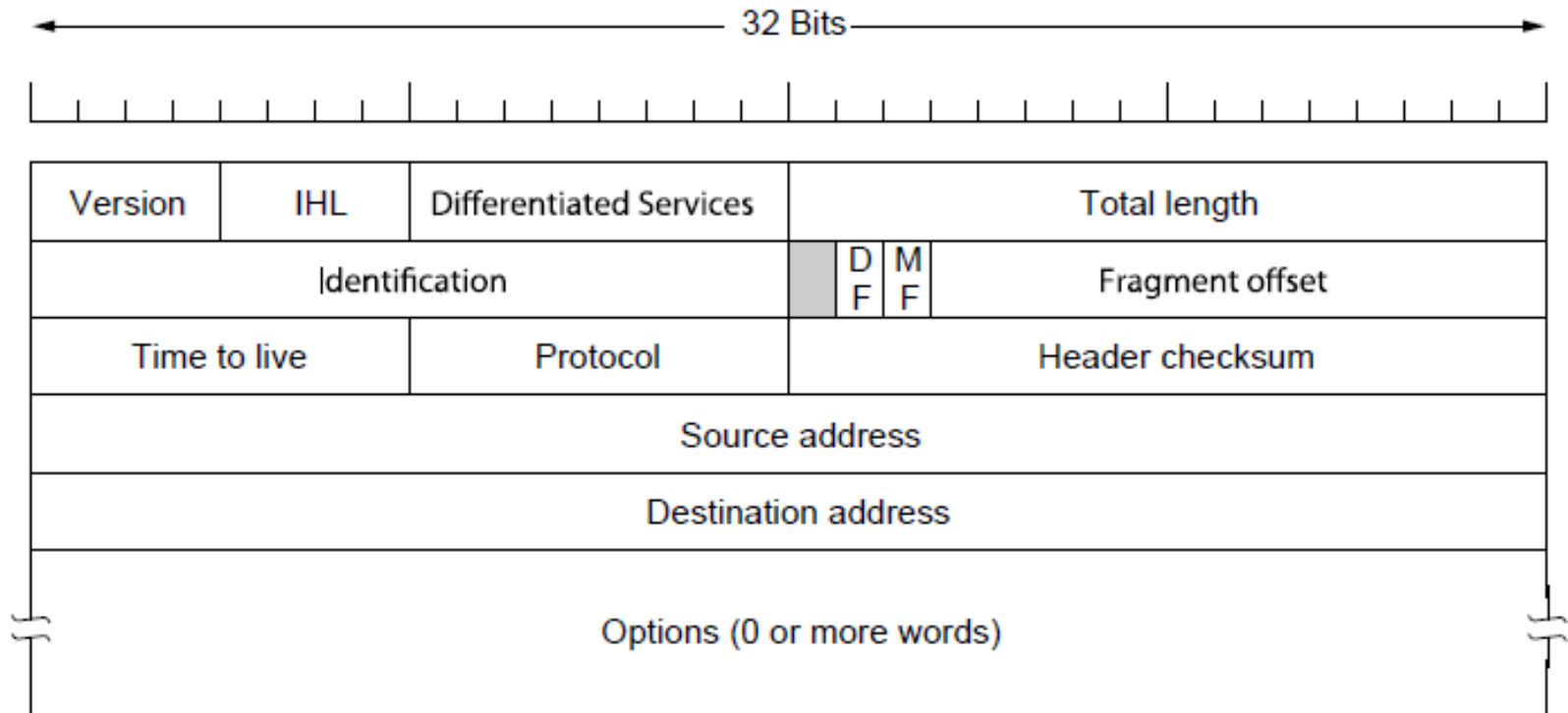
# Network Layer in the Internet (3)

- Internet is an interconnected collection of many networks that is held together by the IP protocol



# IP Version 4 Protocol (1)

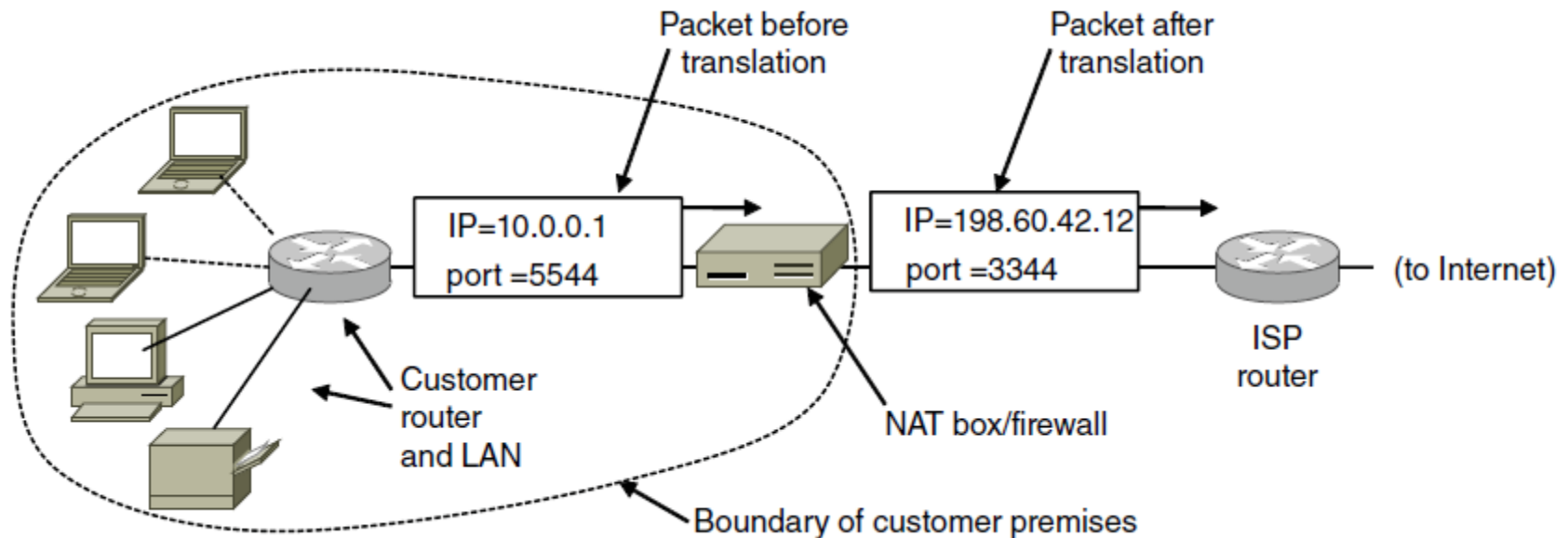
IPv4 (Internet Protocol) header is carried on all packets and has fields for the key parts of the protocol:



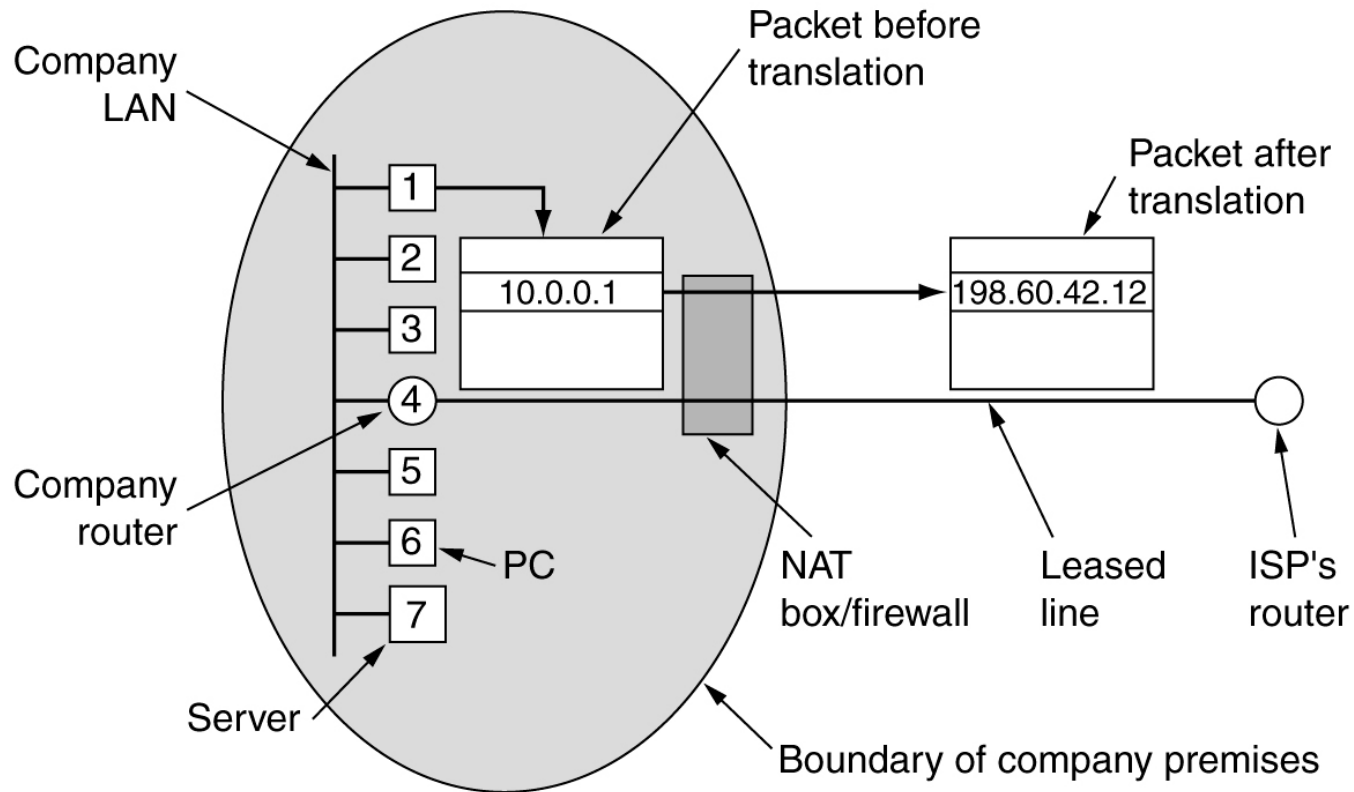
# IP Addresses (6) – NAT

NAT (Network Address Translation) box maps one external IP address to many internal IP addresses

- Uses TCP/UDP port to tell connections apart
- Violates layering; very common in homes, etc.



# NAT – Network Address Translation



# IPv6 - Introduction

## Basics

# What is IPv6

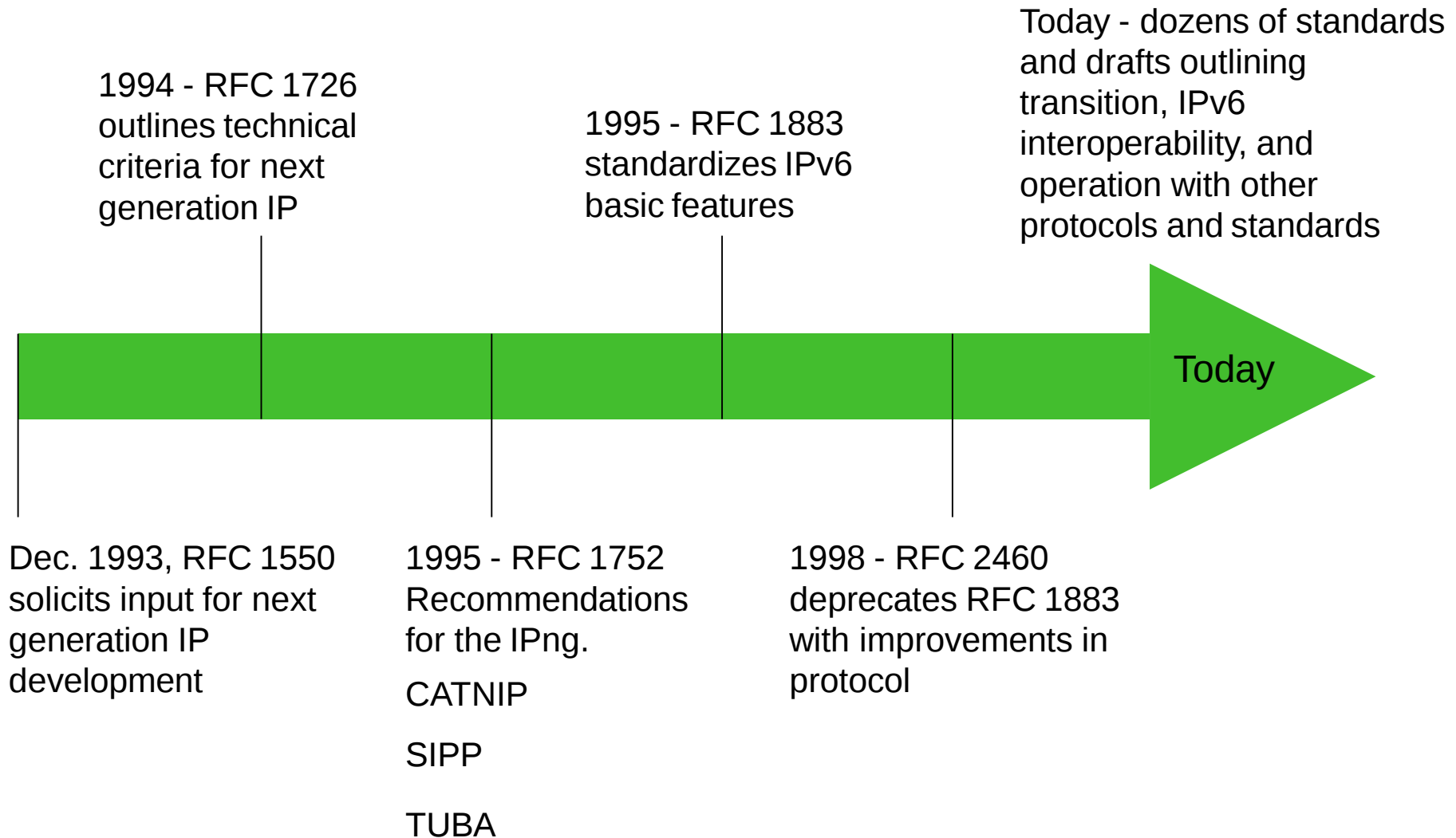
- Also known as IPng (next generation)
- A new version of the Internet Protocol
  - Primarily designed to extend address space
  - Enhancements and new features

# Brief History of the Internet\*

- First node of DARPA NET established 1969
- 1972 electronic mail (e-mail) was invented
- 1974 TCP/IP invented by Kahn/Cerf
- 1980-1994 massive growth in use of “networks” beyond Military/University
- 1991 World Wide Web developed
- 1993 Mosaic is released

\*Source: [www.netvalley.com](http://www.netvalley.com)

# History of IPv6

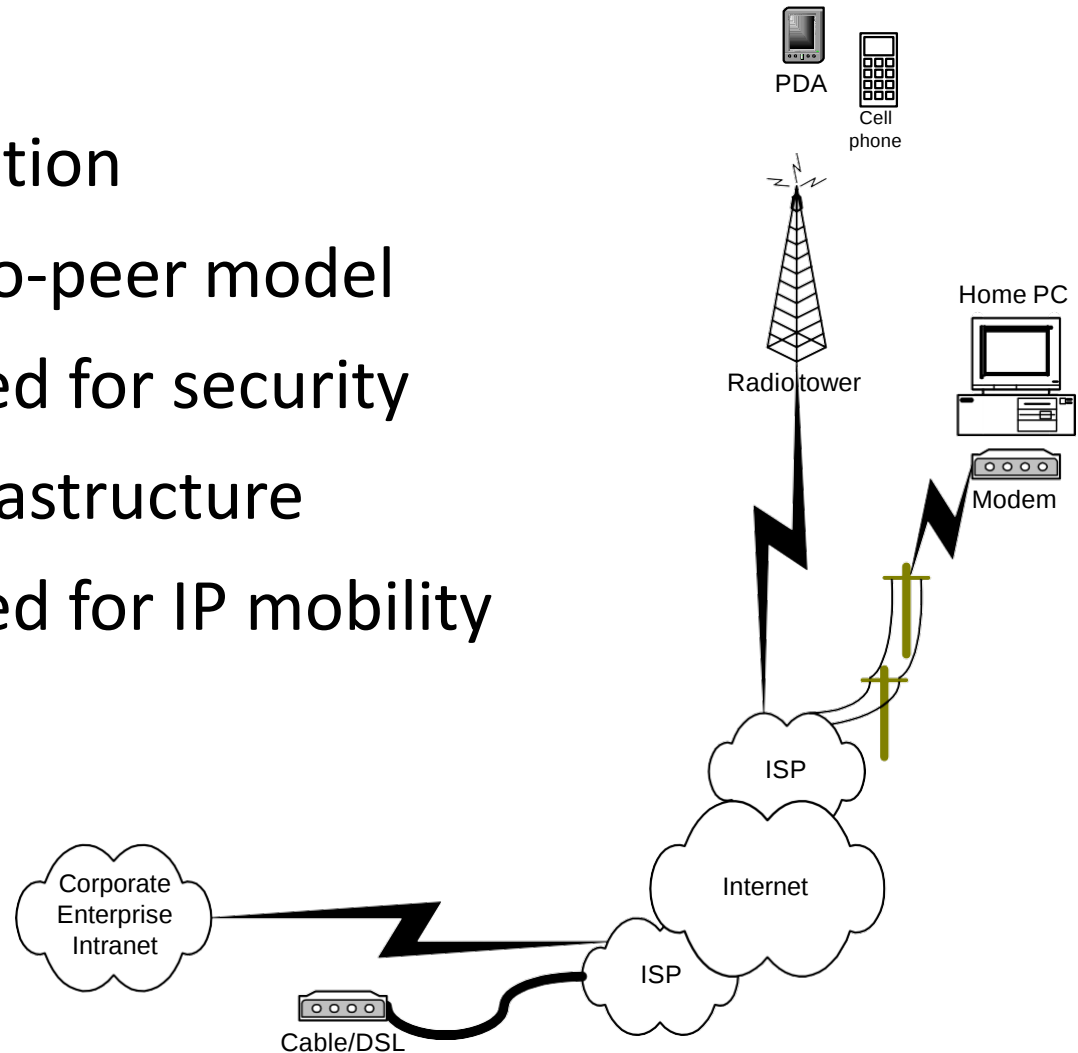


# History of IPv6

- What happened to IPv5
  - Version 5 in IP header was assigned to ST protocol (a.k.a, Internet Streaming Protocol)
  - Experimental non-IP real-time streaming protocol
  - Never widely used
  - RFC 1819

# Challenges in Today's Internet

- Address depletion
- Loss of peer-to-peer model
- Increasing need for security
- Burdened Infrastructure
- Increasing need for IP mobility



# Challenges in Today's Internet

- Increased traffic flow
- New applications have specific delivery requirements
- Increasing need for security and information protection
- Varied usage of Internet requires continual “tweaking” of current Internet protocols

# Challenges in Today's Internet

- IPv4 growth and adoption phenomenal
  - Exhausting address space
  - Internet routing table very big
- IPv4 Lack of inherent features
  - Quality of Service (QoS)
  - Security
- IPv4 failings
  - Broadcast
  - Uncontrolled Packet Fragmentation

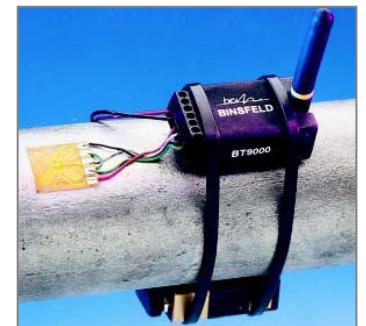
# Challenges in Today's Internet

- QOS Issues
  - Multiple methods available
  - Non-uniformity across network boundaries
- Security
  - No uniformity, myriad methods
  - Multitude of methods for “hackers” to attack networks

# Challenges in Today's Internet

- Mobility
  - Increasing number of wireless/mobile devices accessing Internet services
  - Inadequate support for IP mobility in devices (moving seamlessly from one network to another)
  - 3G & 4G initiatives, 802.11x, Bluetooth, UWB moving connectivity from wireline to wireless

# What IP is touching



# Use of IP in Telecommunications

- Telecommunication carriers extending circuit usage with VoIP/VoDS.
  - July, 2002 Verizon announces move to VoATM
  - Feb, 2001 AT&T launches VoIP portfolio
  - Oct., 2001 Qwest announces it will replace all circuit switched telephony to VoIP
- Increasing number of mobile users accessing web-based services/applications

# Why is IPv6 Here

- IPv6 provides a platform for new Internet functionality that will be needed in the immediate future, and provide flexibility for further growth and expansion.

# Features of IPv6

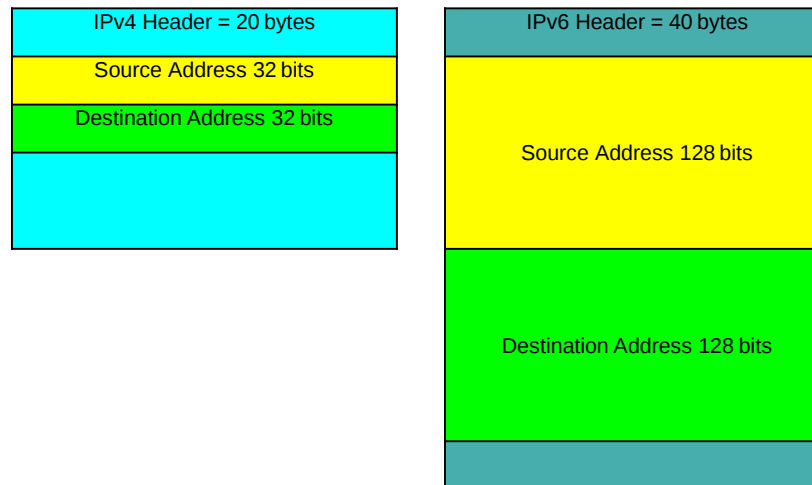
- Addressing
- Header
- Security
- Privacy
- Autoconfiguration
- Routing
- Quality of Service

# Features of IPv6

- Expanded addressing and routing capabilities
  - 128-bit addresses
  - Multicast routing is now scalable with “scope” field
  - Defined usage of “Anycast” addressing

# Features of IPv6

- Simplified header format
  - Some IPv4 fields dropped or made optional
  - IPv6 packet header only twice the size of v4 header, even though address is four times a v4 address



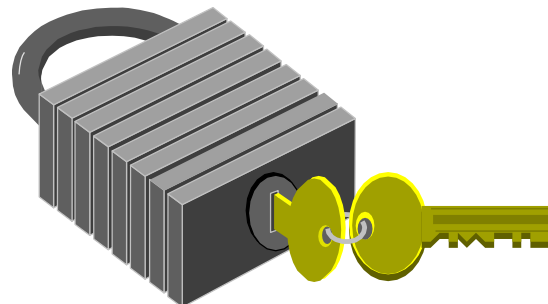
# Features of IPv6

- Extension headers
  - “Options” are now placed in separate headers
  - “Options” are now any length
  - Router doesn’t have to look at most “Options”\*

\* except the hop-by-hop options

# Features of IPv6

- Authentication and Privacy
  - Basic **required** support for authentication and data integrity
  - Basic support for Payload encryption
  - Support for Header and payload encryption

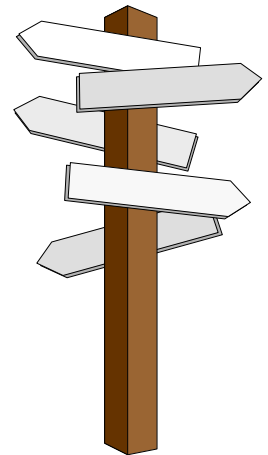


# Features of IPv6

- Auto-configuration
  - Self-configuring nodes for local links
  - Auto-configuration for site links
  - Stateless uses “Router Advertisement”
    - Cost savings
    - Home market potential
  - Stateful uses “DHCPv6”
    - Centralized management
    - New “v6” features

# Features of IPv6

- Source Routing
  - Support for Source Demand Routing Protocol (SDRP)
  - Sender can specify packet route
  - Destination can return packet via same route



# Features of IPv6

- Route Aggregation
  - Address Hierarchy
  - Deployed in “Groups”
  - Defined in RFC 2374

# Features of IPv6

- Quality of Service/Class of Service
  - (flow labels)
  - New 20 bit field for labeling traffic “flows”
  - Continued support for current IPv4 standards

# Features of IPv6

- Transition technologies
  - Required by RFC 1726
  - Various methodologies exist today
    - IPv6 Relays and Gateways (6to4) RFC 3056
    - Tunnels (automatic and manual)
      - – Broker RFC3053
    - NAT-PT RFC 2766
    - and many others...

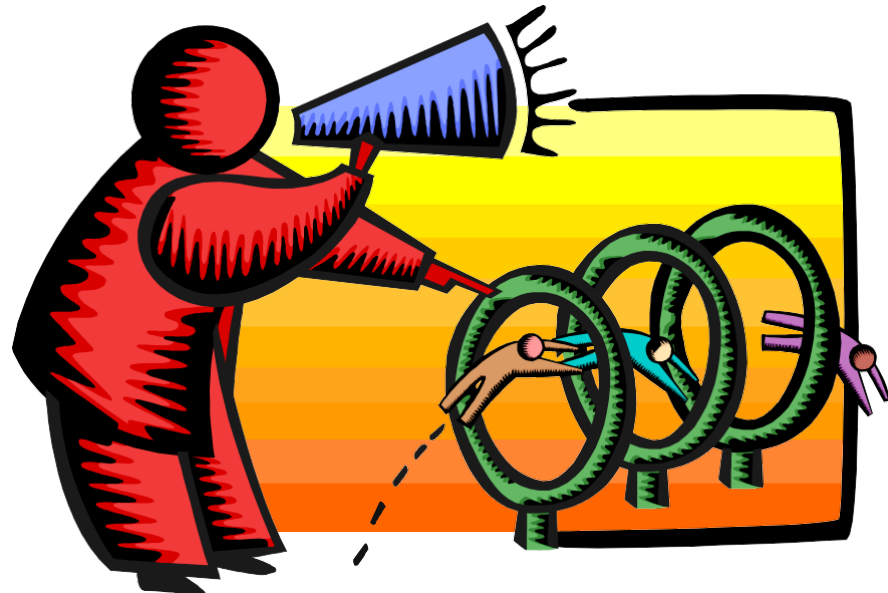
# IPv6 versus IPv4

- IPv4 addresses  $2^{32} = 4,294,967,296$
- IPv6 addresses  $2^{128} =$ 
  - 340,282,366,920,938,463,463,374,607,431,768,211,456
  - 340 undecillion –US, 340 sextillion-UK
  - 79,228,162,514,264,337,593,543,950,336
    - times more v6 addresses than v4
- If IP addresses weighed one gram each
  - IPv4 = 1/7th of the Empire State Building
  - IPv6 = 56 billion(US) earths



# IPv6 versus IPv4

- No Broadcast in IPv6
  - Replaced by Multicast
  - Multicast scope provides flexibility



# IPv6 versus IPv4

- Uncontrolled fragmentation in IPv4
  - Only a source node can fragment IPv6
  - Limits packet size to minimum MTU in path (Maximum Transmission Unit)

# IPv6 versus IPv4

- Extension Headers
  - Supports multiple headers including upper layer headers. Provides for future enhancements.

# IPv6 versus IPv4

- Other features inherent to IPv6
  - Security encryption
  - Header encryption
  - Sender authentication
  - Privacy

# IPv6 Specifications

# IPv6 specifications

- IPv6 - RFC 2460
- IPv6 Neighbor Discovery – RFC 2461
- IPv6 Auto Configuration – RFC 2462

# RFC 2460

- IPv6 Terminology
- IPv6 Packet Header
- IPv6 Addressing
- IPv6 Header Extensions

# IPv6 Terminology

- Node – A device that implements IPv6.
  - A node can be a host or a router
  - A node can be an entertainment system
  - A node can be a PDA or cell phone



# IPv6 Terminology

- Router – A node that forwards IPv6 packets not explicitly addressed to itself.
  - Routers operate at the Network Layer
  - Routers use metrics to determine optimal paths which network traffic should be forwarded



# IPv6 Terminology

- Host – Any node that is not a router
  - A host can be your computer at home
  - A host can be your cell phone or PDA

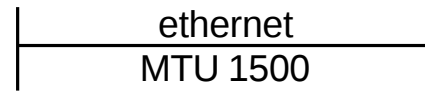


# IPv6 Terminology

- Upper Layer – A protocol layer immediately above IPv6.
  - TCP/UDP
  - ICMP
  - OSPF
  - And many more...

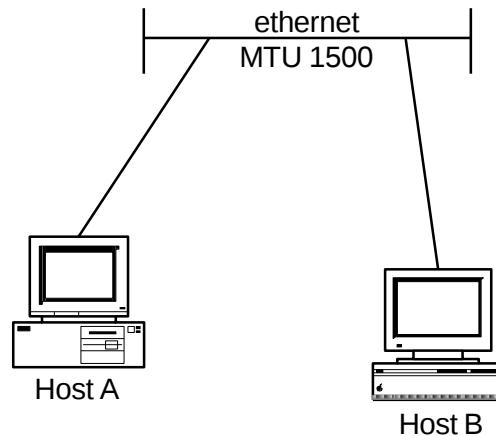
# IPv6 Terminology

- Link – A communication facility or medium over which nodes can communicate at the Data Link Layer.
  - Ethernet
  - Token ring
  - Frame relay
  - A “tunnel”
  - And many more...



# IPv6 Terminology

- Neighbors – Nodes attached to the same link.
  - 2 PC's on a hub can be neighbors
  - Hosts and routers can be neighbors



# IPv6 Terminology

- Interface – A node's attachment to a link.
  - A Network Interface Card (NIC) is an interface
  - A node can have more than one interface
  - Routers often have multiple interfaces
  - A PC's dial-up adapter is an interface



# IPv6 Terminology

- Address – An IPv6-layer identifier for an interface or a set of interfaces.
  - 128-bit address
  - Prefix represented by /nn bits
  - New address structure (to be discussed)

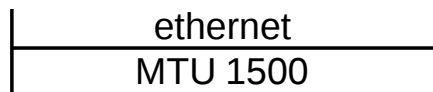
3ffe:80f0:ffff:1::202/64

# IPv6 Terminology

- Packet – An IPv6 header plus payload.
  - Cannot be larger than the “Path MTU”\*
  - Includes header and header extensions
  - Delivered in frames

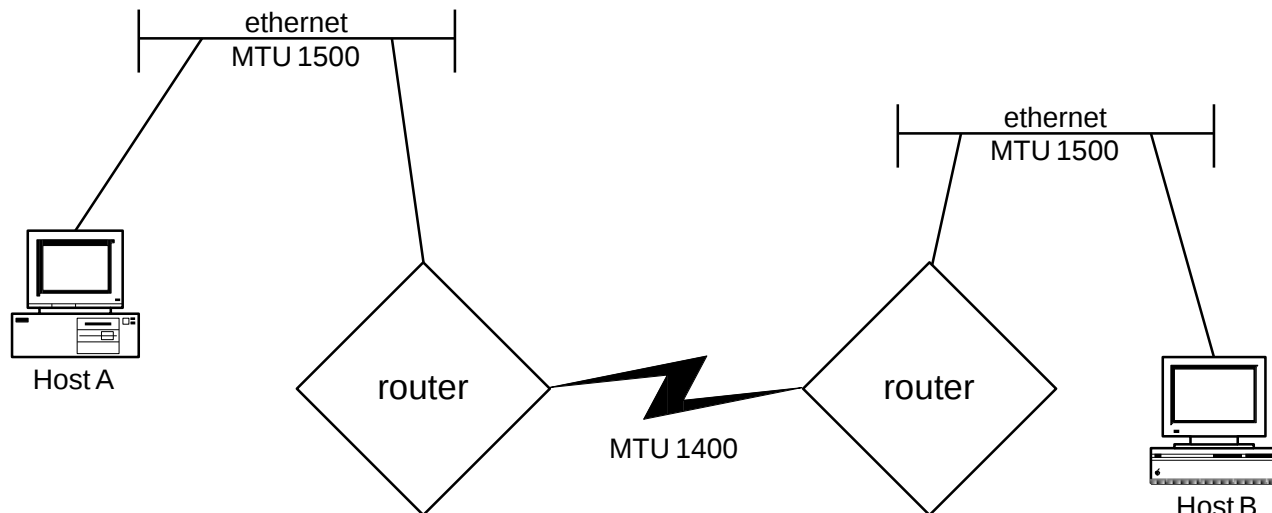
# IPv6 Terminology

- Link MTU – The maximum transmission unit, or packet size, given in octets, that can be conveyed in one piece over a link.
  - MTU – Maximum Transmission Unit
  - Ethernet MTU is 1500 octets
  - 4mb Token Ring MTU is 4464 octets



# IPv6 Terminology

- Path MTU – The minimum link MTU of all the links in a path between a source node and a destination node.
  - The smallest MTU in a network path



# IPv6 Packet Header

# IPv6 Packet

- Similar to IPv4
- No options with padding
- New fields
- Is only 2x length even though address is 4x

# IPv6 Packet Header

- Compare to IPv4 header

|                     |   |   |   |     |   |   |   |                 |   |   |   |   |   |   |   |                 |   |   |   |                 |   |   |   |   |   |   |   |   |   |   |   |
|---------------------|---|---|---|-----|---|---|---|-----------------|---|---|---|---|---|---|---|-----------------|---|---|---|-----------------|---|---|---|---|---|---|---|---|---|---|---|
|                     |   |   |   |     |   |   |   |                 |   | 1 | 1 | 1 | 1 | 1 | 1 | 1               | 1 | 1 | 1 | 2               | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 3 | 3 |
| 0                   | 1 | 2 | 3 | 4   | 5 | 6 | 7 | 8               | 9 | 0 | 1 | 2 | 3 | 4 | 5 | 6               | 7 | 8 | 9 | 0               | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 0 | 1 |
| Version=4           |   |   |   | IHL |   |   |   | Type of Service |   |   |   |   |   |   |   | Total Length    |   |   |   |                 |   |   |   |   |   |   |   |   |   |   |   |
| Identifier          |   |   |   |     |   |   |   |                 |   |   |   |   |   |   |   | Flags           |   |   |   | Fragment Offset |   |   |   |   |   |   |   |   |   |   |   |
| Time to Live        |   |   |   |     |   |   |   | Protocol        |   |   |   |   |   |   |   | Header Checksum |   |   |   |                 |   |   |   |   |   |   |   |   |   |   |   |
| Source Address      |   |   |   |     |   |   |   |                 |   |   |   |   |   |   |   |                 |   |   |   |                 |   |   |   |   |   |   |   |   |   |   |   |
| Destination Address |   |   |   |     |   |   |   |                 |   |   |   |   |   |   |   |                 |   |   |   |                 |   |   |   |   |   |   |   |   |   |   |   |
| Options + Padding   |   |   |   |     |   |   |   |                 |   |   |   |   |   |   |   |                 |   |   |   |                 |   |   |   |   |   |   |   |   |   |   |   |

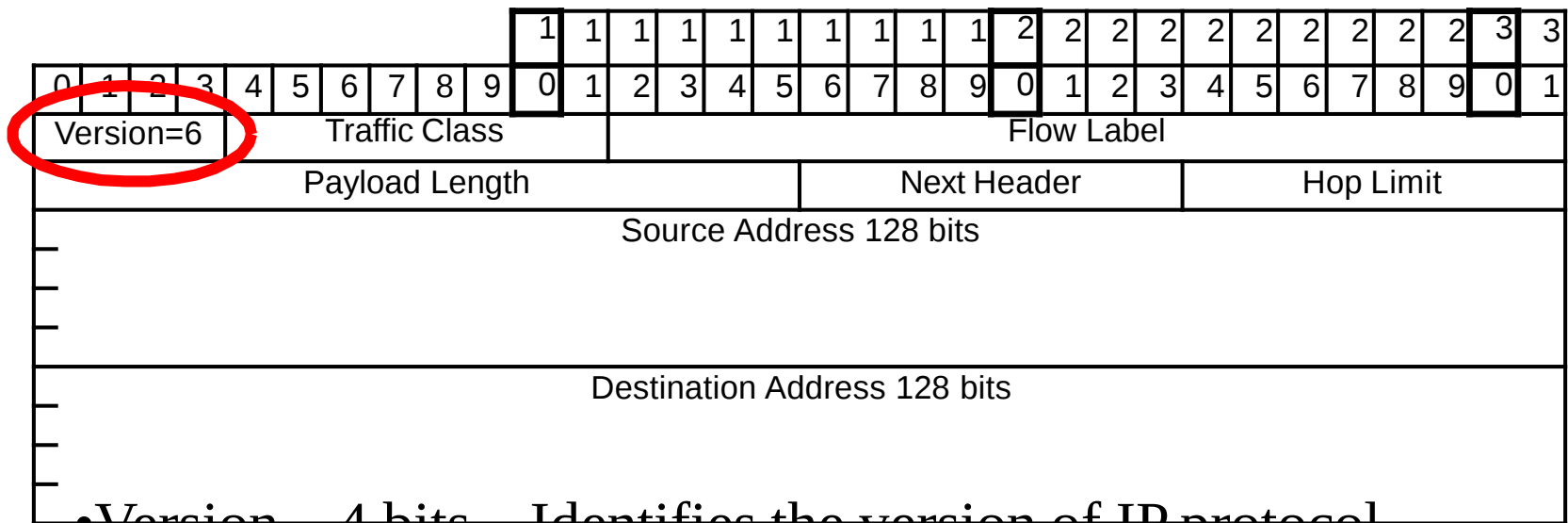


# IPv6 Packet Header

|                              |   |   |   |               |   |   |   |   |   |   |   |            |   |   |   |             |   |   |   |   |   |   |   |           |   |   |   |   |   |   |   |   |   |   |
|------------------------------|---|---|---|---------------|---|---|---|---|---|---|---|------------|---|---|---|-------------|---|---|---|---|---|---|---|-----------|---|---|---|---|---|---|---|---|---|---|
|                              |   |   |   |               |   |   |   |   |   |   |   | 1          | 1 | 1 | 1 | 1           | 1 | 1 | 1 | 1 | 1 | 2 | 2 | 2         | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 2 | 3 | 3 |
| 0                            | 1 | 2 | 3 | 4             | 5 | 6 | 7 | 8 | 9 | 0 | 1 | 2          | 3 | 4 | 5 | 6           | 7 | 8 | 9 | 0 | 1 | 2 | 3 | 4         | 5 | 6 | 7 | 8 | 9 | 0 | 1 |   |   |   |
| Version=6                    |   |   |   | Traffic Class |   |   |   |   |   |   |   | Flow Label |   |   |   |             |   |   |   |   |   |   |   |           |   |   |   |   |   |   |   |   |   |   |
| Payload Length               |   |   |   |               |   |   |   |   |   |   |   |            |   |   |   | Next Header |   |   |   |   |   |   |   | Hop Limit |   |   |   |   |   |   |   |   |   |   |
| Source Address 128 bits      |   |   |   |               |   |   |   |   |   |   |   |            |   |   |   |             |   |   |   |   |   |   |   |           |   |   |   |   |   |   |   |   |   |   |
| Destination Address 128 bits |   |   |   |               |   |   |   |   |   |   |   |            |   |   |   |             |   |   |   |   |   |   |   |           |   |   |   |   |   |   |   |   |   |   |

- IPv6 Header with no header extensions
- Carried in frames
- 8 fields

# IPv6 Packet Header

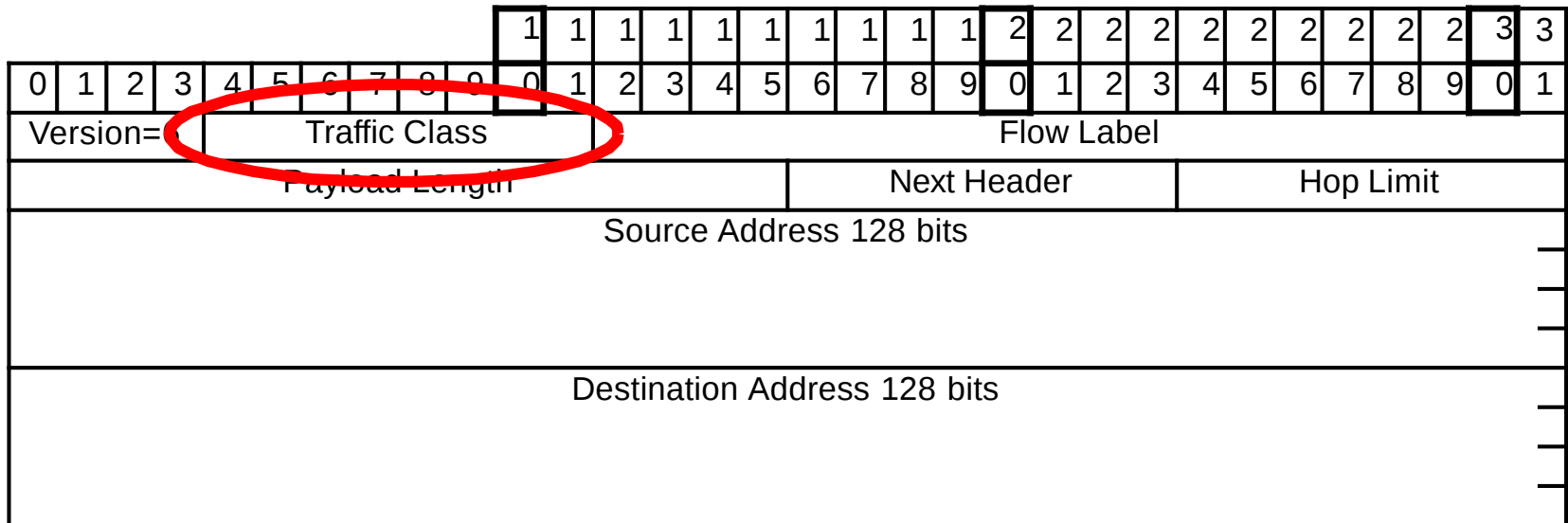


• Version = 4 bits = Identifies the version of IP protocol

• 0100 (4) for IPv4

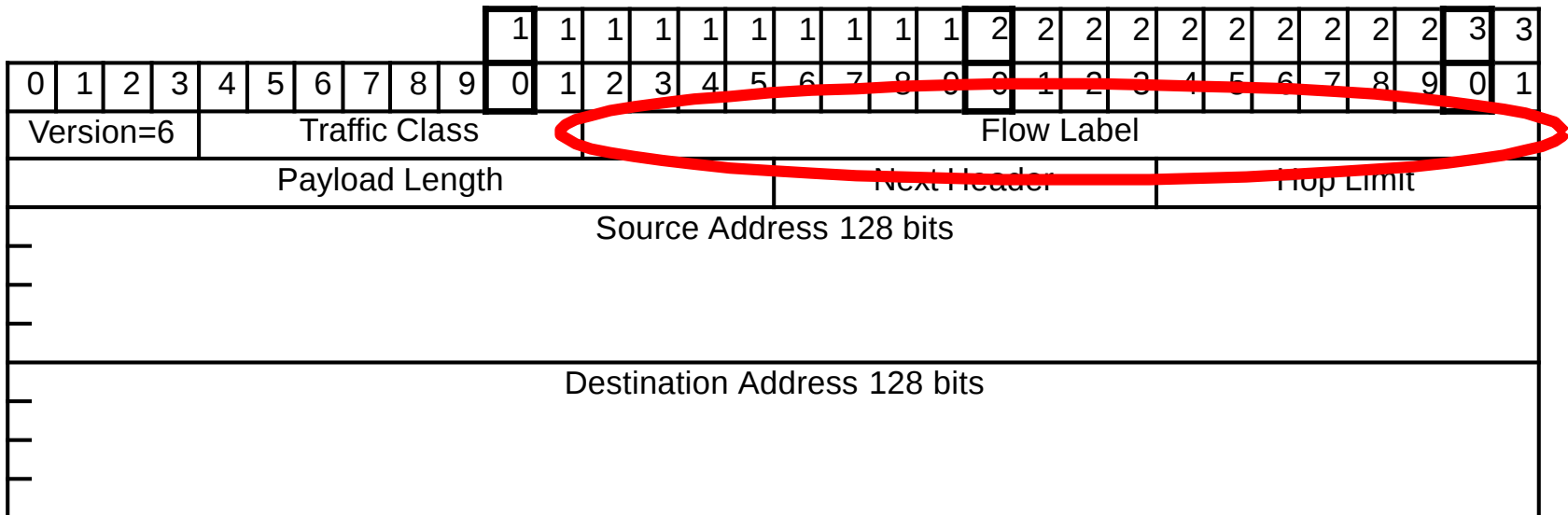
• 0110 (6) for IPv6

# IPv6 Packet Header



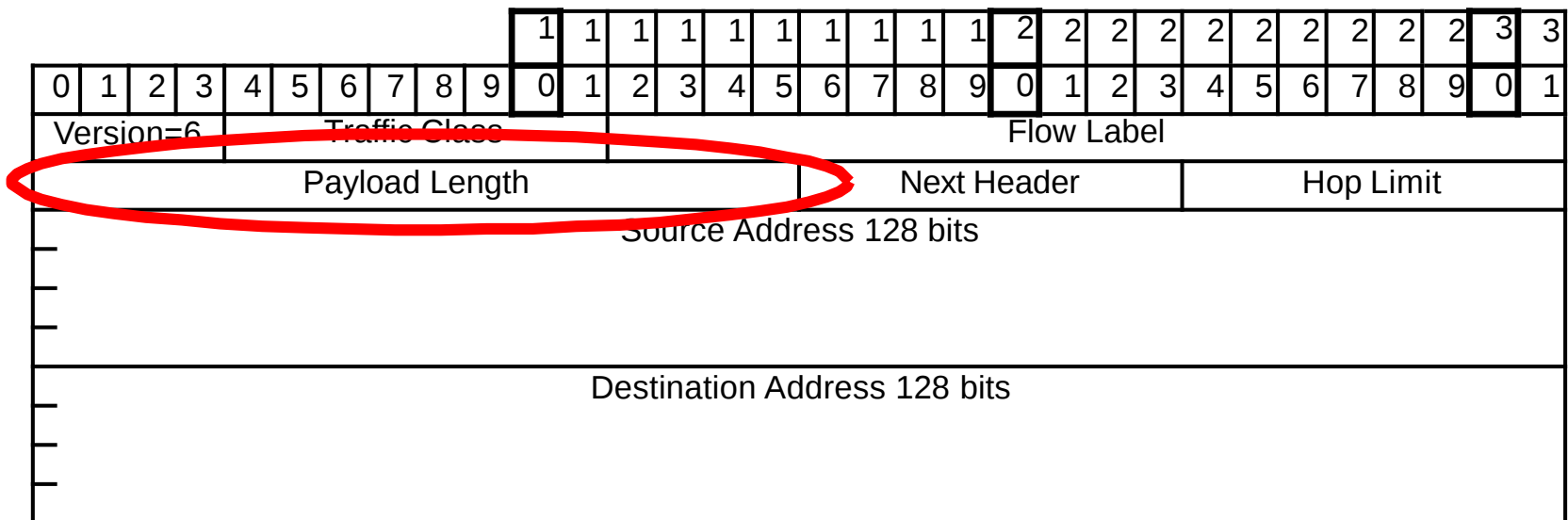
- Traffic Class – 8 bits – Allows originating nodes and/or routers to distinguish between different classes or priorities of IPv6 packets
  - QoS is an example implementation
  - RSVP uses Traffic Class

# IPv6 Packet Header



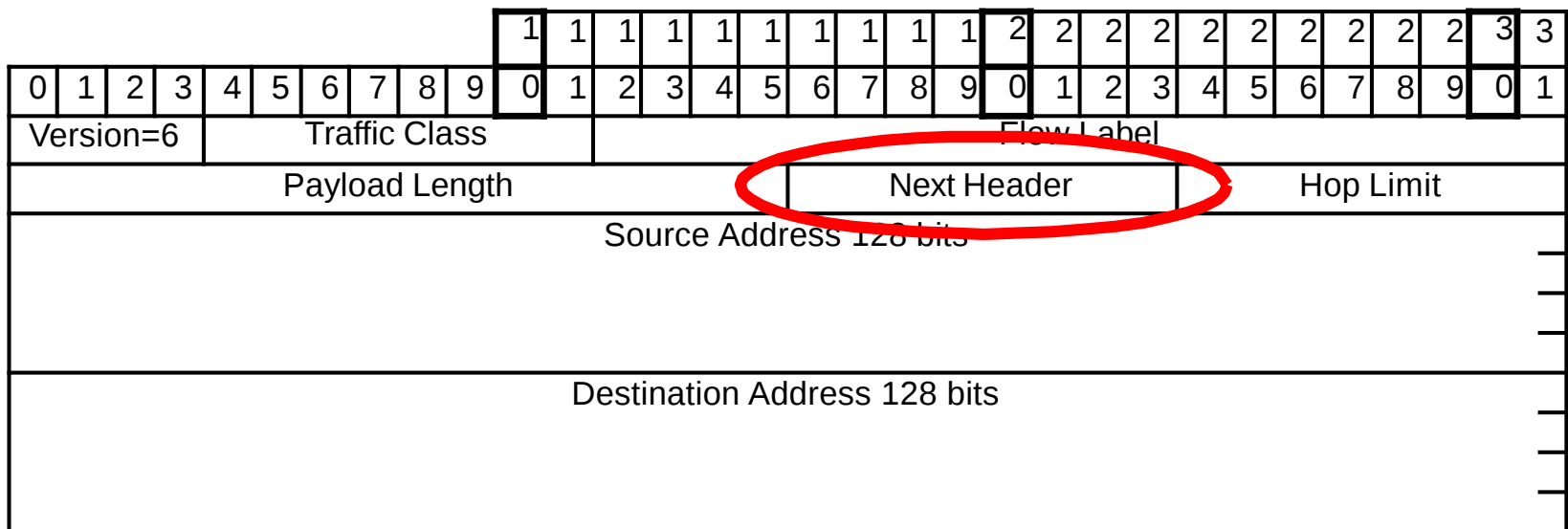
- Flow Label – 20 bits – Used to “label” a flow of traffic.
  - May be used to request special handling
  - RFC 1809 “Using the Flow Label Field in IPv6”

# IPv6 Packet Header



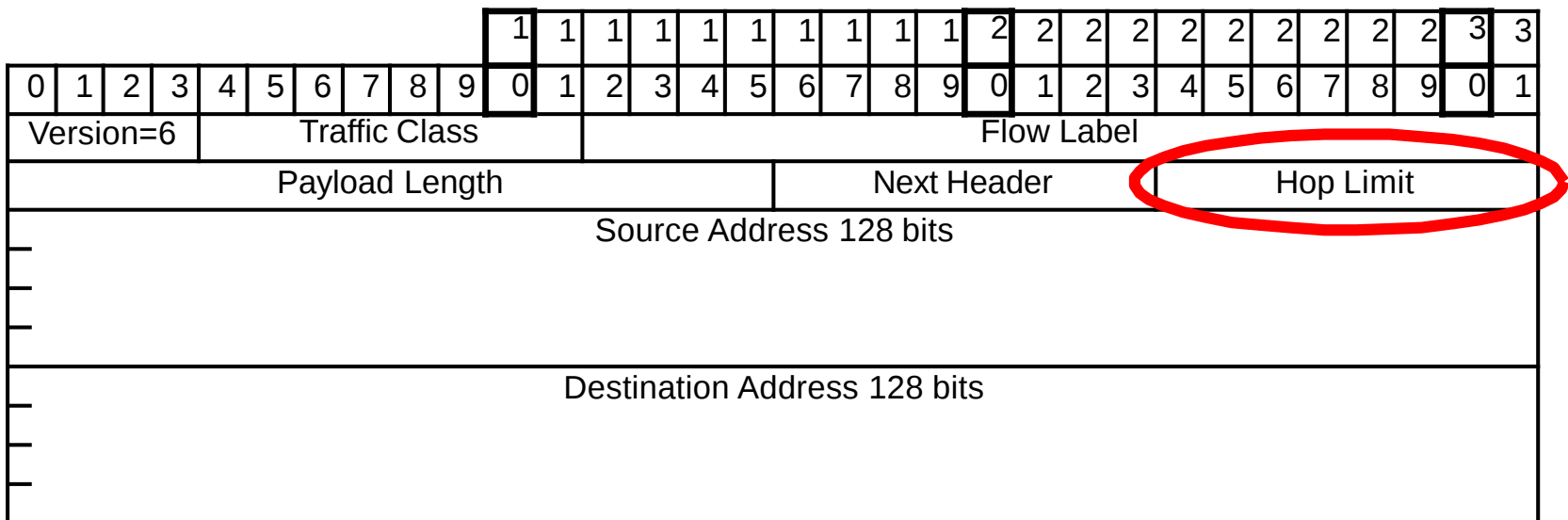
- Payload Length – 16 bits – Length, in octets, of the payload
- Payload is balance of IPv6 packet following header
- Extension headers are part of payload
- Jumbo Payloads\*

# IPv6 Packet Header



- Next Header – 8 bits – Identifies the “extension” header immediately following
- Packet may have zero, one, or more extension headers
- Extension header order is important\*

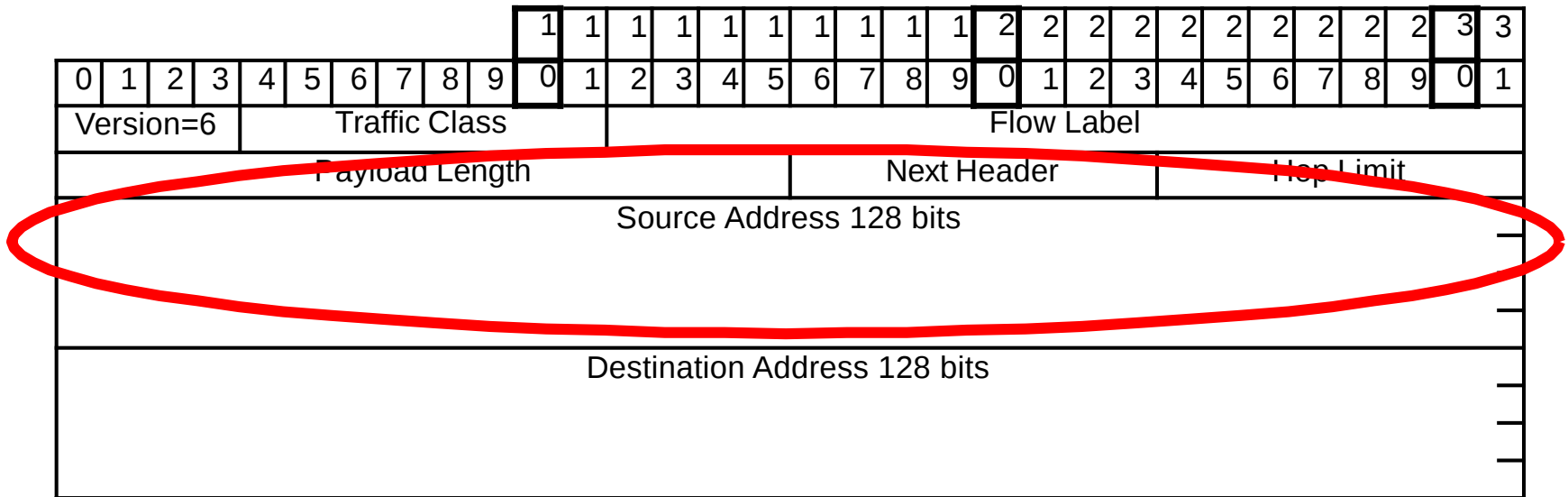
# IPv6 Packet Header



- Hop Limit – 8 bits – Maximum number of **hops** an IPv6 packet can be forwarded.

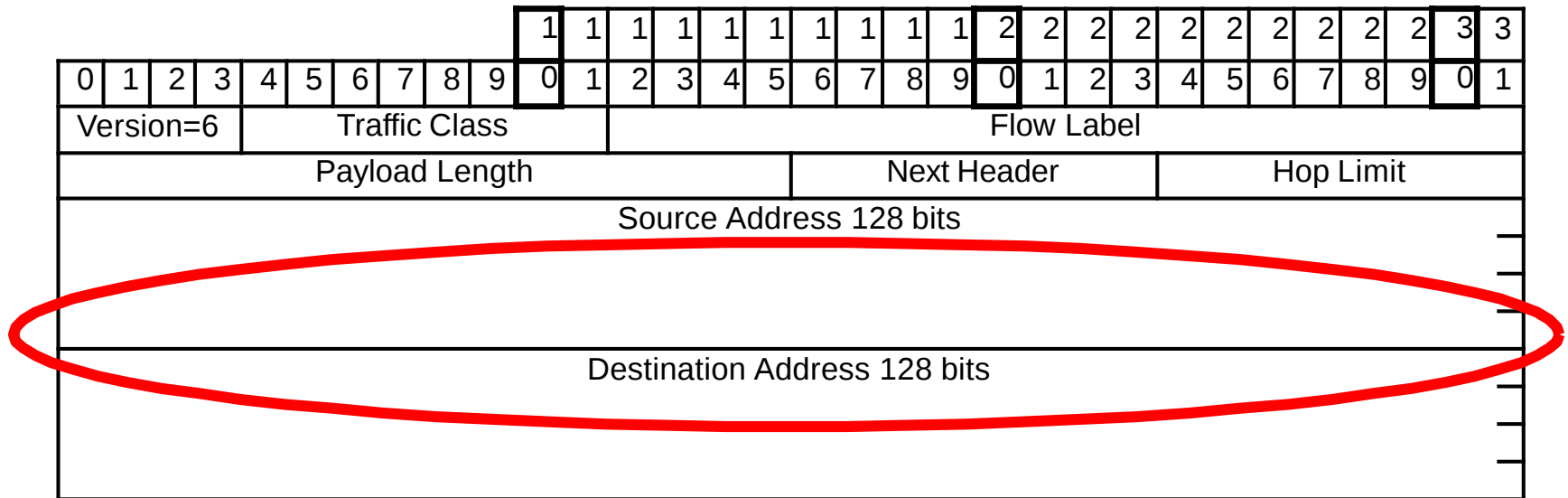
- Similar to IPv4 TTL, but NOT time
- Decrement by each node on path

# IPv6 Packet Header



- Source Address – 128-bits
  - versus IPv4 32-bit

# IPv6 Packet Header



- Destination Address – 128-bits
  - versus IPv4 32-bit

# IPv6 Extension Headers

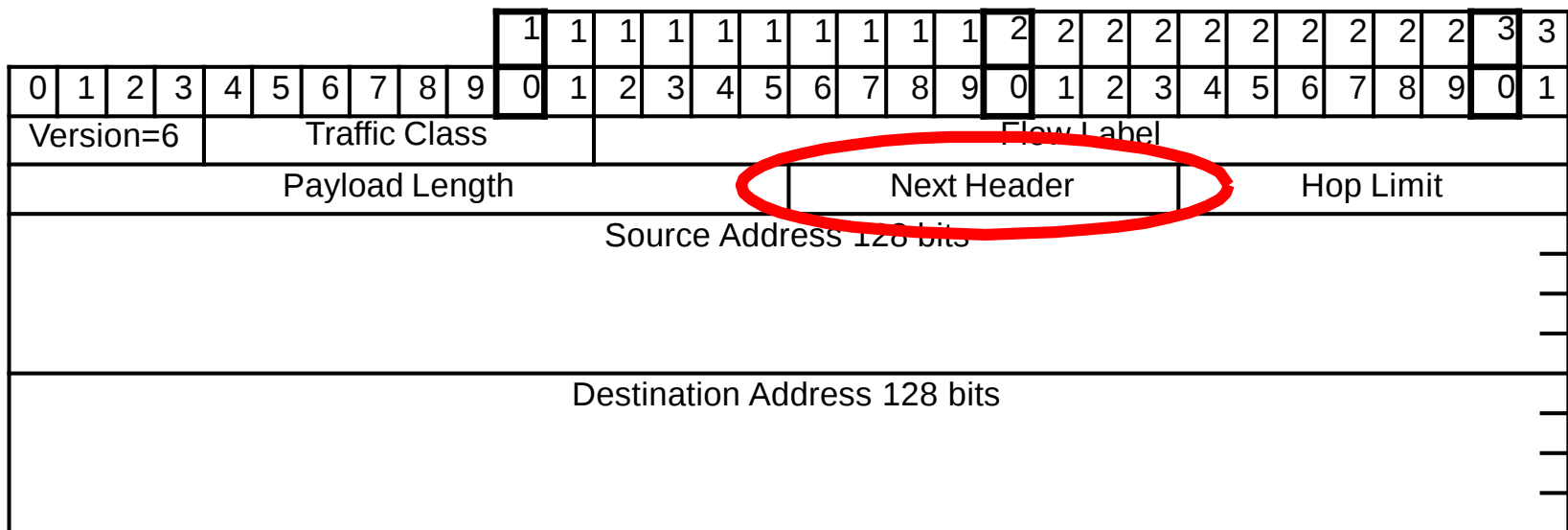
# IPv6 Extension Headers

- What are Extension Headers?
- How are they identified in the IPv6 Header?
- Extension Header Order
- Extension Header Specifics

# IPv6 Extension Headers

- Upper layer headers and options
  - Not examined by nodes on path
  - Allows flexibility

# IPv6 Extension Headers



- Next Header – 8 bits – Identifies the “extension” header immediately following
  - Packet may have zero, one, or more extension headers
  - Extension header order is important\*

# IPv6 Extension Headers

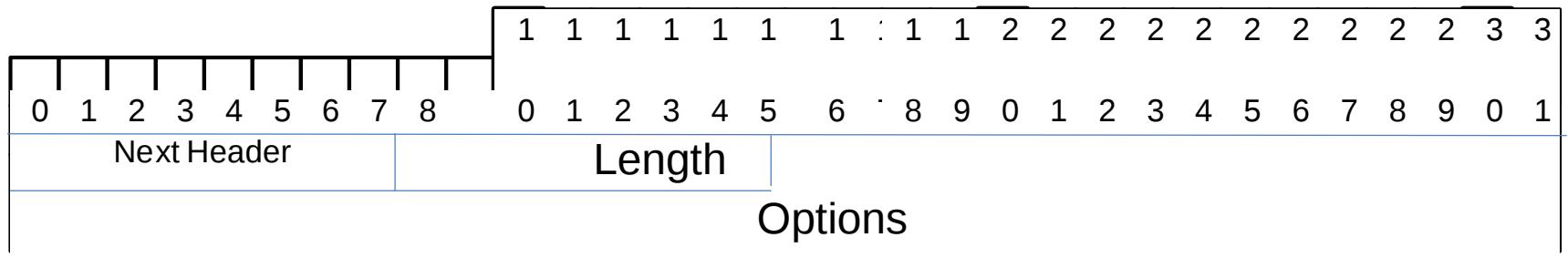
- IPv6 Header
- Hop-by-Hop
- Routing Header
- Fragment Header
- Authentication Header
- Encapsulating Security Payload
- Upper Layer protocol

# IPv6 Extension Headers

- 0 – Hop-by-Hop Options – RFC 2460
  - Must be first header extension
  - Examined by every node on a delivery path
  - Supports Jumbo payload  $>65535$   $<4$  billion
  - Cannot use Jumbo with Fragment
  - Only one allowed per packet

# IPv6 Extension Headers

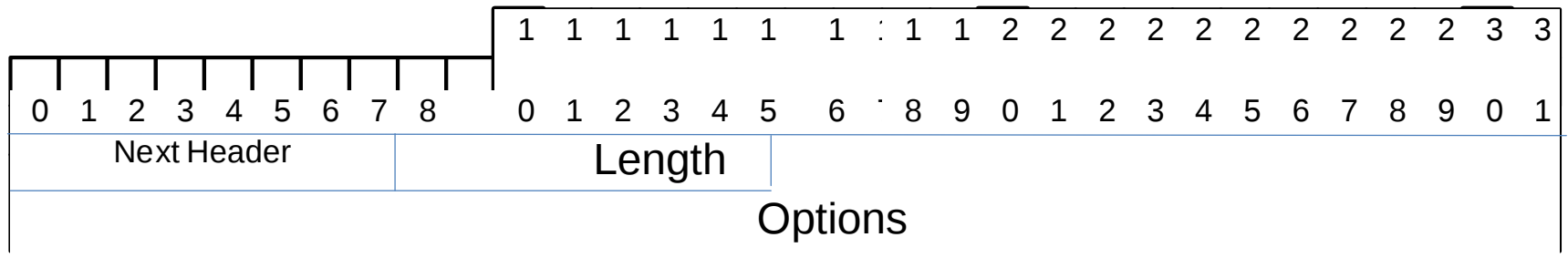
## Example Hop-by-Hop Extension Header



- Next Header field identifies the header immediately following.
- Header Extension Length identifies the length of the hop-by-hop extension header, in octets, not including the first 8 octets.

# IPv6 Extension Headers

## Example Hop-by-Hop Extension Header



- Options - Variable-length field, of length such that the complete Hop-by-Hop Options header is an integer multiple of 8 octets long. Contains one or more TLV-encoded options. (type-length-value)

# IPv6 Extension Headers

## Example Hop-by-Hop Extension Header -continued

|             |   |   |   |   |   |   |   |   |               |   |   |   |   |   |             |
|-------------|---|---|---|---|---|---|---|---|---------------|---|---|---|---|---|-------------|
|             |   |   |   |   |   |   |   |   |               | 1 | 1 | 1 | 1 | 1 | 1           |
| 0           | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9             | 0 | 1 | 2 | 3 | 4 | 5           |
| Option Type |   |   |   |   |   |   |   |   | Option Length |   |   |   |   |   | Option Data |

- TLV field is used in Hop-by-Hop header and Destination header.

# IPv6 Extension Headers

- 1 – ICMPv4
  - Continued support for ICMPv4
  - New header for ICMPv6
- Internet Control Message Protocol

# IPv6 Extension Headers

- 6 – TCP – Transmission Control Protocol
  - Provides reliable delivery
  - Upper Layer header

# IPv6 Extension Headers

- 17 – UDP – User Datagram Protocol
  - Unreliable delivery
  - Upper Layer header

# IPv6 Extension Headers

- 43 – Routing Options – RFC 2460
  - Lists one or more IPv6 nodes to be “visited” on the way to a packets destination
  - Not looked at by each node on path

# IPv6 Extension Headers

- 44 – Fragmentation Options – RFC 2460
  - Only the source node can fragment a packet in IPv6
  - Expected that packets sent are no larger than path-MTU

# IPv6 Extension Headers

- 50 – Encapsulating Security Payload – RFC 2406
  - Provides Encryption security
  - Confidentiality
  - Data origin Authentication
  - Connectionless integrity

# IPv6 Extension Headers

- 51 – Authentication Options – RFC 2402
  - Provides connectionless integrity
  - Data origin Authentication

# IPv6 Extension Headers

- 58 – ICMPv6 – RFC 2463
  - ICMPv6 is used by IPv6 nodes to report errors encountered in processing packets
  - Is an integral part of IPv6 and MUST be fully implemented by every IPv6 node
  - Used for Neighbor Discovery

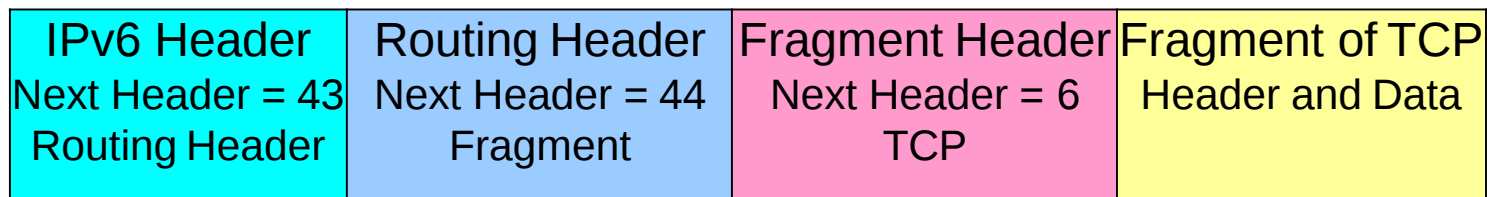
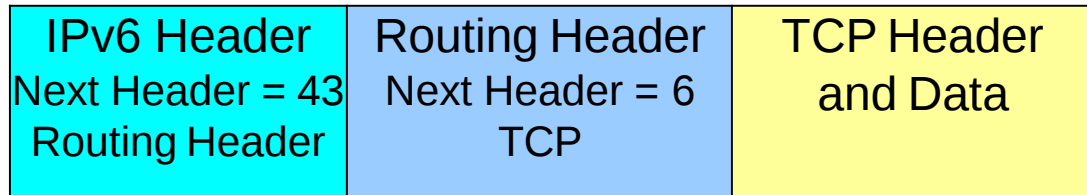
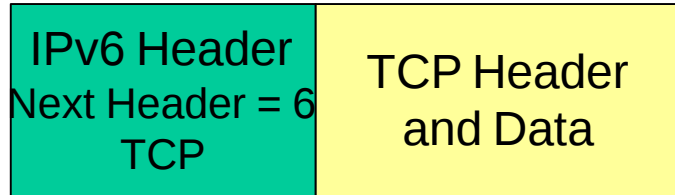
# IPv6 Extension Headers

- 59 – No next header – RFC 2460
  - There will be no Extension Header following this header

# IPv6 Extension Headers

- 60 - Destination Options – RFC 2460
  - – Used to carry optional information for the Destination

# IPv6 Extension Headers



# Internet Control Protocols (1)

IP works with the help of several control protocols:

- ICMP is a companion to IP that returns error info
  - Required, and used in many ways, e.g., for traceroute
- ARP finds Ethernet address of a local IP address
  - Glue that is needed to send any IP packets
  - Host queries an address and the owner replies
- DHCP assigns a local IP address to a host
  - Gets host started by automatically configuring it
  - Host sends request to server, which grants a lease

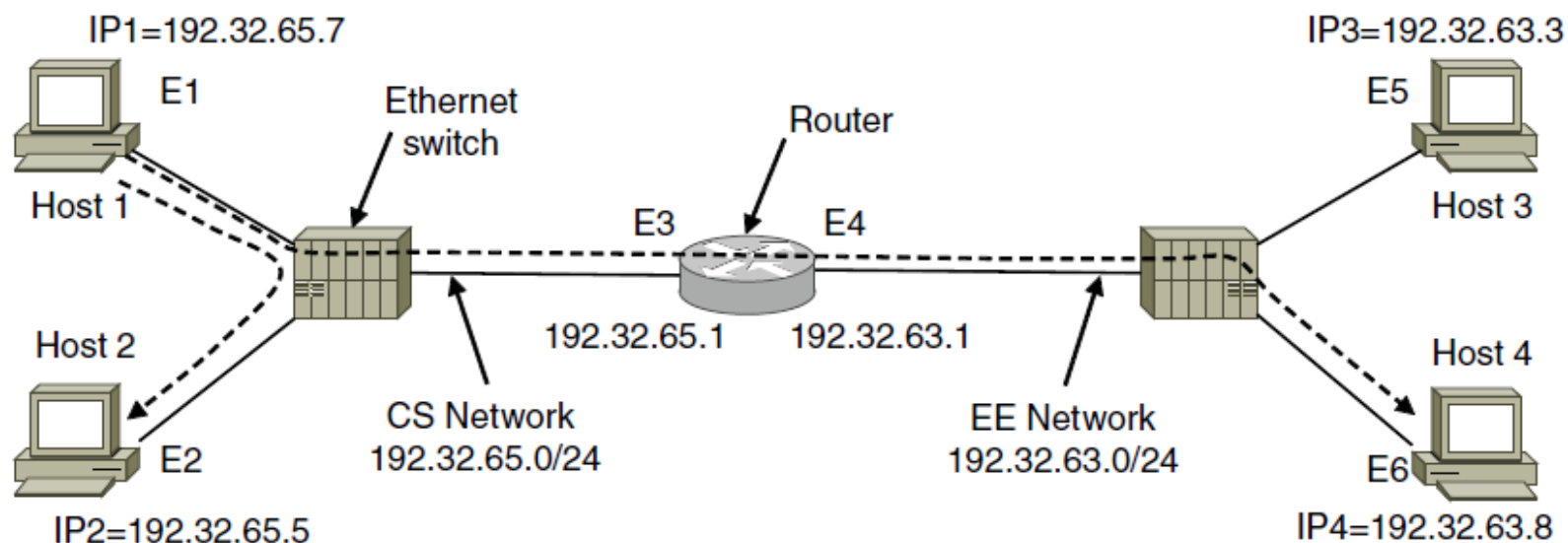
# Internet Control Protocols (2)

Main ICMP (Internet Control Message Protocol) types:

| Message type                      | Description                      |
|-----------------------------------|----------------------------------|
| Destination unreachable           | Packet could not be delivered    |
| Time exceeded                     | Time to live field hit 0         |
| Parameter problem                 | Invalid header field             |
| Source quench                     | Choke packet                     |
| Redirect                          | Teach a router about geography   |
| Echo and Echo reply               | Check if a machine is alive      |
| Timestamp request/reply           | Same as Echo, but with timestamp |
| Router advertisement/solicitation | Find a nearby router             |

# Internet Control Protocols (3)

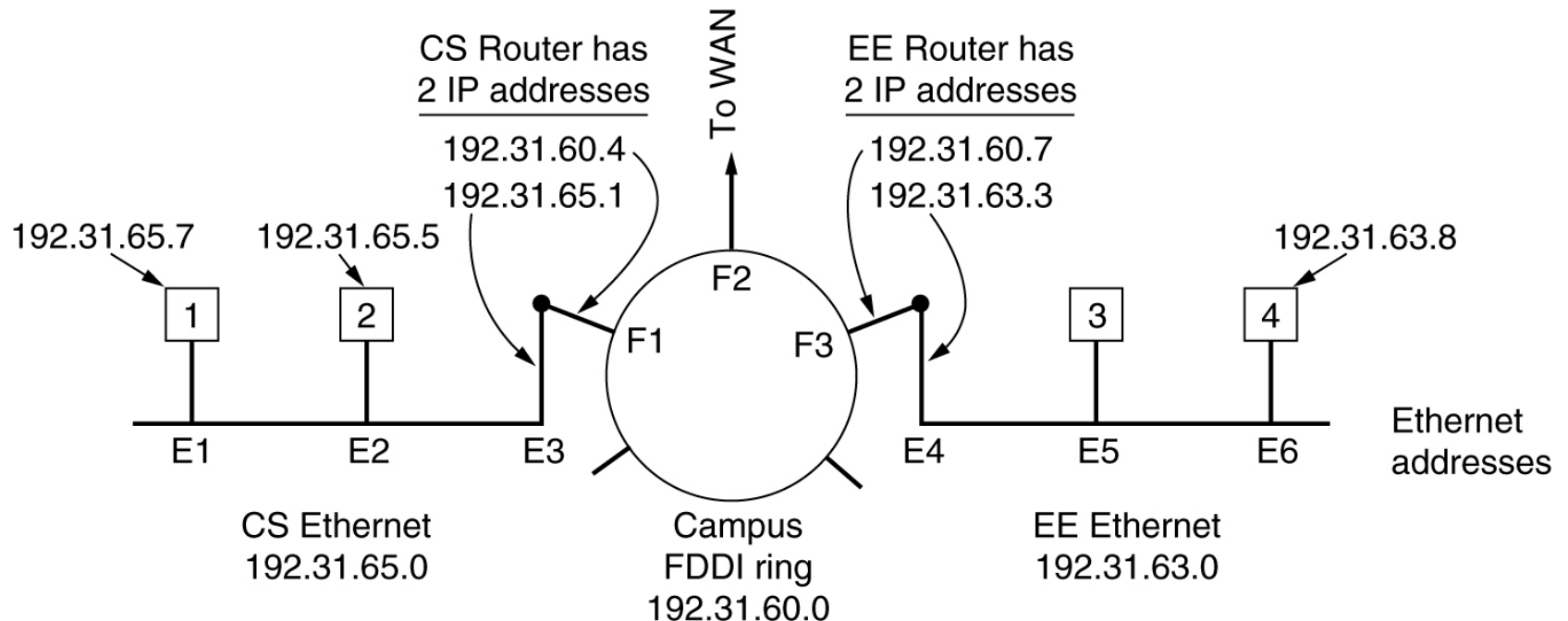
- ARP (Address Resolution Protocol) lets nodes find target Ethernet addresses [pink] from their IP addresses



| Frame                  | Source IP | Source Eth. | Destination IP | Destination Eth. |
|------------------------|-----------|-------------|----------------|------------------|
| Host 1 to 2, on CS net | IP1       | E1          | IP2            | E2               |
| Host 1 to 4, on CS net | IP1       | E1          | IP4            | E3               |
| Host 1 to 4, on EE net | IP1       | E4          | IP4            | E6               |

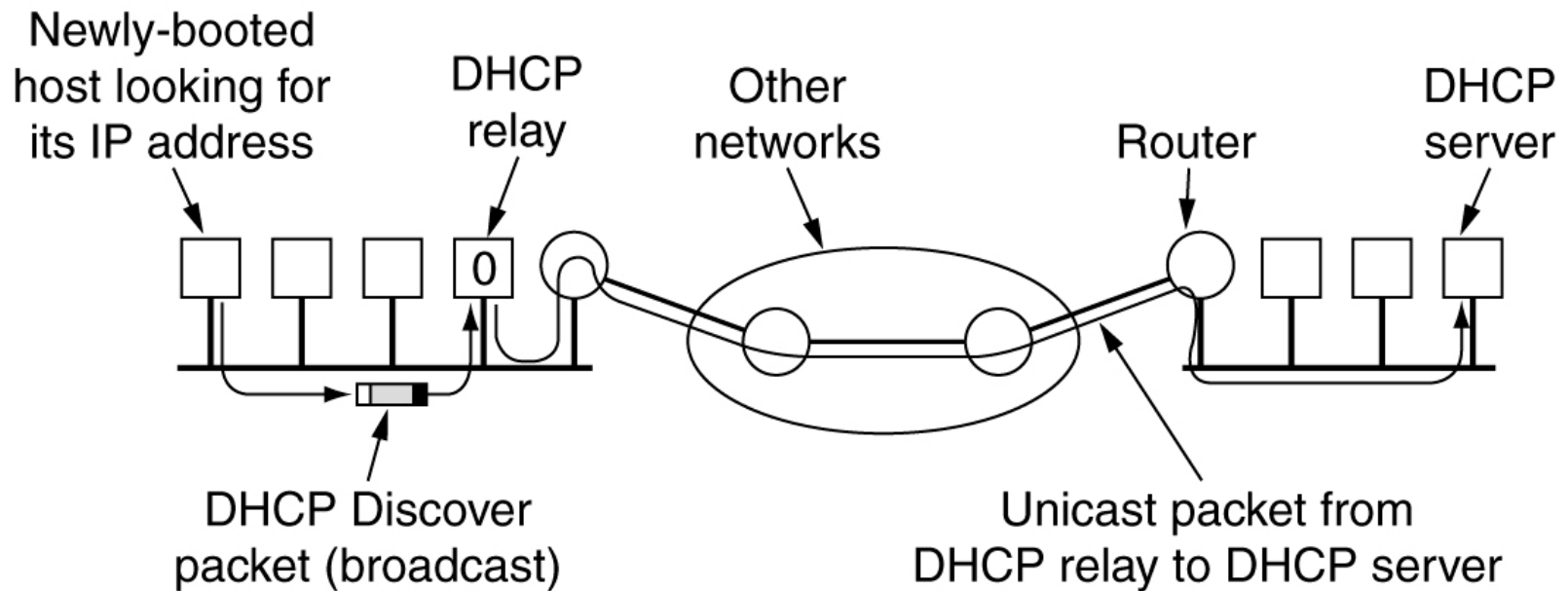
# ARP– The Address Resolution Protocol

Three interconnected /24 networks: two Ethernet and an FDDI ring.

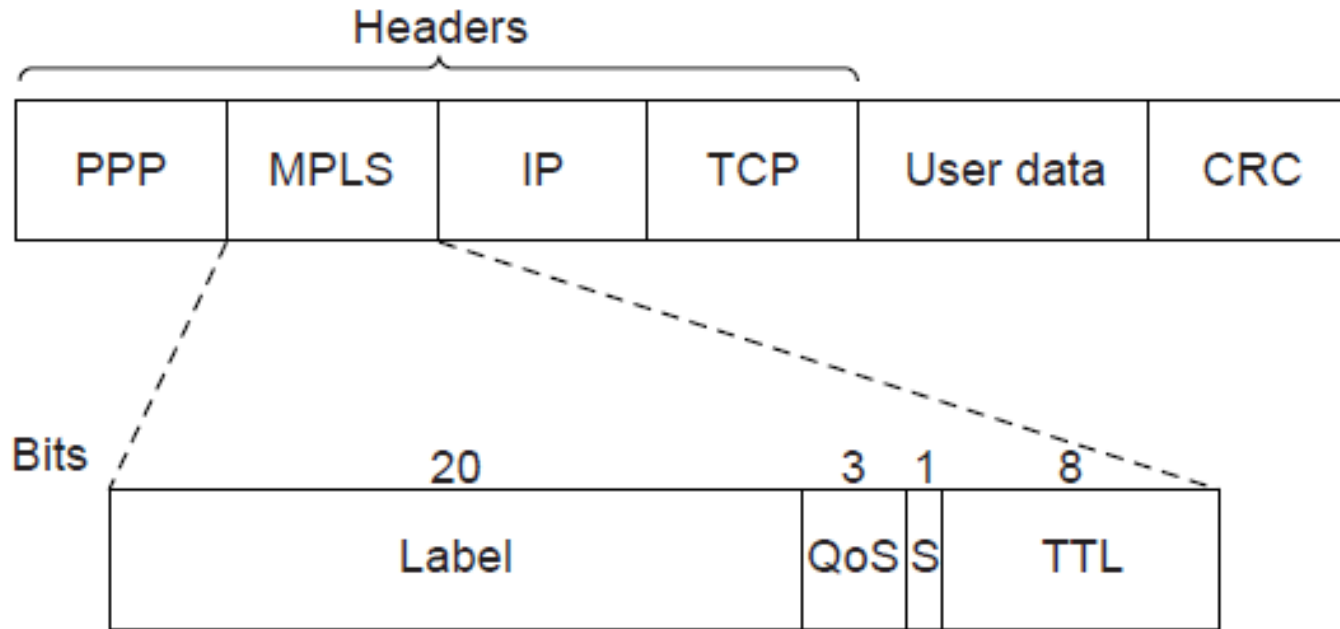


# Dynamic Host Configuration Protocol

## Operation of DHCP.

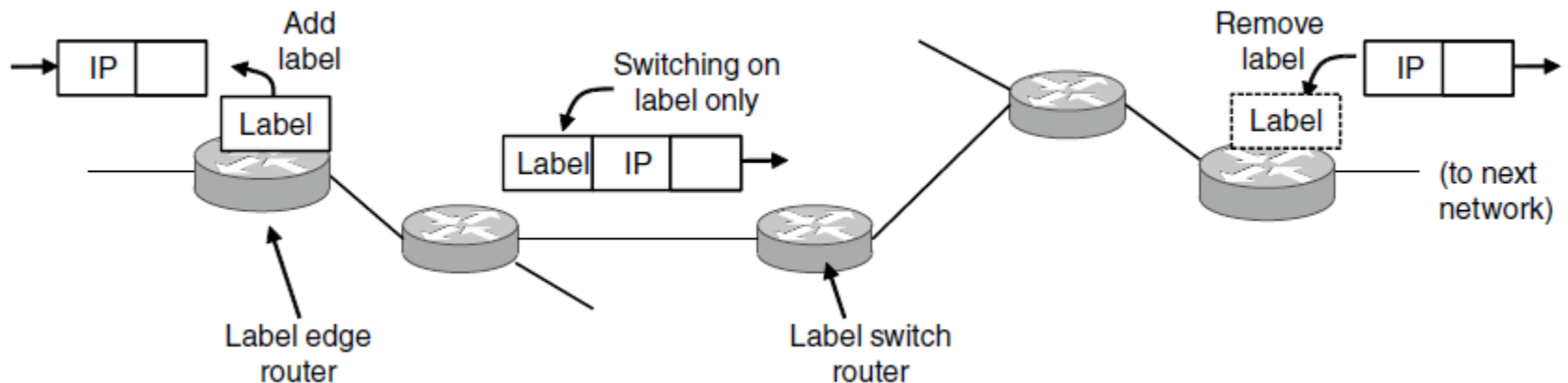


# Label Switching and MPLS (1)



Transmitting a TCP segment using IP, MPLS, and PPP.

# Label Switching and MPLS (2)



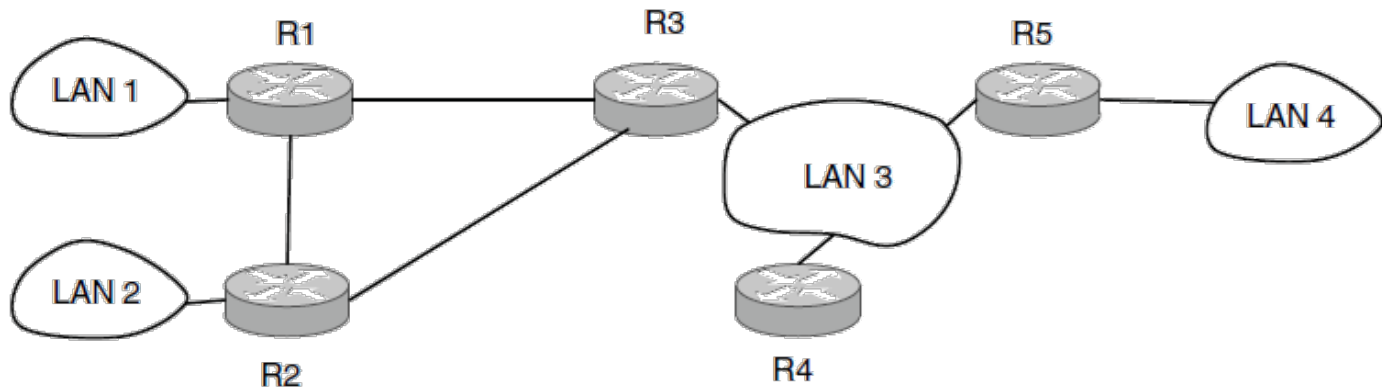
Forwarding an IP packet through an MPLS network

# OSPF— Interior Routing Protocol (1)

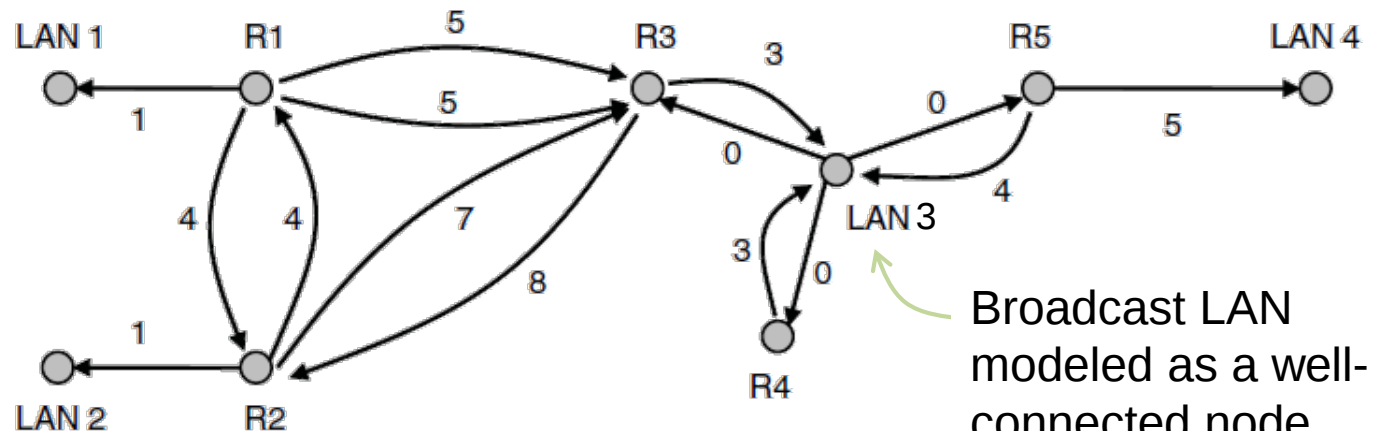
OSPF computes routes for a single network (e.g., ISP)

- Models network as a graph of weighted edges

Network:



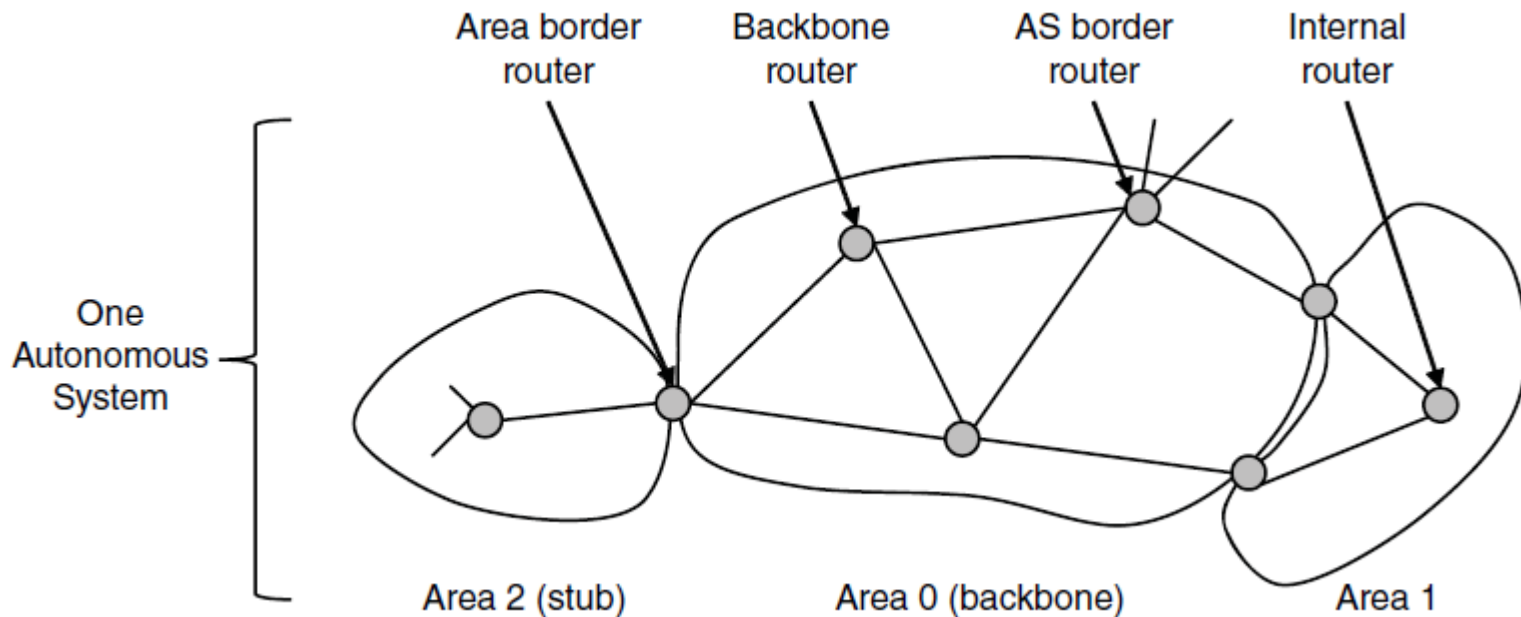
Graph:



# OSPF— Interior Routing Protocol (2)

OSPF divides one large network (Autonomous System) into areas connected to a backbone area

- Helps to scale; summaries go over area borders



# OSPF— Interior Routing Protocol (3)

OSPF (Open Shortest Path First) is link-state routing:

- Uses messages below to reliably flood topology
- Then runs Dijkstra to compute routes

| 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        |



# BGP— Exterior Routing Protocol (1)

BGP (Border Gateway Protocol) computes routes across interconnected, autonomous networks

- Key role is to respect networks' policy constraints

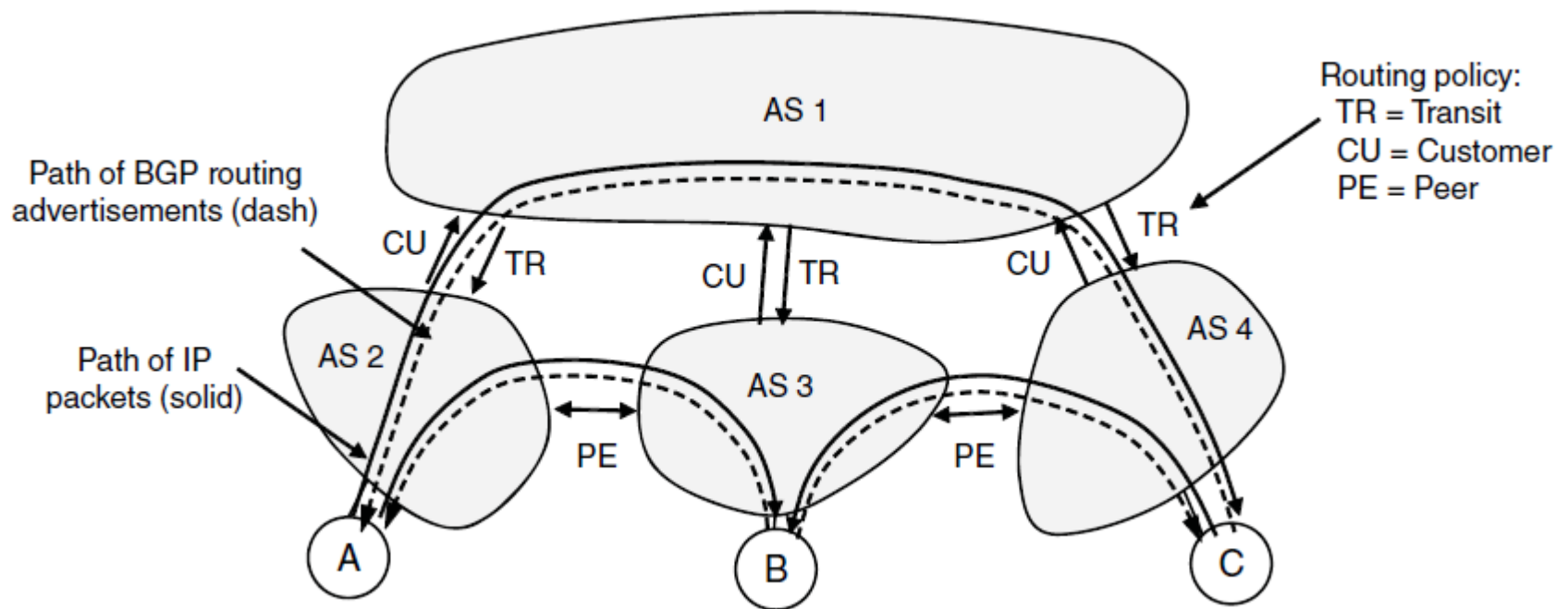
Example policy constraints:

- No commercial traffic for educational network
- Never put Iraq on route starting at Pentagon
- Choose cheaper network
- Choose better performing network
- Don't go from Apple to Google to Apple

# BGP— Exterior Routing Protocol (2)

Common policy distinction is transit vs. peering:

- Transit carries traffic for pay; peers for mutual benefit
- AS1 carries  $AS2 \leftrightarrow AS4$  (Transit) but not AS3 (Peer)



# BGP— Exterior Routing Protocol (3)

- BGP propagates messages along policy-compliant routes
  - Message has prefix, AS path (to detect loops) and next-hop IP (to send over the local network)

