# Computer Networks Basic Protocols

Assoc. Prof. Dr. Berk CANBERK

**25 October 2017**

**-Network Layer-**

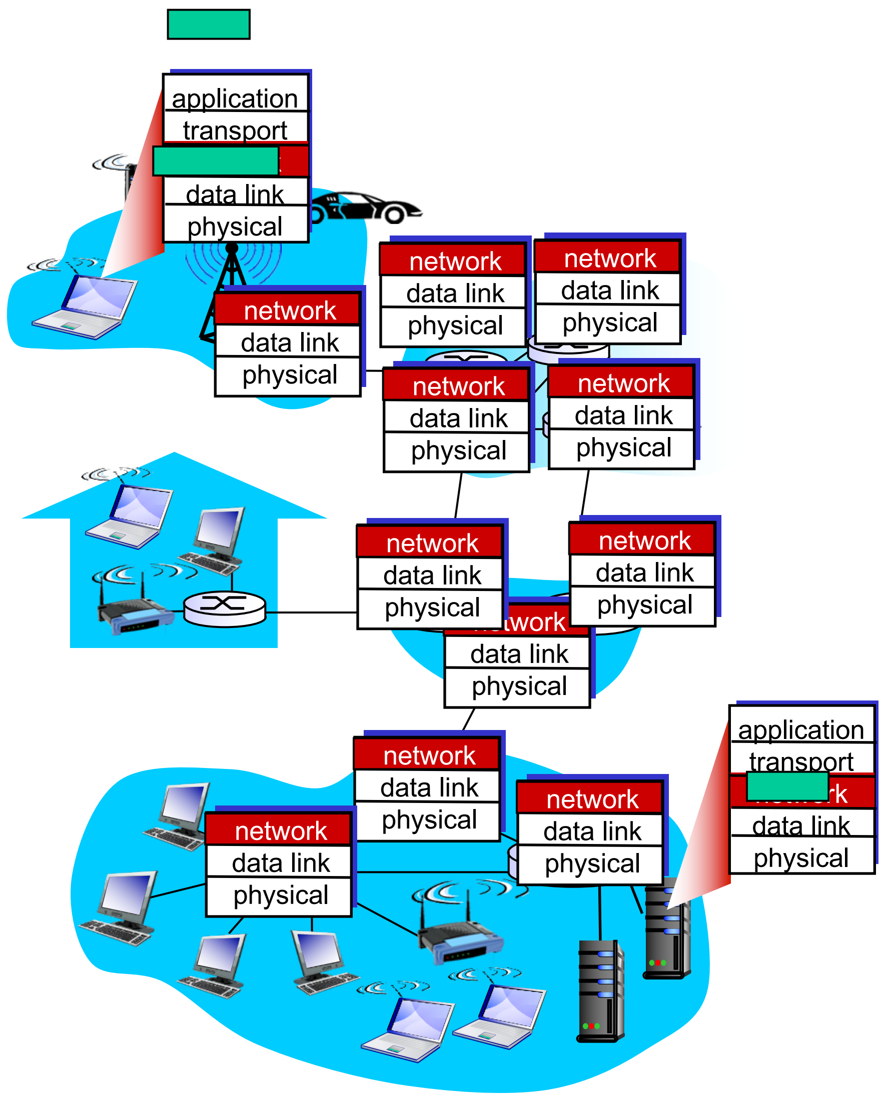
References:

-*Data and Computer Communications*, William Stallings, Pearson-Prentice Hall, 9th Edition, 2010.

-*Computer Networking, A Top-Down Approach Featuring the Internet*, James F.Kurose, Keith W.Ross, Pearson-Addison Wesley, 6th Edition, 2012.

# Network layer

- transport segment from sending to receiving host
- on sending side encapsulates segments into datagrams
- on receiving side, delivers segments to transport layer
- network layer protocols in *every* host, router
- router examines header fields in all IP datagrams passing through it



- Concerned with getting packets from source to destination
- Network layer must
  - know the subnet topology and
  - choose appropriate paths through it
- When source and destination are in different networks, network layer must handle
- Services provided to Transport Layer:
  - Should be independent of the subnet topology
  - Should be independent of the router
  - Transport Layer should be shielded from the number, type and topology of the subnets present
  - The network addresses available to the Transport Layer should use a uniform numbering plan

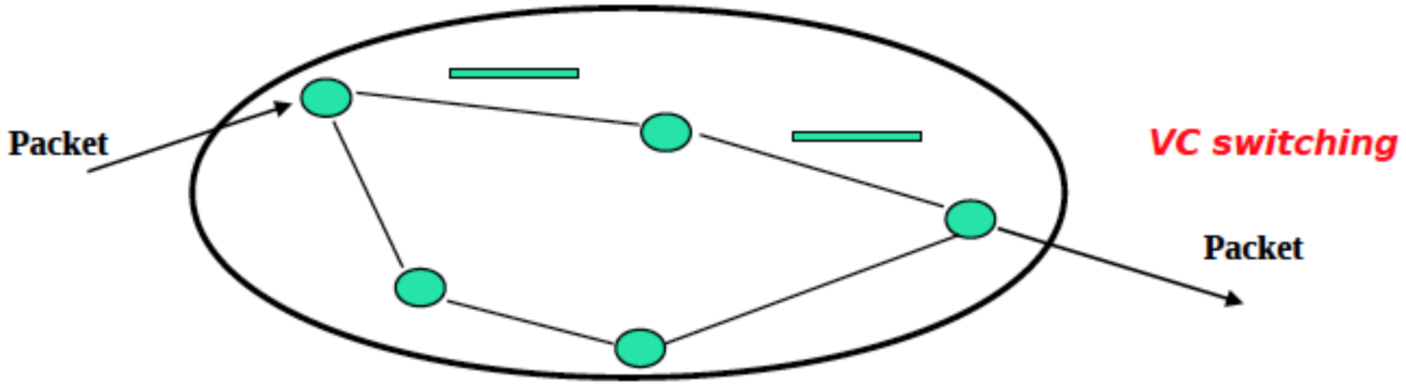
## Network layer connection and connection-less service

- **Datagram network** provides network-layer connectionless service
- **VC (virtual circuit) network** provides network-layer connection service

# Connection Oriented (VC Networks)

"source-to-dest path behaves much like *telephone circuit*"

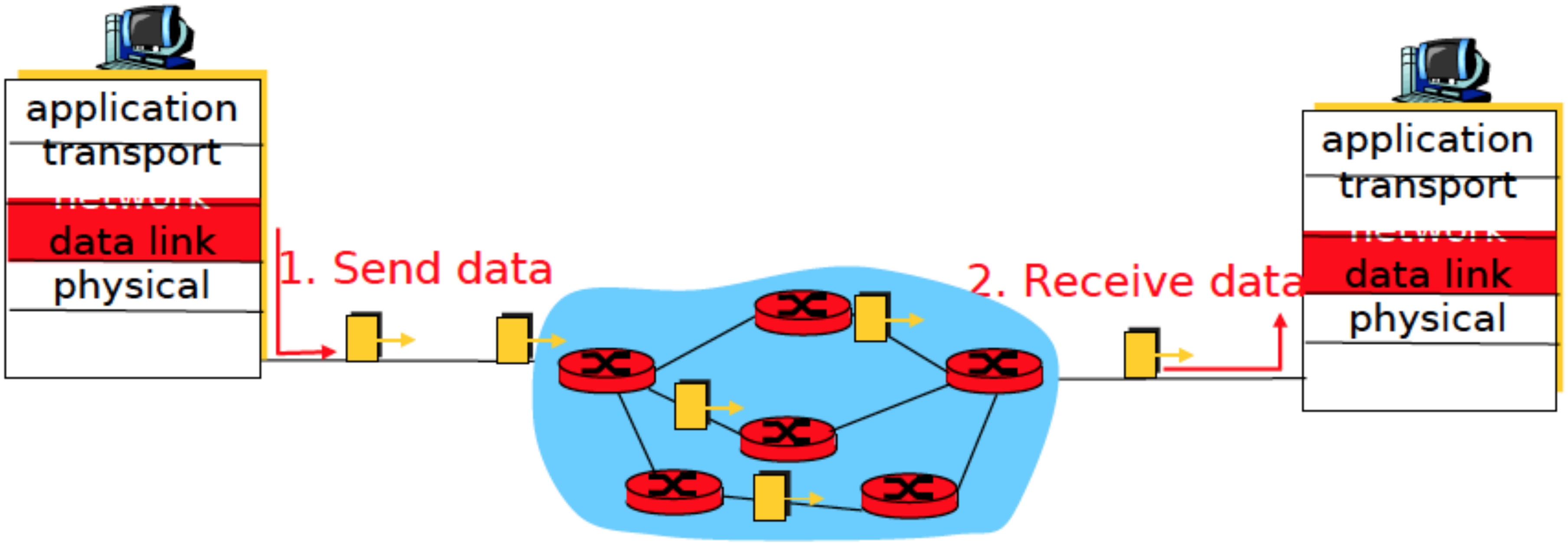
- performance-wise
- network actions along source-to-dest path

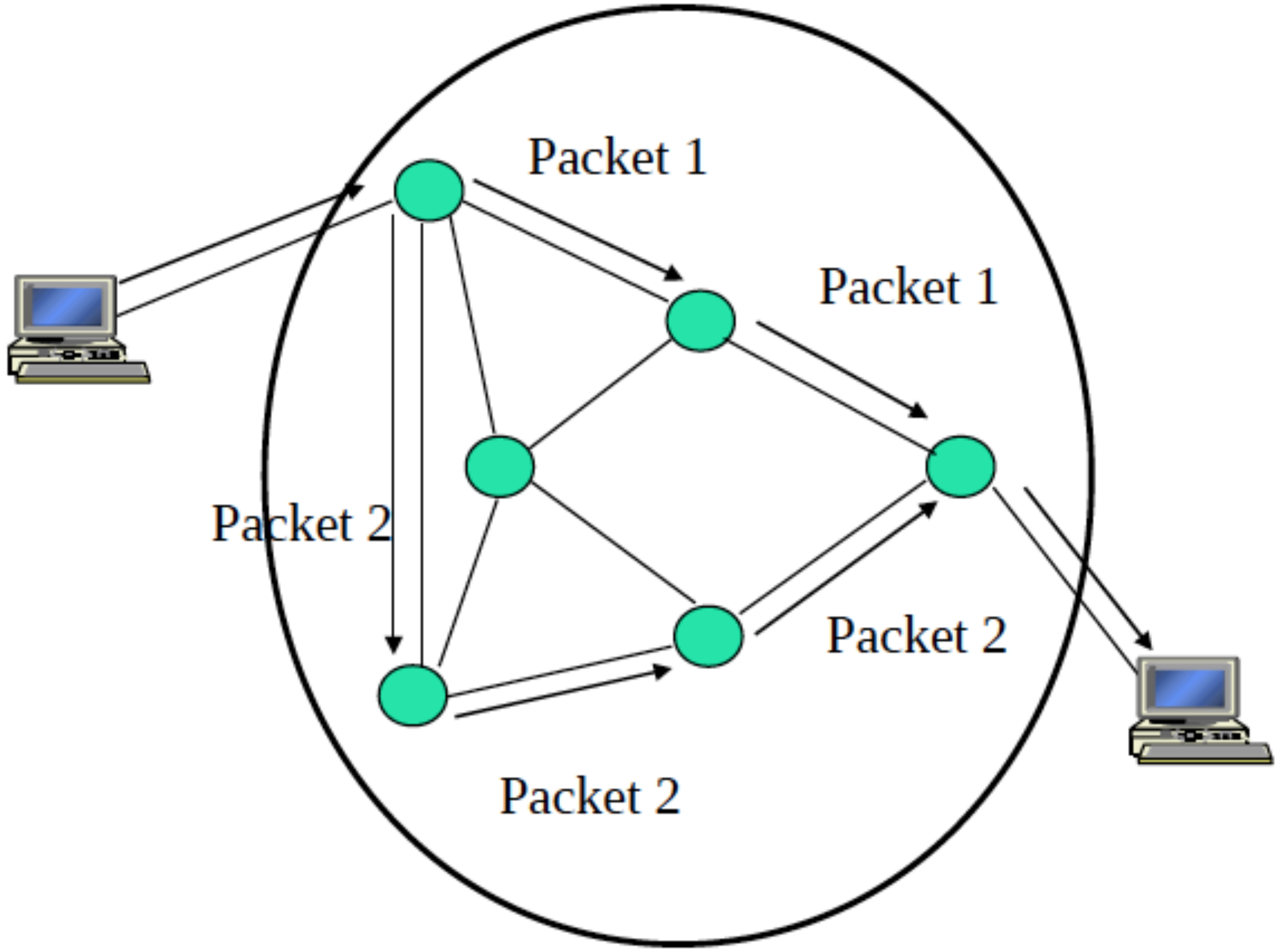


- call setup, teardown for each call *before* data can flow
- each packet carries VC identifier (not destination host address)
- routers on source-dest path maintains "state" for each passing connection
- link, router resources (bandwidth, buffers) may be *allocated* to VC

# Connectionless (Datagram Networks)

- no call setup at network layer
- routers: do not maintain state for e2e connections
  - no network-level concept of “connection”
- packets forwarded using destination host address
  - packets between the same source-dest pair may take different paths




Packet 1
Packet 1
Packet 2
Packet 2
Packet 2

# Routing

- ***Routing algorithm* :** Part of the Network Layer responsible for deciding on which output line to transmit an incoming packet.
  - Remember: For virtual circuit subnets the routing decision is made ONLY at setup

- **Algorithm properties**:
  - Efficiency, correctness, simplicity, robustness, stability, fairness, optimality, and scalability

# Key Network-Layer Functions

- *routing:* determine route taken by packets from source to dest
- *forwarding:* move packets from router's input to appropriate router output

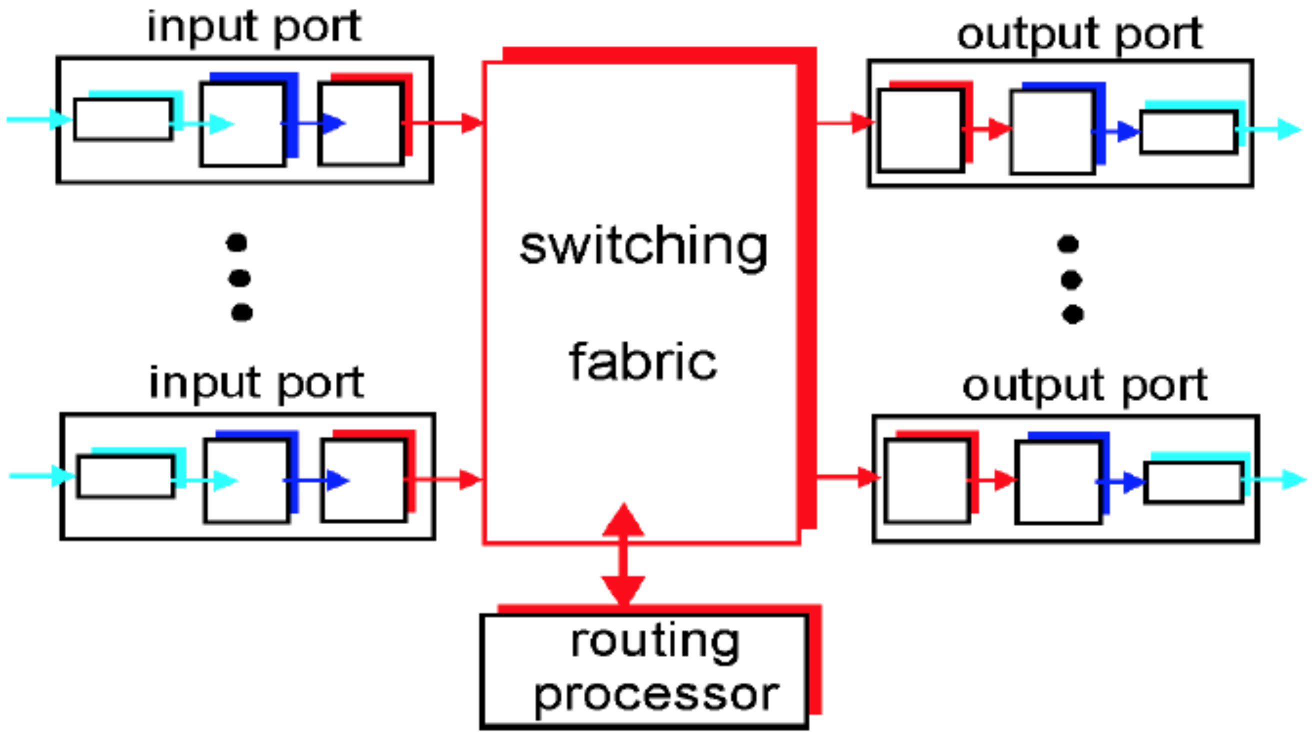
Analogy:

- *routing:* process of planning trip from source to dest
- *forwarding:* process of getting through single interchange

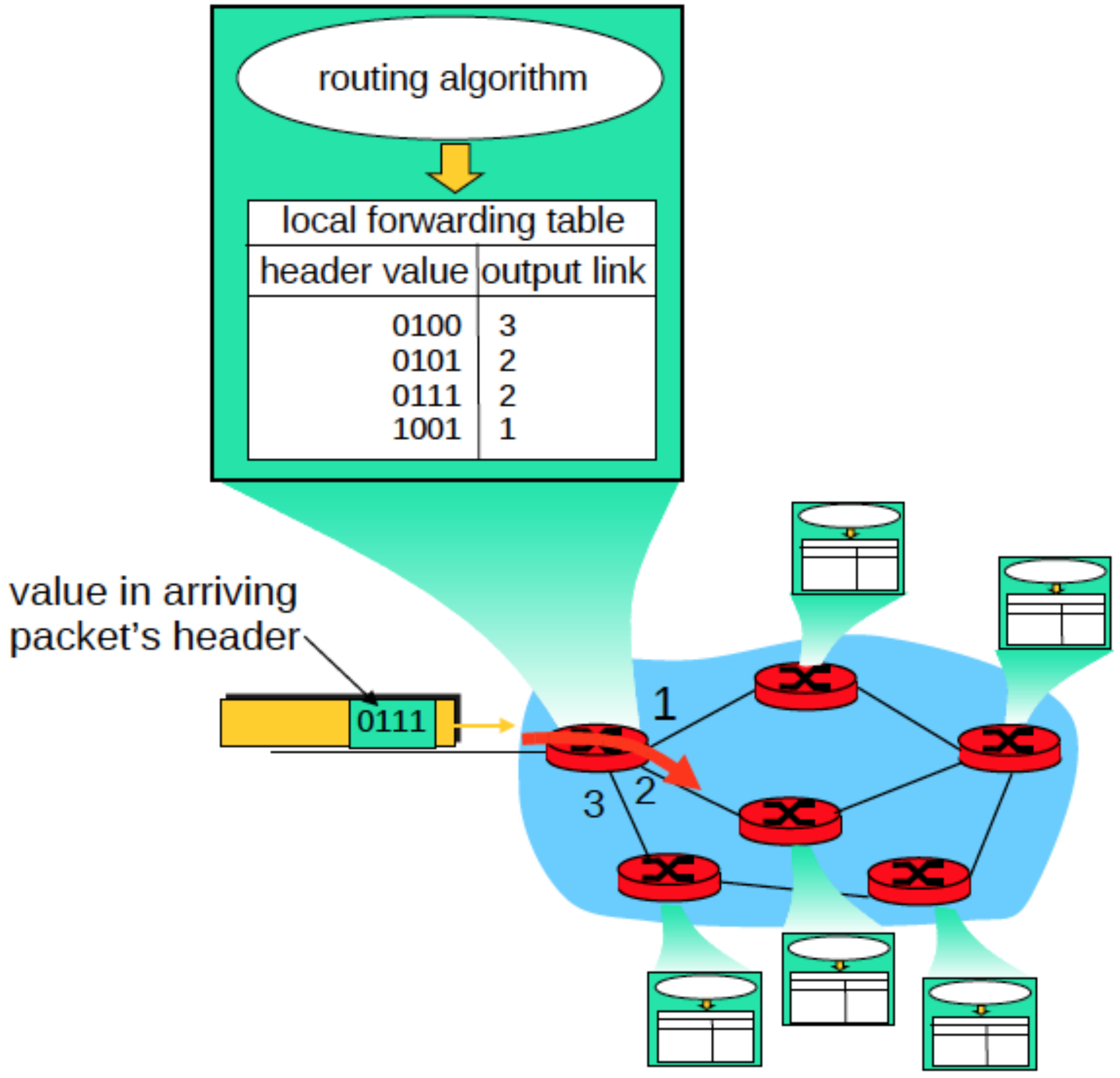
# Router Architecture Overview

Two key router functions:

- Run *routing algorithms/protocol* (RIP, OSPF, BGP)
- *Forward* datagrams from incoming to outgoing link



# Relation Between Forwarding and Routing



| local forwarding table | |
|---|---|
| header value | output link |
| 0100 | 3 |
| 0101 | 2 |
| 0111 | 2 |
| 1001 | 1 |

# Routing Table

| Destination address | Output port |
|---|---|
| | |
| 0785 | 7 |
| | |
| 1345 | 12 |
| | |
| 1566 | 6 |
| | |
| 2458 | 12 |
| | |

# Datagram forwarding table

| **Destination Address Range** | Link Interface |
|---|---|
| `11001000 00010111 00010000 00000000` through `11001000 00010111 00010111 11111111` | 0 |
| `11001000 00010111 00011000 00000000` through `11001000 00010111 00011000 11111111` | 1 |
| `11001000 00010111 00011001 00000000` through `11001000 00010111 00011111 11111111` | 2 |
| otherwise | 3 |

# Longest prefix matching

*longest prefix matching*

when looking for forwarding table entry for given destination address, use *longest* address prefix that matches destination address.

| Destination Address Range | Link interface |
|---|---|
| 11001000 00010111 00010*** ********* | 0 |
| 11001000 00010111 00011000 ********* | 1 |
| 11001000 00010111 00011*** ********* | 2 |
| otherwise | 3 |

examples:

DA: 11001000 00010111 00010110 10100001 which interface?

DA: 11001000 00010111 00011000 10101010 which interface?

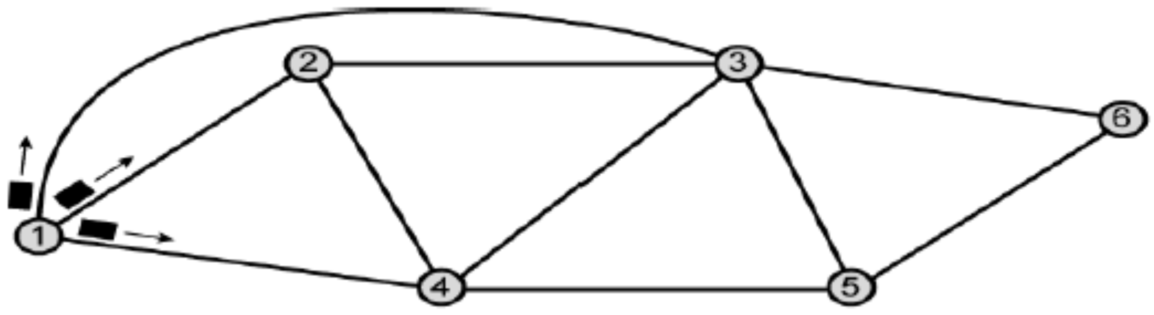
## Elements of Routing Techniques

- Performance criteria: Used for selection of routes
  - # of hops, cost, delay, throughput
- Decision Place:
  - Distributed (each node)/Centralized/Source routing
- Decision Time: Packet or VC basis
- Network Information Source:
  - None, local, adjacent node, all nodes
- Network Information Update:
  - Continuous, periodical, on change

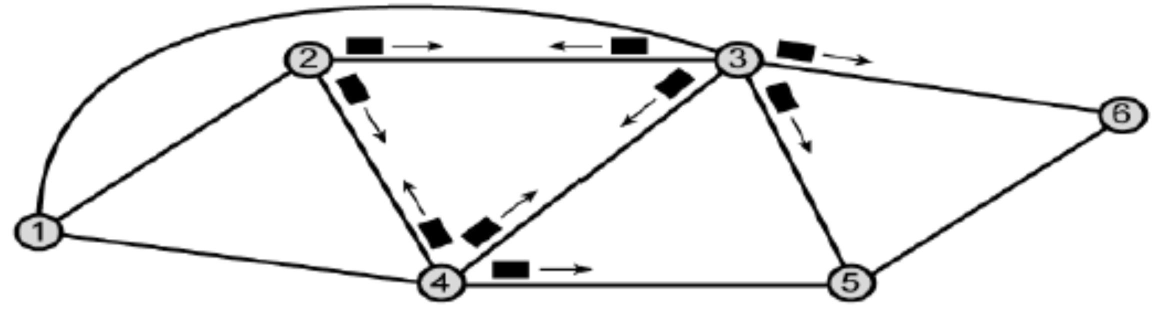
## Flooding

- No network info required
- Packet sent by node to every neighbor
- Incoming packets retransmitted on every link except incoming link
- Eventually a number of copies will arrive at destination
- Each packet is uniquely numbered so duplicates can be discarded
- Nodes can remember packets already forwarded to keep network load in bounds
- Can include a hop count in packets

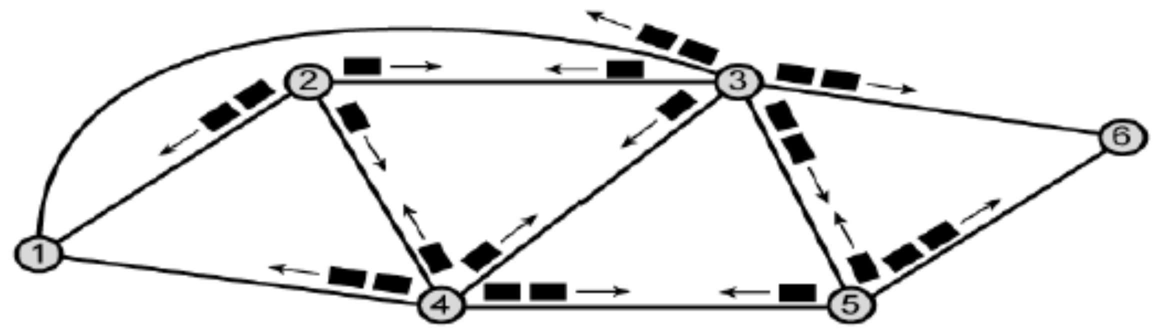
# Flooding Example



(a) First hop



(b) Second hop



(c) Third hop

## Properties of Flooding

- All possible routes are tried
  - Very robust
- At least one packet will have taken minimum hop count route
  - Can be used to set up virtual circuit
- All nodes are visited
  - Useful to distribute information (e.g. routing)

## Adaptive Routing

- Distance vector routing:
  - Each router maintain a table (i.e, a vector) giving the best known distance to each destination and which line to use to get there.
  - These tables are updated by exchanging information with the neighbors. (E.g., Bellman-Ford Algorithm)
- Link state routing:
  - Each router must do the following:
    - Discover its neighbors and learn their network addresses.
    - Measure the delay or cost to each of its neighbors.
    - Construct a packet telling all it has just learned.
    - Send this packet to all other routers.
    - (The complete topology and all delays are experimentally measured and distributed to every router.) Then, compute the shortest path to every other router.

# Shortest Path Routing

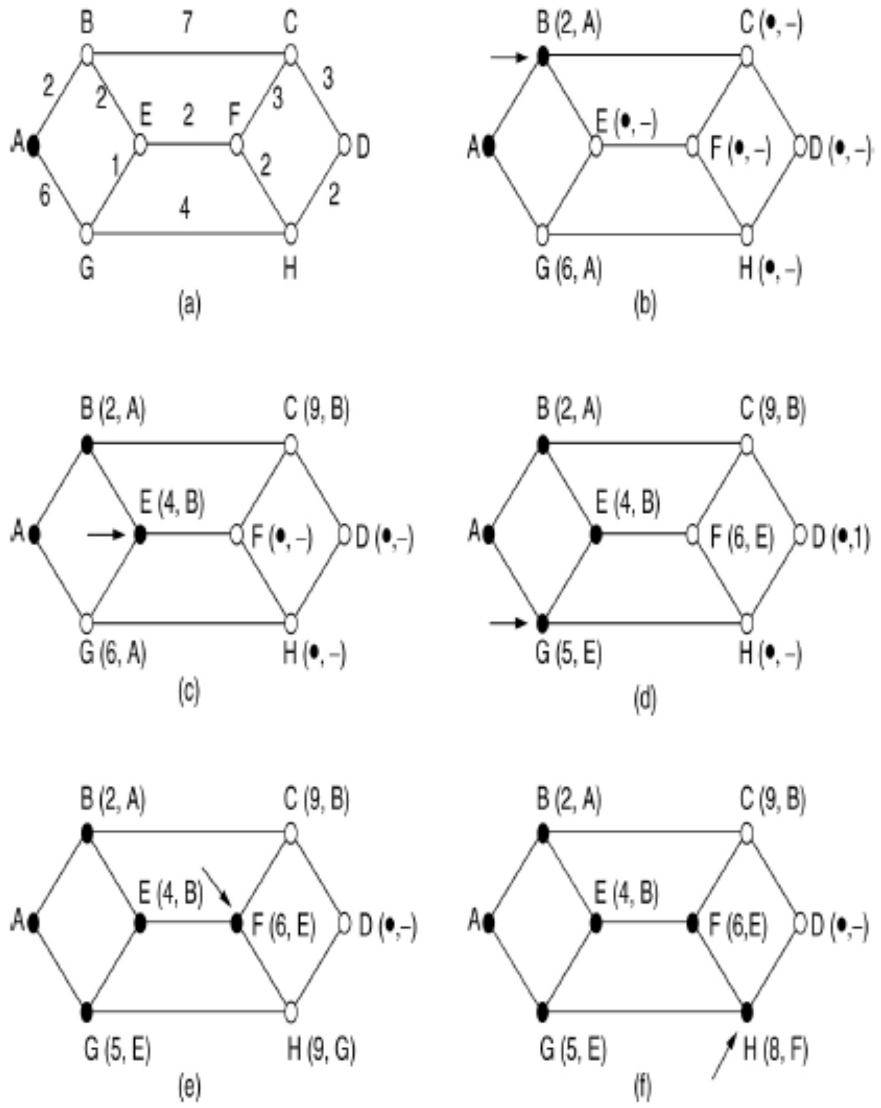
1. Bellman-Ford Algorithm [Distance Vector]
2. Dijkstra's Algorithm [Link State]

***What does it mean to be the shortest (or optimal) route?***

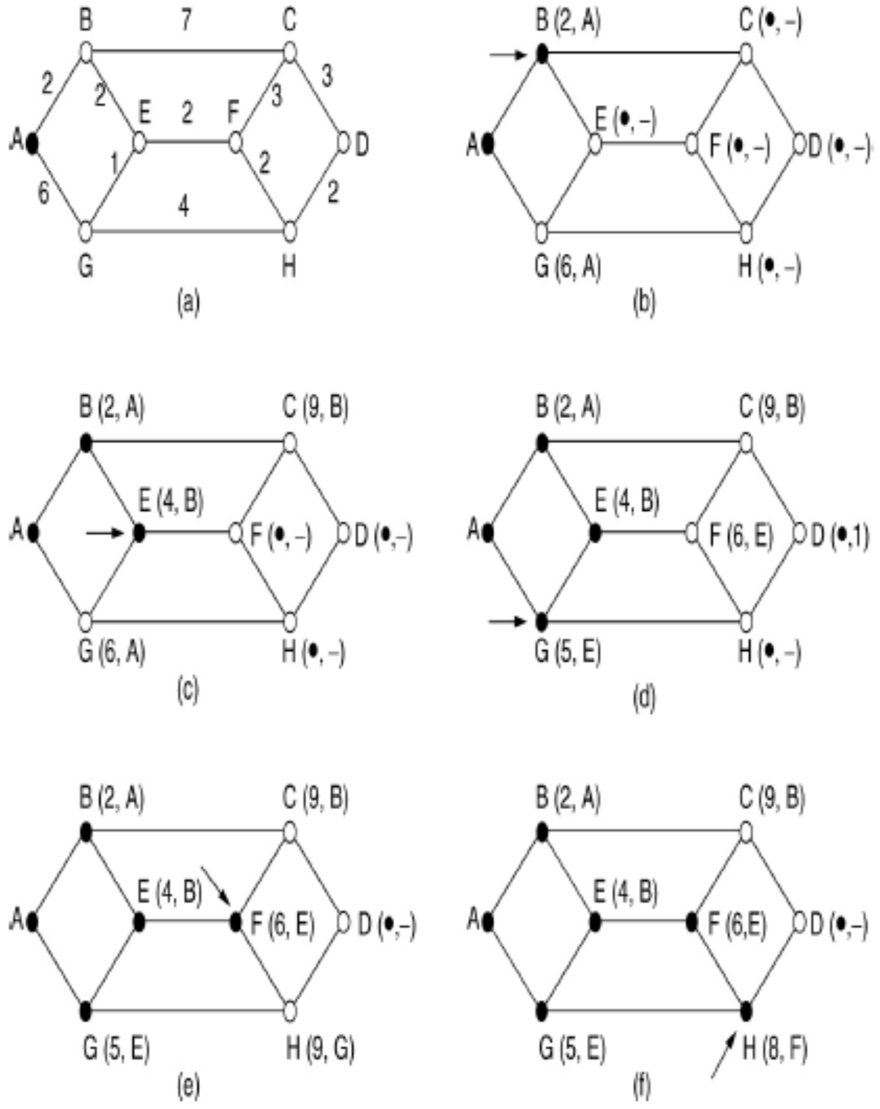
- Minimize mean packet delay
- Maximize the network throughput
- Minimize the number of hops along the path

## Example: Dijkstra's Shortest Path Algorithm

- Want to find the shortest path from A to D.
- Each node is labeled (in parentheses) with its distance from the source node along the best known path
  - Initially, no paths are known, so all nodes are labeled with infinity
  - A label may be either **tentative** or **permanent**.
  - Initially, all labels are tentative.
- Start out by marking node A as permanent
- Then examine, in turn, each of nodes adjacent to A (the working node), re-labeling each one with the distance to A
- Whenever a node is relabeled, label it with the node from which the probe was made so that the final path can be reconstructed later
- Having examined each of the nodes adjacent to A, examine all the tentatively labeled nodes in the whole graph and make the one with the smallest label permanent
- This one becomes the new working node



- Now start at B and examine all nodes adjacent to it.
- If the sum of the label on B and the distance from B to the node being considered is less than the label on that node → a shorter path, so the node is relabeled
- After all nodes adjacent to working node have been inspected and the tentative labels changed if possible, the entire graph is searched for the tentatively-labeled node with the smallest value
- This node is made permanent and becomes the working node for the next round.



## Distance vector algorithm

*Bellman-Ford equation (dynamic programming)*

let

$d_x(y)$ := cost of least-cost path from x to y

then

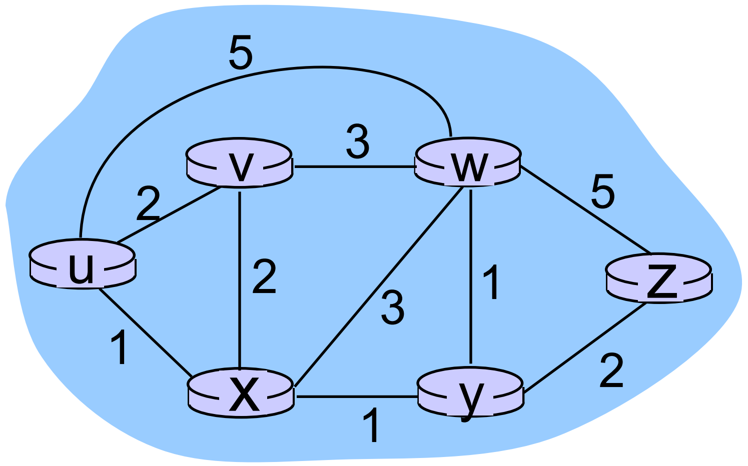
$$d_x(y) = \min_v \{c(x,v) + d_v(y)\}$$

cost from neighbor v to destination y

cost to neighbor v

*min* taken over all neighbors v of x

# Bellman-Ford example



clearly, $d_v(z) = 5$, $d_x(z) = 3$, $d_w(z) = 3$

B-F equation says:

$$
\begin{aligned}
d_u(z) &= \min \{ c(u,v) + d_v(z), \\
&\qquad c(u,x) + d_x(z), \\
&\qquad c(u,w) + d_w(z) \} \\
&= \min \{2 + 5, \\
&\qquad 1 + 3, \\
&\qquad 5 + 3\} = 4
\end{aligned}
$$

node achieving minimum is next
hop in shortest path, used in forwarding table

# Distance vector algorithm

- $D_x(y)$ = estimate of least cost from x to y
  - x maintains distance vector $\mathbf{D}_x = [D_x(y): y \in N\ ]$
- node x:
  - knows cost to each neighbor v: $c(x,v)$
  - maintains its neighbors' distance vectors. For each neighbor v, x maintains
    $\mathbf{D}_v = [D_v(y): y \in N\ ]$

## Distance vector algorithm

*key idea:*

- from time-to-time, each node sends its own distance vector estimate to neighbors
- when x receives new DV estimate from neighbor, it updates its own DV using B-F equation:

$$D_x(y) \leftarrow \min_v\{c(x,v) + D_v(y)\} \quad \text{for each node } y \in N$$

- under minor, natural conditions, the estimate $D_x(y)$ *converge to the actual least cost* $d_x(y)$

## Distance vector algorithm

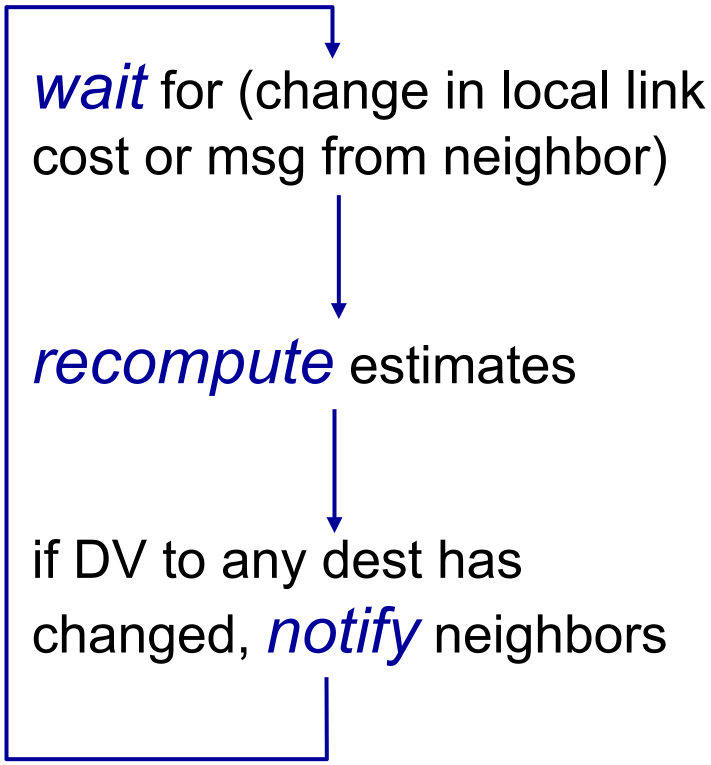
*iterative, asynchronous:* each local iteration caused by:

- local link cost change
- DV update message from neighbor

*distributed:*

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

*each node:*



*wait* for (change in local link cost or msg from neighbor)

*recompute* estimates

if DV to any dest has changed, *notify* neighbors

$$D_x(y) = \min\{c(x,y) + D_y(y), c(x,z) + D_z(y)\}$$
$$= \min\{2+0 , 7+1\} = 2$$

$$D_x(z) = \min\{c(x,y) + D_y(z), c(x,z) + D_z(z)\}$$
$$= \min\{2+1 , 7+0\} = 3$$

**node x table**

| from \ cost to | x | y | z |
|---|---|---|---|
| x | 0 | 2 | 7 |
| y | ∞ | ∞ | ∞ |
| z | ∞ | ∞ | ∞ |

| from \ cost to | x | y | z |
|---|---|---|---|
| x | 0 | 2 | 3 |
| y | 2 | 0 | 1 |
| z | 7 | 1 | 0 |

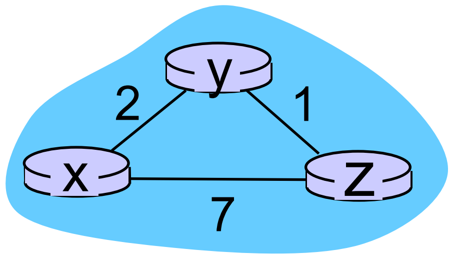
**node y table**

| from \ cost to | x | y | z |
|---|---|---|---|
| x | ∞ | ∞ | ∞ |
| y | 2 | 0 | 1 |
| z | ∞ | ∞ | ∞ |

**node z table**

| from \ cost to | x | y | z |
|---|---|---|---|
| x | ∞ | ∞ | ∞ |
| y | ∞ | ∞ | ∞ |
| z | 7 | 1 | 0 |

time



$$D_x(y) = \min\{c(x,y) + D_y(y),\ c(x,z) + D_z(y)\} = \min\{2+0\ ,\ 7+1\} = 2$$

$$D_x(z) = \min\{c(x,y) + D_y(z),\ c(x,z) + D_z(z)\} = \min\{2+1\ ,\ 7+0\} = 3$$

**node x table**

| | | cost to | |
|---|---|---|---|
| | x | y | z |
| from x | 0 | 2 | 7 |
| y | ∞ | ∞ | ∞ |
| z | ∞ | ∞ | ∞ |

| | | cost to | |
|---|---|---|---|
| | x | y | z |
| from x | 0 | 2 | 3 |
| y | 2 | 0 | 1 |
| z | 7 | 1 | 0 |

| | | cost to | |
|---|---|---|---|
| | x | y | z |
| from x | 0 | 2 | 3 |
| y | 2 | 0 | 1 |
| z | 3 | 1 | 0 |

**node y table**

| | | cost to | |
|---|---|---|---|
| | x | y | z |
| from x | ∞ | ∞ | ∞ |
| y | 2 | 0 | 1 |
| z | ∞ | ∞ | ∞ |

| | | cost to | |
|---|---|---|---|
| | x | y | z |
| from x | 0 | 2 | 7 |
| y | 2 | 0 | 1 |
| z | 7 | 1 | 0 |

| | | cost to | |
|---|---|---|---|
| | x | y | z |
| from x | 0 | 2 | 3 |
| y | 2 | 0 | 1 |
| z | 3 | 1 | 0 |

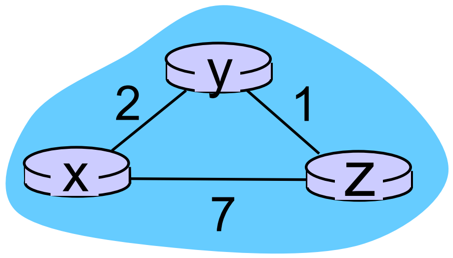
**node z table**

| | | cost to | |
|---|---|---|---|
| | x | y | z |
| from x | ∞ | ∞ | ∞ |
| y | ∞ | ∞ | ∞ |
| z | 7 | 1 | 0 |

| | | cost to | |
|---|---|---|---|
| | x | y | z |
| from x | 0 | 2 | 7 |
| y | 2 | 0 | 1 |
| z | 3 | 1 | 0 |

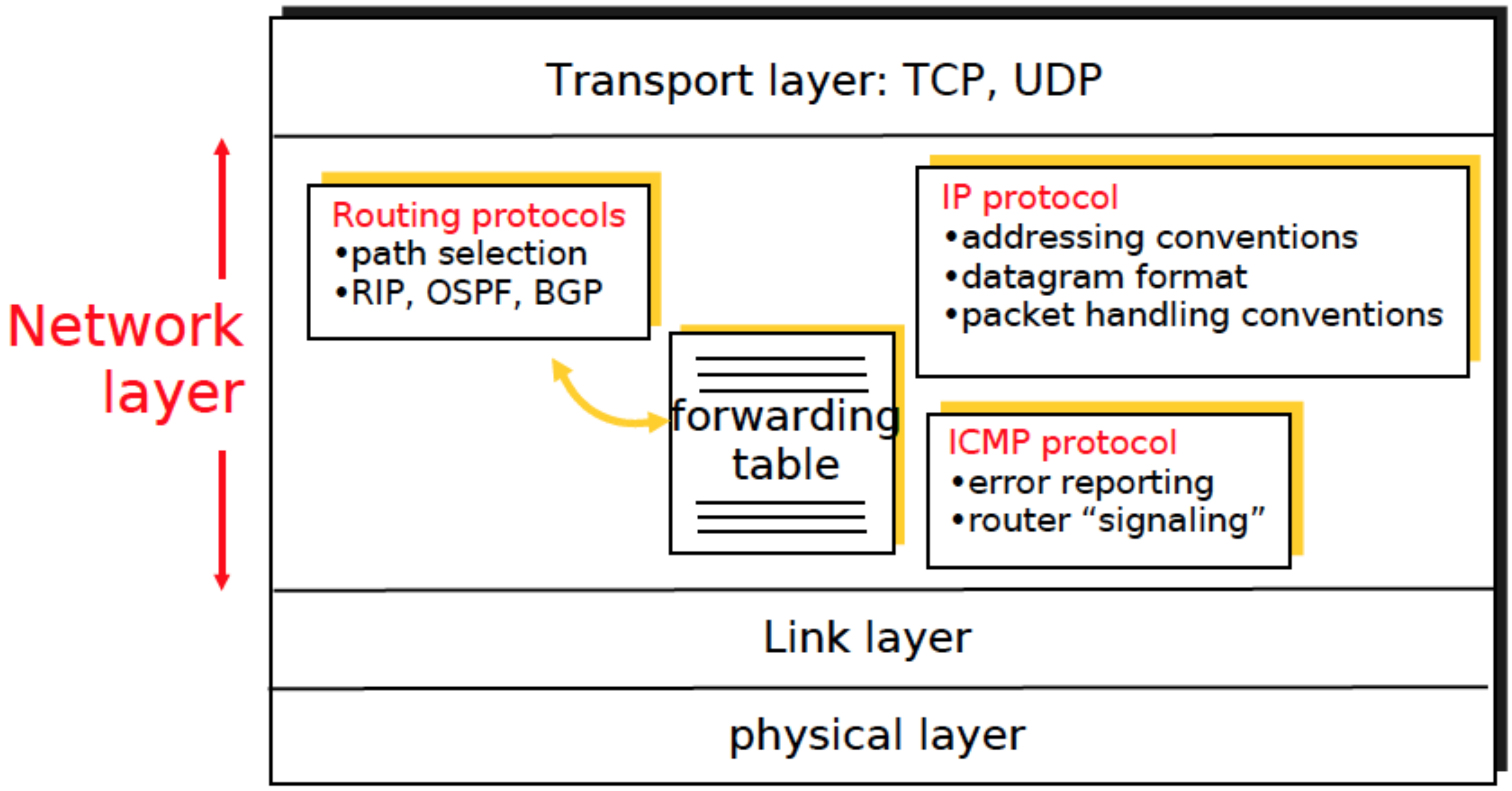
| | | cost to | |
|---|---|---|---|
| | x | y | z |
| from x | 0 | 2 | 3 |
| y | 2 | 0 | 1 |
| z | 3 | 1 | 0 |

time

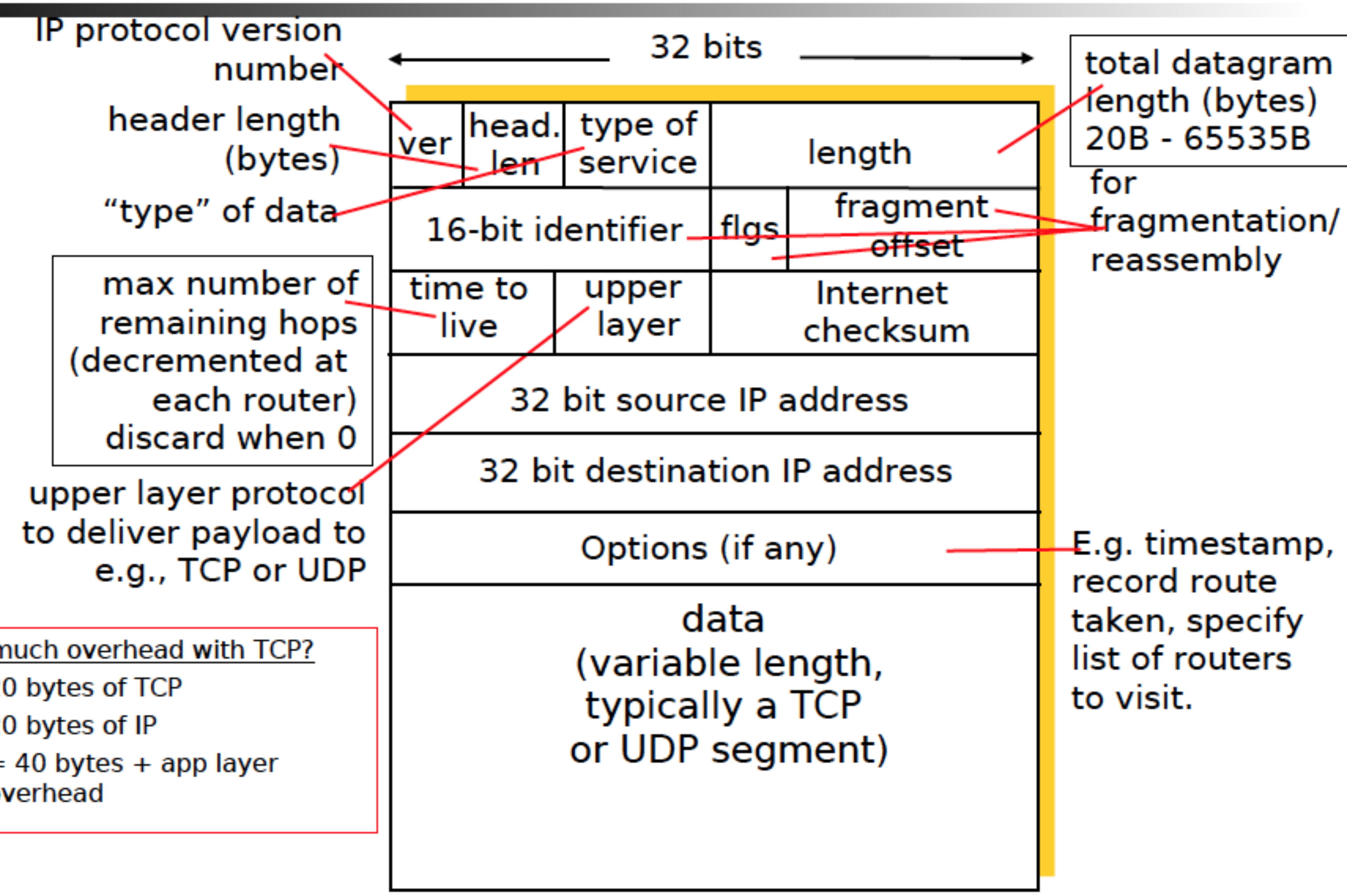


# The Internet Network layer

Host, router network layer functions:



# IP Datagram

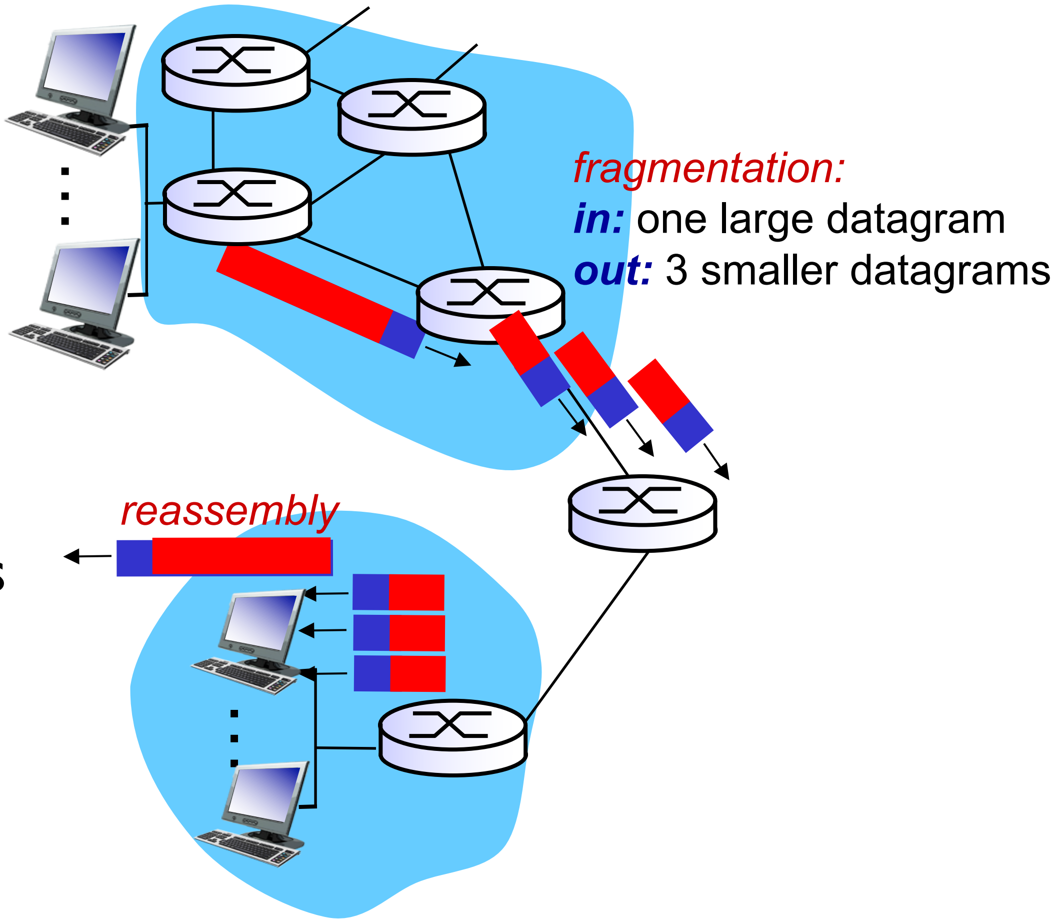

IP protocol version number
header length (bytes)
"type" of data
32 bits
ver
head. len
type of service
length
16-bit identifier
flgs
fragment offset
time to live
upper layer
Internet checksum
32 bit source IP address
32 bit destination IP address
Options (if any)
data (variable length, typically a TCP or UDP segment)
total datagram length (bytes) 20B - 65535B
for fragmentation/ reassembly
max number of remaining hops (decremented at each router) discard when 0
upper layer protocol to deliver payload to e.g., TCP or UDP
E.g. timestamp, record route taken, specify list of routers to visit.

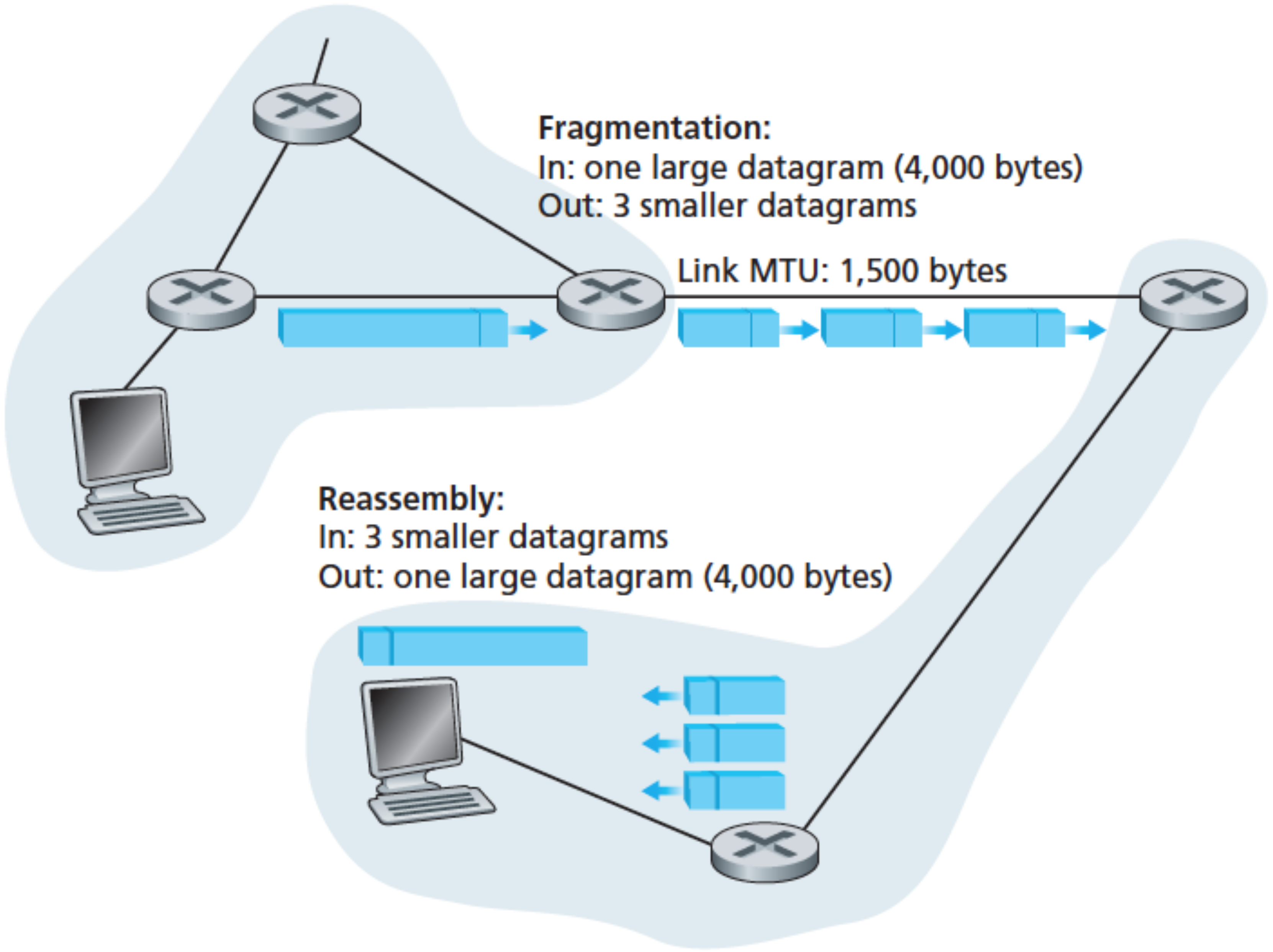
how much overhead with TCP?

- 20 bytes of TCP
- 20 bytes of IP
- = 40 bytes + app layer overhead

# IP fragmentation, reassembly

- network links have MTU (max.transfer size) - largest possible link-level frame
  - different link types, different MTUs
- large IP datagram divided ("fragmented") within net
  - one datagram becomes several datagrams
  - "reassembled" only at final destination
  - IP header bits used to identify, order related fragments



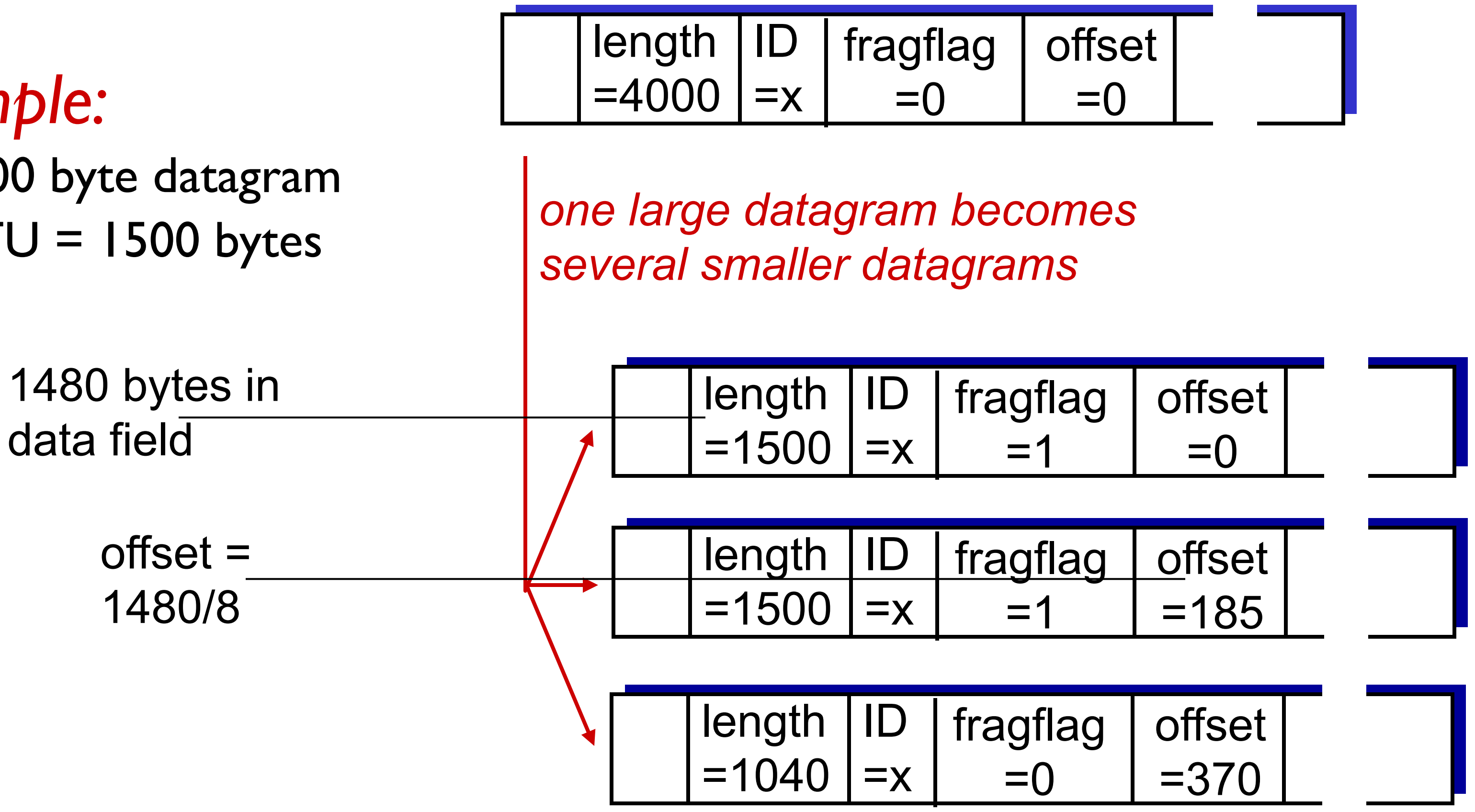

Fragmentation:
In: one large datagram (4,000 bytes)
Out: 3 smaller datagrams
Link MTU: 1,500 bytes
Reassembly:
In: 3 smaller datagrams
Out: one large datagram (4,000 bytes)

| Fragment | Bytes | ID | Offset | Flag |
|---|---|---|---|---|
| 1st fragment | 1,480 bytes in the data field of the IP datagram | identification = 777 | offset = 0 (meaning the data should be inserted beginning at byte 0) | flag = 1 (meaning there is more) |
| 2nd fragment | 1,480 bytes of data | identification = 777 | offset = 185 (meaning the data should be inserted beginning at byte 1,480. Note that 185 · 8 = 1,480) | flag = 1 (meaning there is more) |
| 3rd fragment | 1,020 bytes (= 3,980–1,480–1,480) of data | identification = 777 | offset = 370 (meaning the data should be inserted beginning at byte 2,960. Note that 370 · 8 = 2,960) | flag = 0 (meaning this is the last fragment) |

# IP fragmentation, reassembly

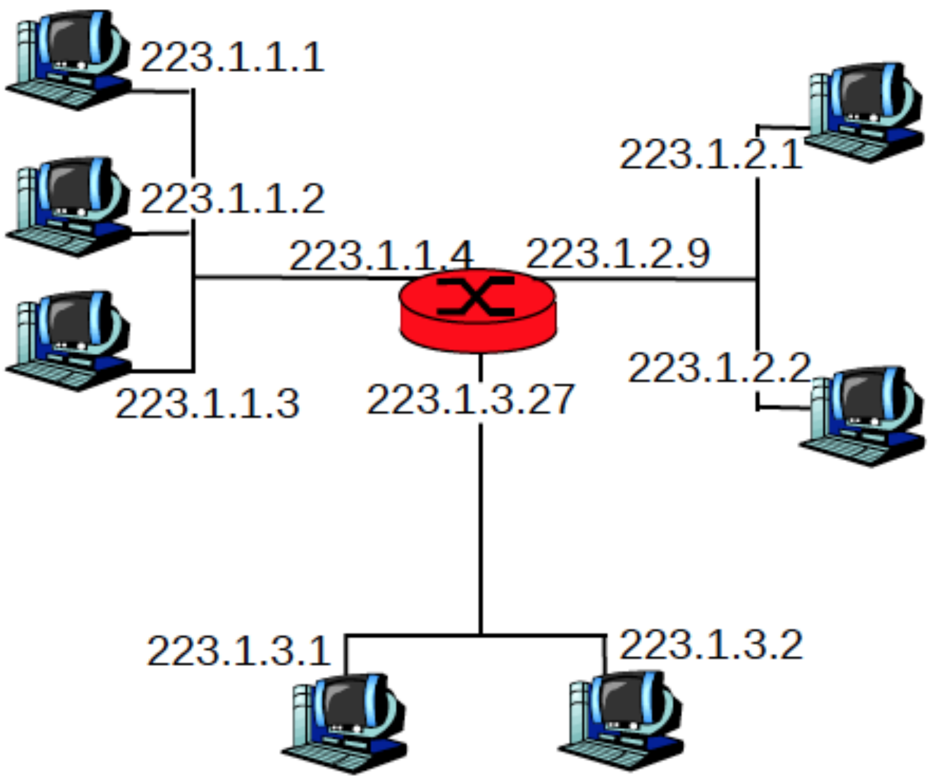
*example:*

- 4000 byte datagram
- MTU = 1500 bytes



# IP Addressing

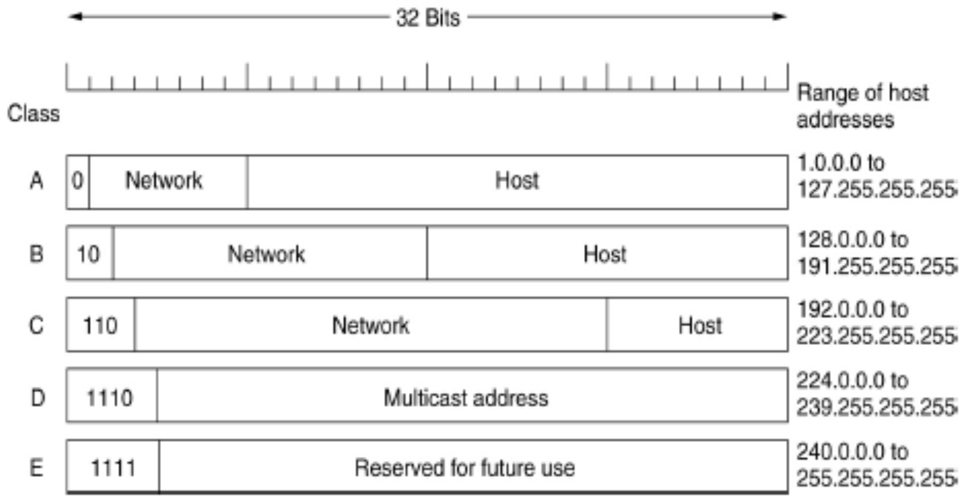
- IP address: 32-bit identifier for host, router *interface*
- *interface:* connection between host/router and physical link
  - routers typically have multiple interfaces
  - host may have multiple interfaces
  - IP addresses associated with each interface

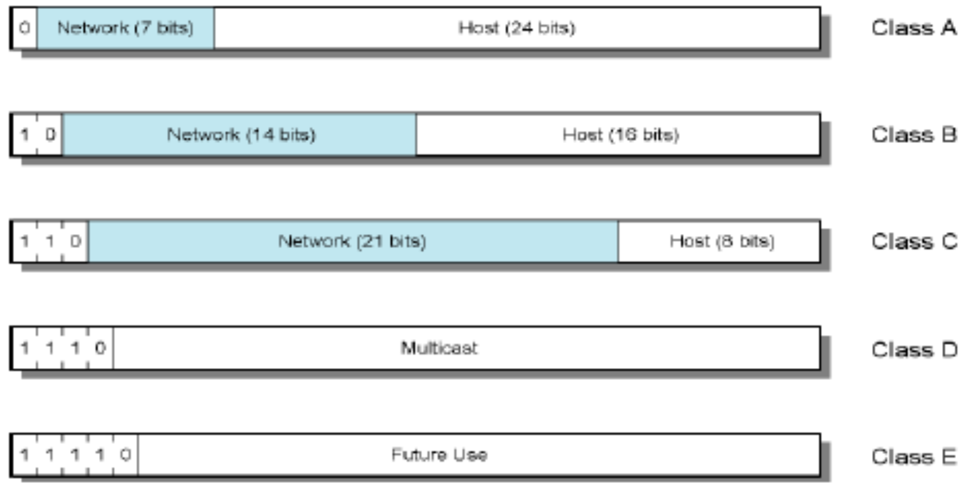


$$223.1.1.1 = \underbrace{11011111}_{223}\ \underbrace{00000001}_{1}\ \underbrace{00000001}_{1}\ \underbrace{00000001}_{1}$$

# IP Addresses

- 32 bit global internet address
- Network part and host part
- **Class A**
  - Start with binary 0
  - All 0 reserved
  - 01111111 (127) reserved for loopback
  - Range 1.x.x.x to 126.x.x.x
  - All allocated
- **Class B**
  - Start 10
  - Range 128.x.x.x to 191.x.x.x
  - Second Octet also included in network address
  - $2^{14}$ = 16,384 class B addresses
  - All allocated
- **Class C**
  - Start 110
  - Range 192.x.x.x to 223.x.x.x
  - Second and third octet also part of network address
  - $2^{21}$ = 2,097,152 addresses
  - Nearly all allocated


32 Bits
Class
Range of host addresses
A
0
Network
Host
1.0.0.0 to 127.255.255.255
B
10
Network
Host
128.0.0.0 to 191.255.255.255
C
110
Network
Host
192.0.0.0 to 223.255.255.255
D
1110
Multicast address
224.0.0.0 to 239.255.255.255
E
1111
Reserved for future use
240.0.0.0 to 255.255.255.255

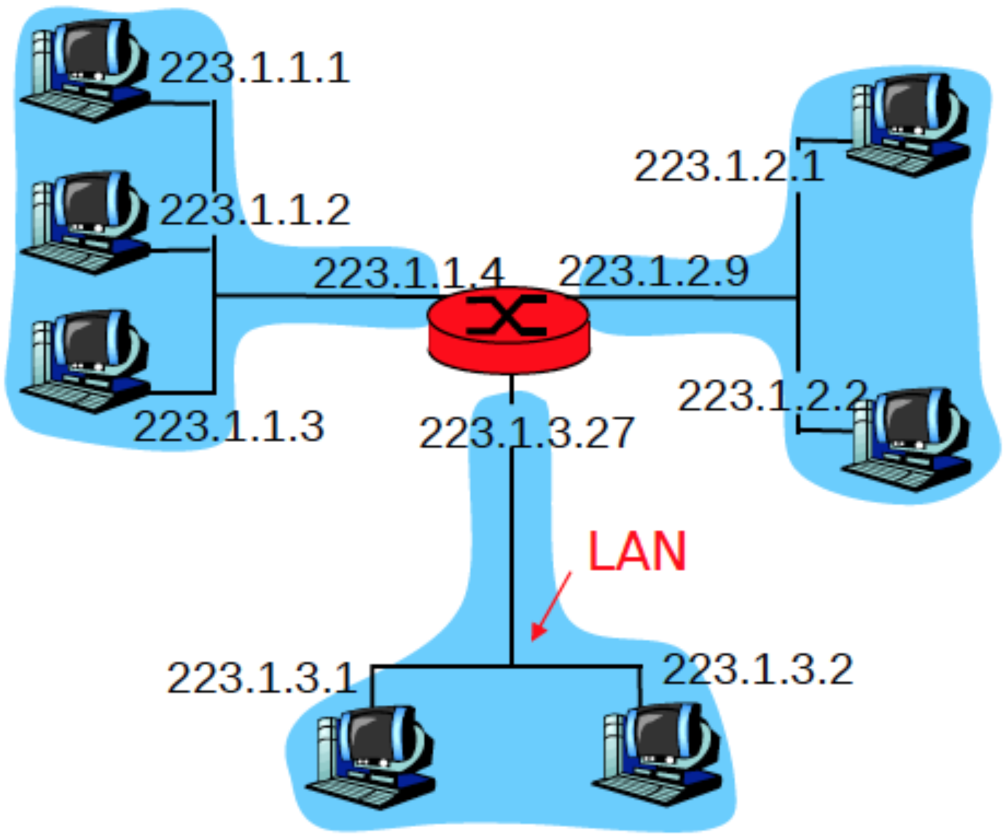


# Subnets and Subnet Masks

- Classes are too coarse
  - Additional partitioning may be needed
- Each LAN assigned subnet number
  - Host portion of address partitioned into subnet number and host number
  - ***Subnet mask*** *indicates which bits are subnet number and which are host number*
- Consider a class B address 144.122.x.x has 16 bits network part 16 bits hosts part
  - $2^{16}$ hosts → may be unnecessary
  - Difficult to keep routing for $2^{16}$ entries
  - Additional partitions may be required (e.g., departments etc.)
  - e.g. use 6 bits for subnet, and 10 bits for hosts
    - Network+subnet=16+6=22 bits=subnetmask
    - 64 subnets with 1022 hosts (0, and 255 are **not** available as host octet)
  - IP address **AND** Subnetmask = Network Address

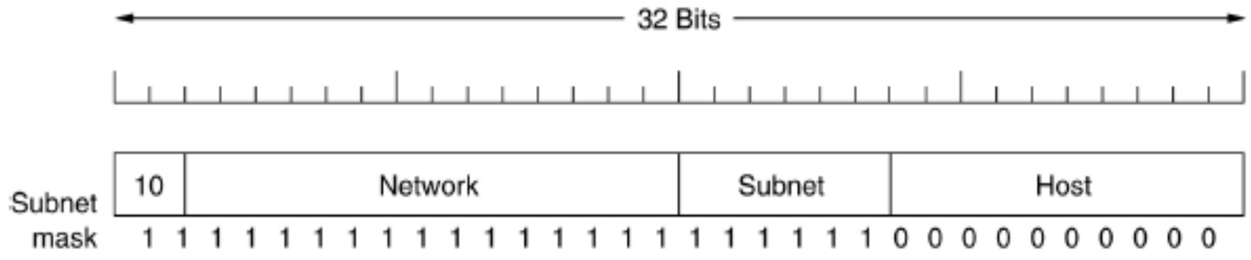
# Subnets

- IP address:
  - subnet part (high order bits)
  - host part (low order bits)
- *What's a subnet ?*
  - device interfaces with the same subnet part of IP address
  - can physically reach each other without intervening router



network consisting of 3 subnets

# Subnets



(**A class B network (130.50.x.x) subnetted into 64 subnets**)

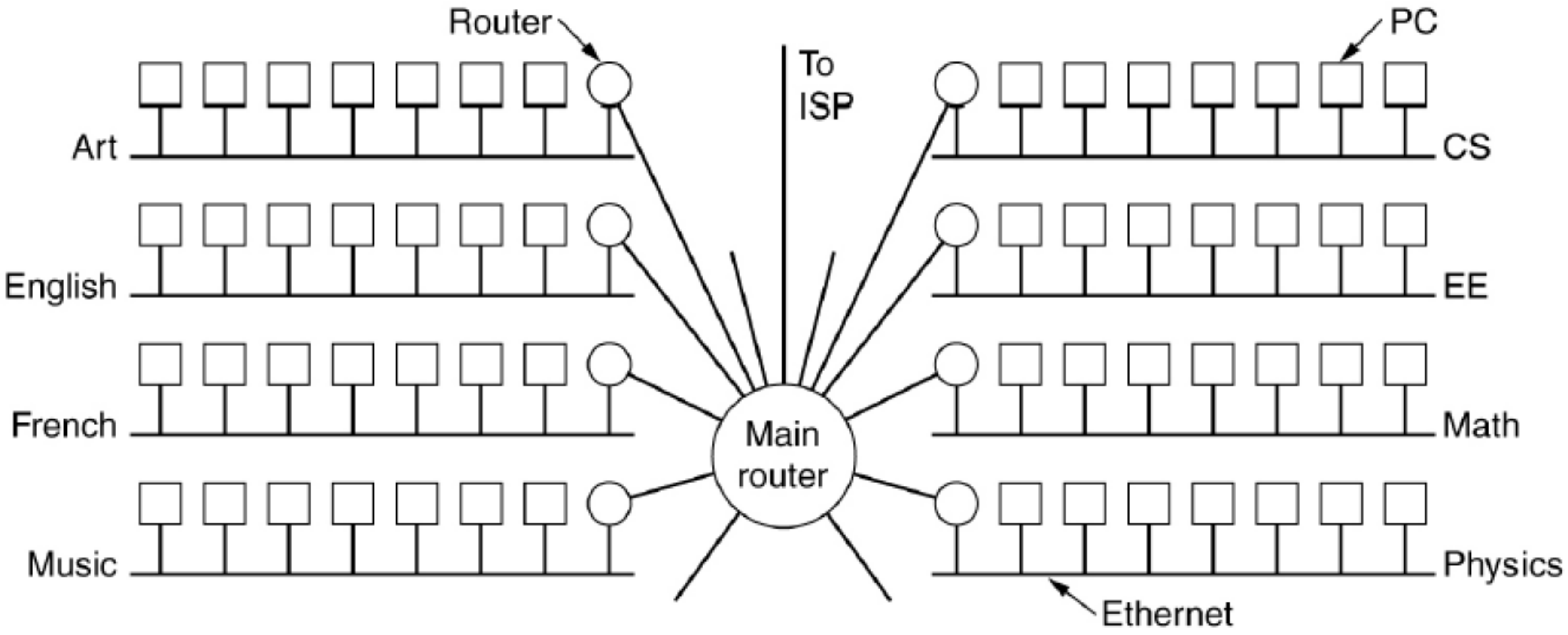
- To implement subnetting, the main router needs a subnet mask that indicates the split between network + subnet number and host.
- 255.255.252.0 or /22 to indicate that the subnet mask is 22 bits long.
- Subnet 0 might use IP addresses starting at 130.50.0.1; Subnet 1 might start at 130.50.4.1; Subnet 3 might start at 130.50.8.1; and so on

```
 Subnet 0: 10000010 00110010 000000|00 00000001
 Subnet 1: 10000010 00110010 000001|00 00000001
 Subnet 2: 10000010 00110010 000010|00 00000001
                       ...
Subnet 63: 10000010 00110010 111111|00 00000001
```

- When an IP packet arrives, router does a Boolean AND with network's subnet mask to get rid of host number and look up the resulting address in its tables
  - e.g. packet addressed to 130.50.15.6 and arriving at the main router is ANDed with the subnet mask 255.255.252.0/22 to give the address 130.50.12.0.
  - This address is looked up to find out which output line to use to get to Subnet 3

## Subnets

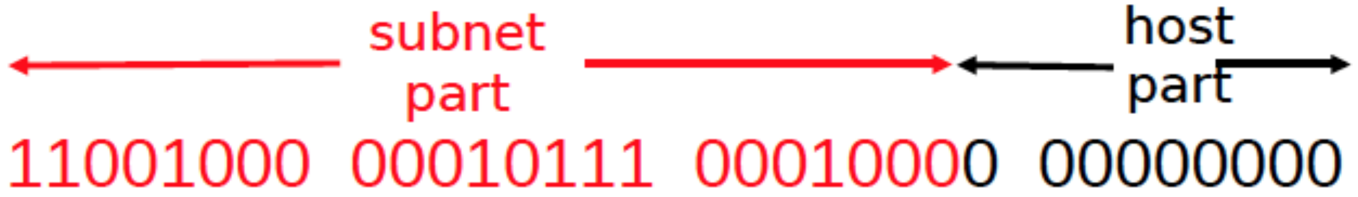
A campus network consisting of LANs for various departments.



## IP addressing: CIDR

CIDR: Classless InterDomain Routing

- subnet portion of address of arbitrary length
- address format: a.b.c.d/x, where x is # bits in subnet portion of address



200.23.16.0/23

# CIDR – Classless InterDomain Routing

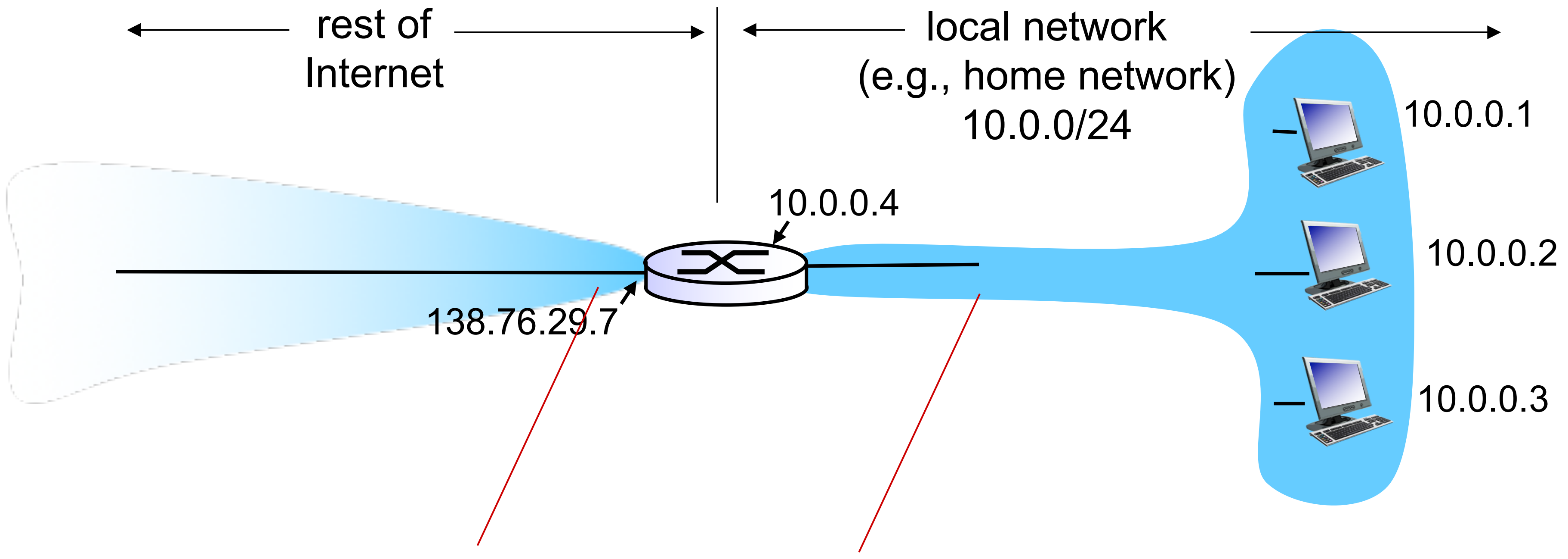
| University | First address | Last address | How many | Written as |
|---|---|---|---|---|
| Cambridge | 194.24.0.0 | 194.24.7.255 | 2048 | 194.24.0.0/21 |
| Edinburgh | 194.24.8.0 | 194.24.11.255 | 1024 | 194.24.8.0/22 |
| (Available) | 194.24.12.0 | 194.24.15.255 | 1024 | 194.24.12/22 |
| Oxford |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

| Address | Mask |
|---|---|
| C: 11000010 00011000 00000000 00000000 | 11111111 11111111 11111000 00000000 |
| E: 11000010 00011000 00001000 00000000 | 11111111 11111111 11111100 00000000 |
| O: 11000010 00011000 00010000 00000000 | 11111111 11111111 11110000 00000000 |

- Consider when a packet comes in addressed to 194.24.17.4, which in binary is represented as the following 32-bit string 11000010 00011000 00010001 00000100
- First it is Boolean ANDed with the Cambridge mask to get
  11000010 00011000 00010000 00000000
- This value does not match the Cambridge base address, so the original address is next ANDed with the Edinburgh mask to get
  11000010 00011000 00010000 00000000
- This value does not match the Edinburgh base address, so Oxford is tried next, yielding
  11000010 00011000 00010000 00000000
- This value does match the Oxford base. If no longer matches are found farther down the table, the Oxford entry is used and the packet is sent along the line named in it.
  - what if 192.168.20.19 is the destination; 192.168.20.16/28 and 192.168.0.0/16 are two entries in the routing table?
  - (longest prefix matching)

*check http://www.subnet-calculator.com/*

# NAT: network address translation



*all* datagrams *leaving* local network have *same* single source NAT IP address: 138.76.29.7,different source port numbers

datagrams with source or destination in this network have 10.0.0/24 address for source, destination (as usual)

# NAT: network address translation

*motivation:* local network uses just one IP address as far as outside world is concerned:

- range of addresses not needed from ISP:  just one IP address for all devices
- can change addresses of devices in local network without notifying outside world
- can change ISP without changing addresses of devices in local network
- devices inside local net not explicitly addressable, visible by outside world (a security plus)

# NAT: network address translation

*implementation:* NAT router must:

- *outgoing datagrams: replace* (source IP address, port #) of every outgoing datagram to (NAT IP address, new port #)
  . . . remote clients/servers will respond using (NAT IP address, new port #) as destination addr

- *remember (in NAT translation table)* every (source IP address, port #) to (NAT IP address, new port #) translation pair

- *incoming datagrams: replace* (NAT IP address, new port #) in dest fields of every incoming datagram with corresponding (source IP address, port #) stored in NAT table

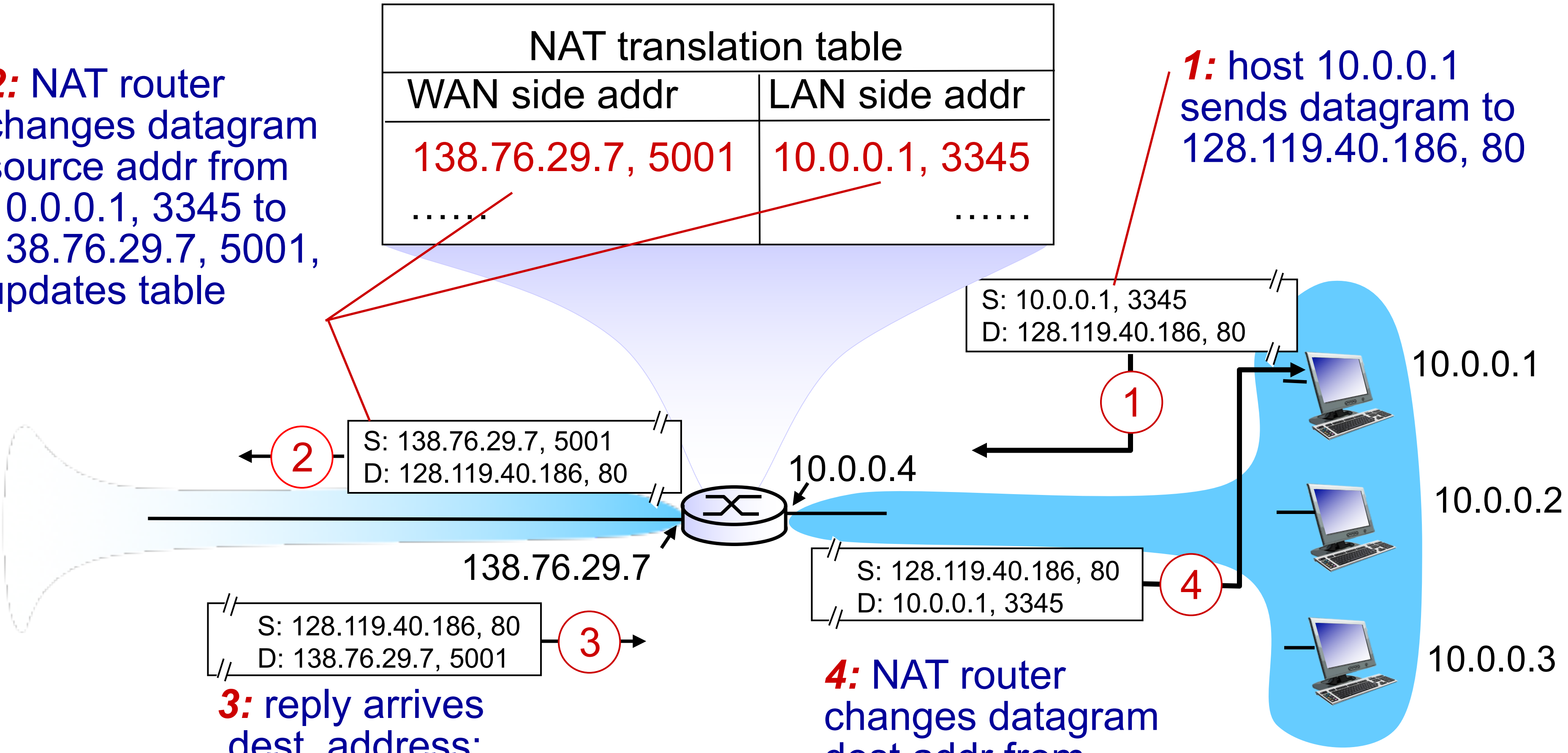
# NAT: network address translation

*2:* NAT router changes datagram source addr from 10.0.0.1, 3345 to 138.76.29.7, 5001, updates table

*1:* host 10.0.0.1 sends datagram to 128.119.40.186, 80

NAT translation table

| WAN side addr | LAN side addr |
|---|---|
| 138.76.29.7, 5001 | 10.0.0.1, 3345 |
| …… | …… |



*3:* reply arrives dest. address: 138.76.29.7, 5001

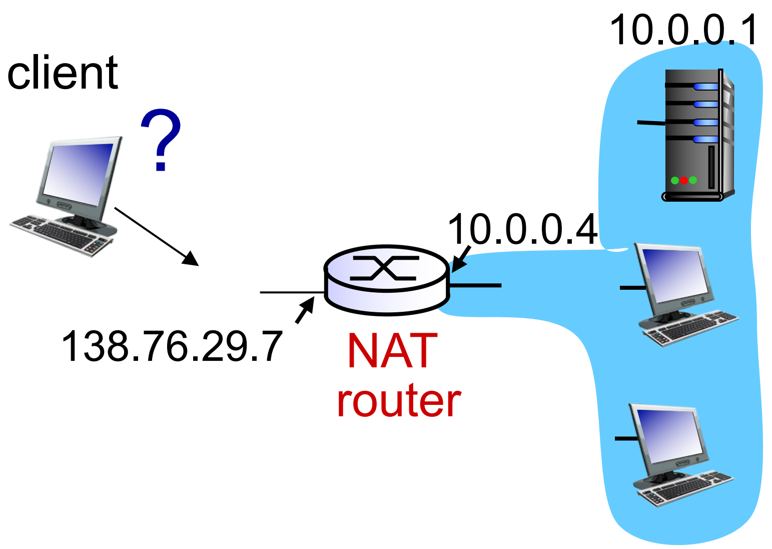
*4:* NAT router changes datagram dest addr from 138.76.29.7, 5001 to 10.0.0.1, 3345

## NAT: network address translation

- 16-bit port-number field:
  - 60,000 simultaneous connections with a single LAN-side address!
- NAT is controversial:
  - routers should only process up to layer 3
  - violates end-to-end argument
    - NAT possibility must be taken into account by app designers, e.g., P2P applications
  - address shortage should instead be solved by IPv6

# NAT traversal problem

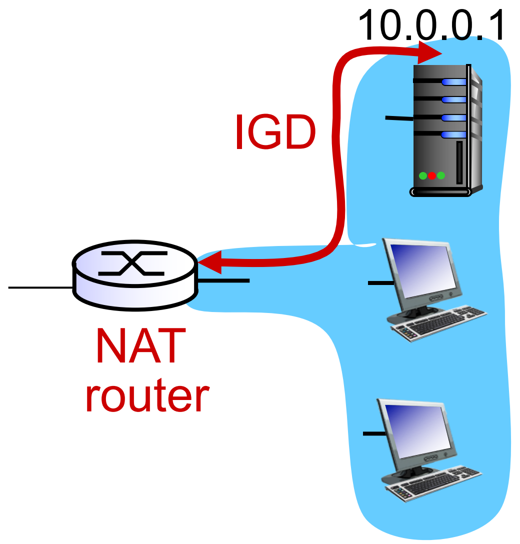
- client wants to connect to server with address 10.0.0.1
  - server address 10.0.0.1 local to LAN (client can't use it as destination addr)
  - only one externally visible NATed address: 138.76.29.7
- *solution1:* statically configure NAT to forward incoming connection requests at given port to server
  - e.g., (123.76.29.7, port 2500) always forwarded to 10.0.0.1 port 25000



# NAT traversal problem

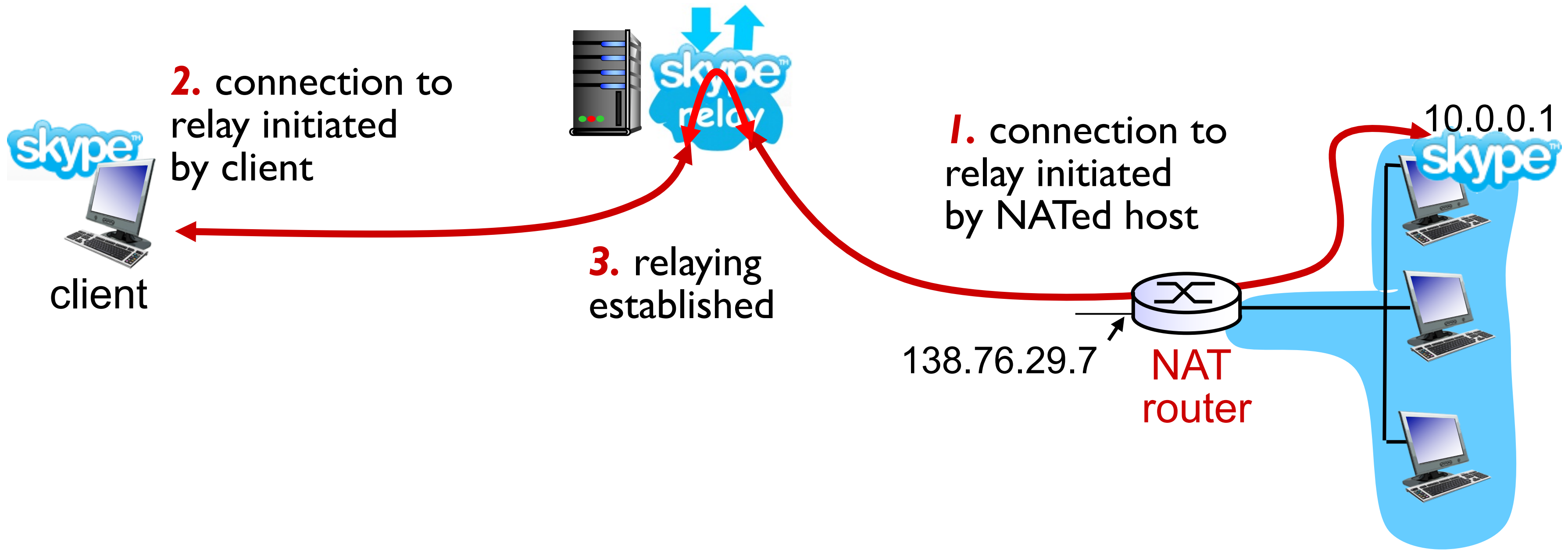
- *solution 2:* Universal Plug and Play (UPnP) Internet Gateway Device (IGD) Protocol. Allows NATed host to:
  - learn public IP address (138.76.29.7)
  - add/remove port mappings (with lease times)

i.e., automate static NAT port map configuration



# NAT traversal problem

- *solution 3:* relaying (used in Skype)
  - NATed client establishes connection to relay
  - external client connects to relay
  - relay bridges packets between to connections

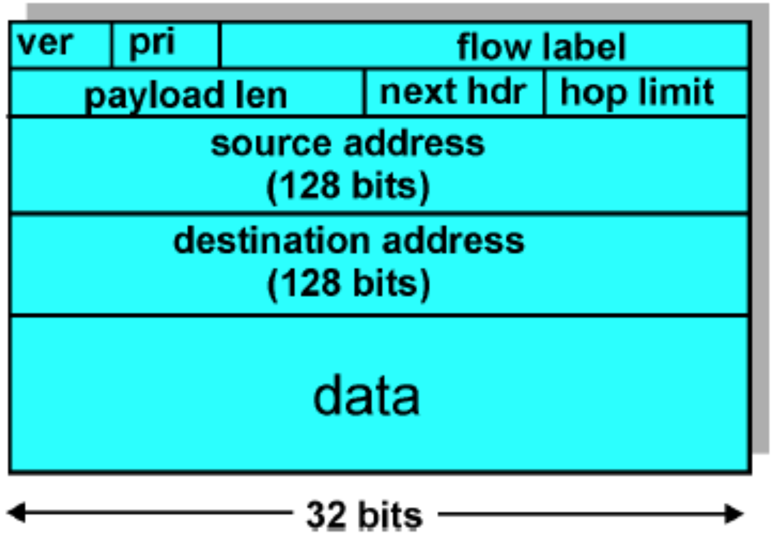


2. connection to relay initiated by client
skype relay
client
3. relaying established
1. connection to relay initiated by NATed host
10.0.0.1
138.76.29.7
NAT router

# IPv6

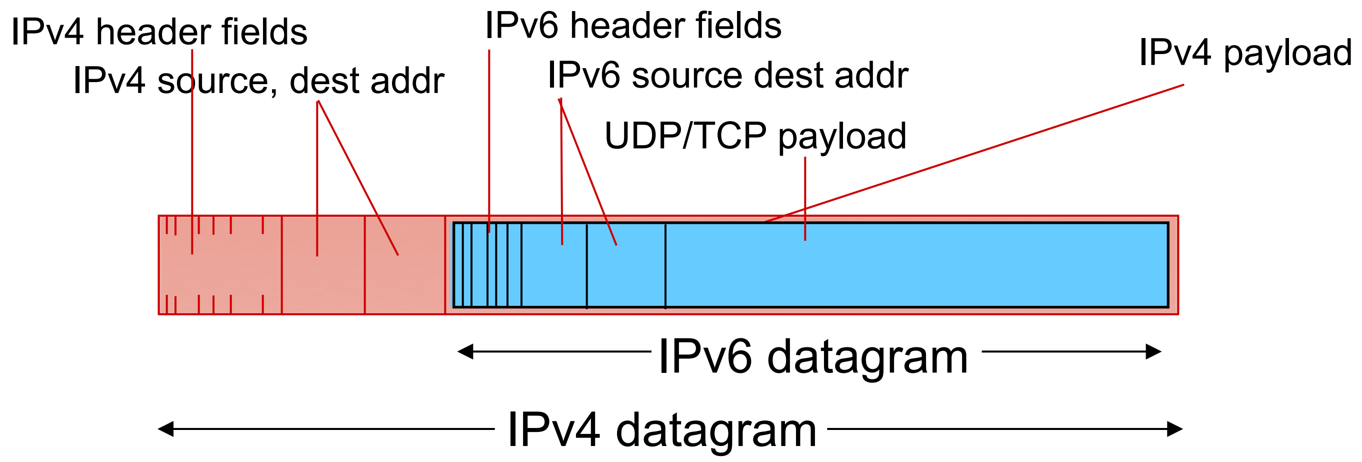
*Priority:* identify priority among datagrams in flow
*Flow Label:* identify datagrams in the same "flow."
(concept of"flow" not well defined).
*Next header:* identify upper layer protocol for data

- Initial motivation: 32-bit address space soon to be completely allocated.
- Additional motivation:
  - header format helps speed processing/forwarding
  - header changes to facilitate QoS
  - IPv6 datagram format:
  - fixed-length 40 byte header
  - no fragmentation allowed

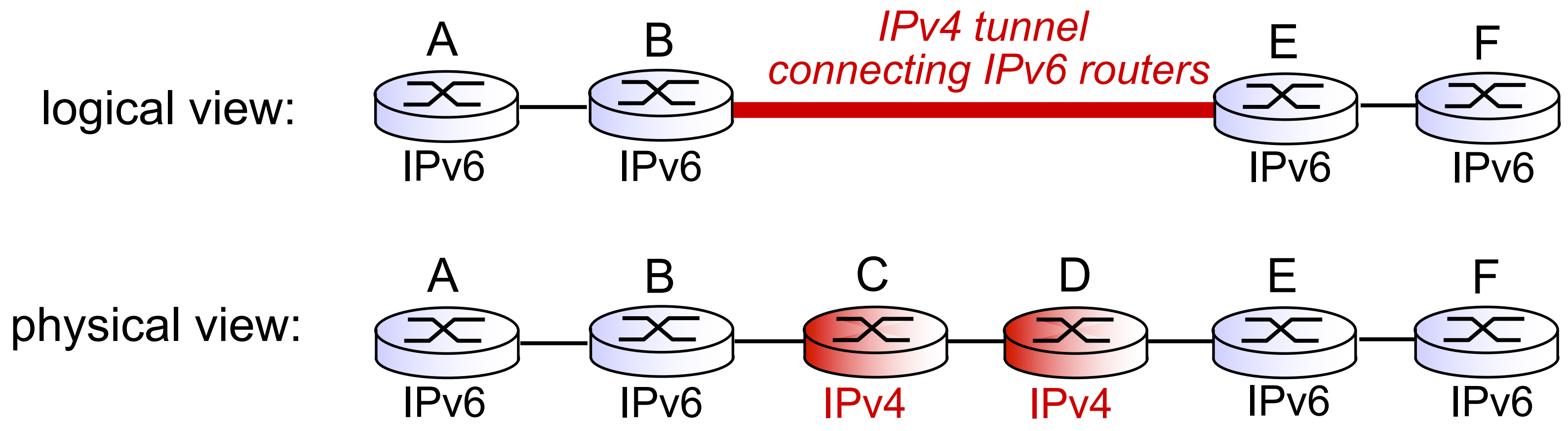


# Transition from IPv4 to IPv6

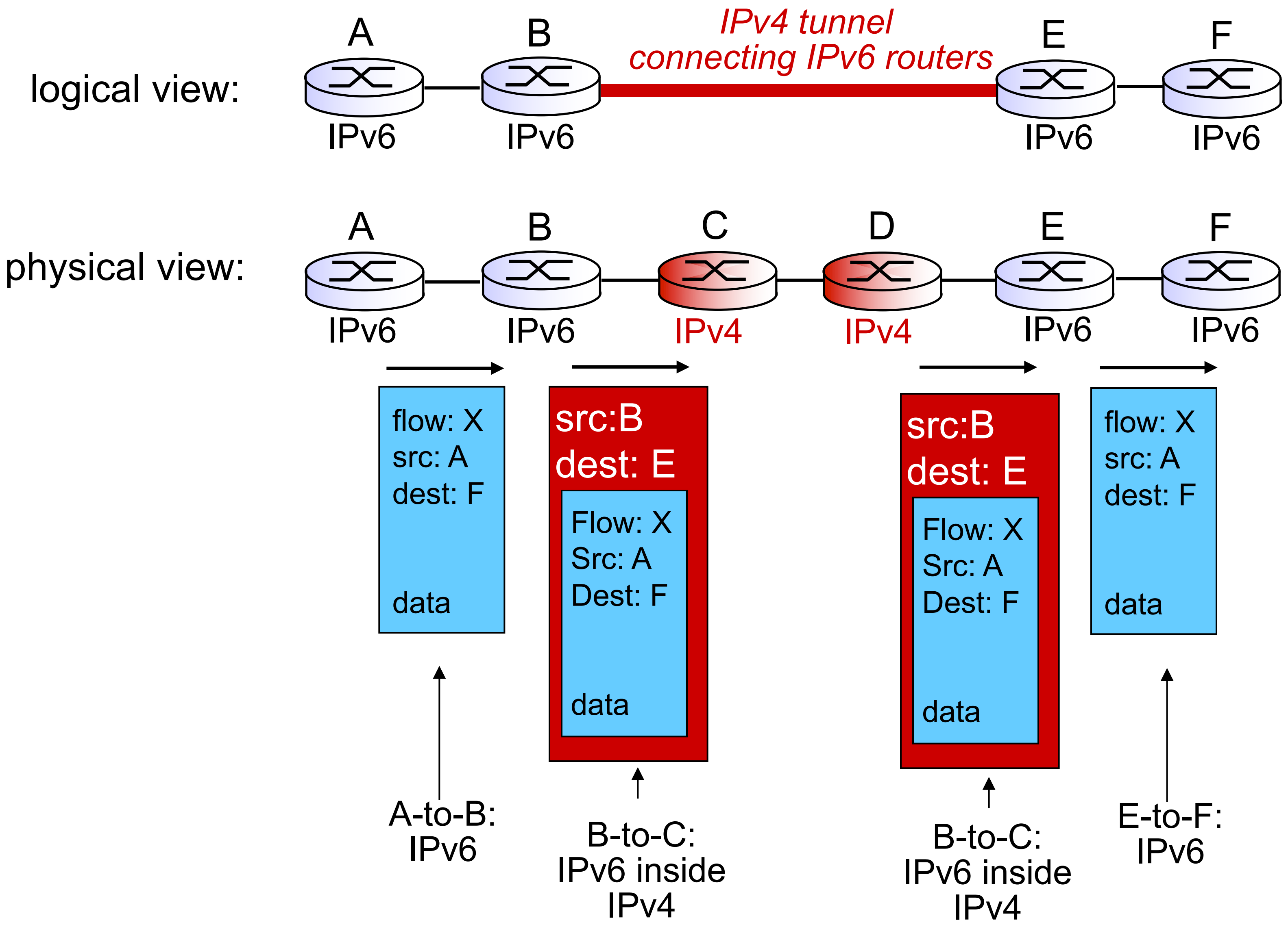
- not all routers can be upgraded simultaneously
  - no "flag days"
  - how will network operate with mixed IPv4 and IPv6 routers?
- *tunneling:* IPv6 datagram carried as *payload* in IPv4 datagram among IPv4 routers



# Tunneling


logical view:
A
B
IPv4 tunnel
connecting IPv6 routers
E
F
IPv6
IPv6
IPv6
IPv6
physical view:
A
B
C
D
E
F
IPv6
IPv6
IPv4
IPv4
IPv6
IPv6

# Tunneling



logical view:
A
B
IPv4 tunnel connecting IPv6 routers
E
F
IPv6
IPv6
IPv6
IPv6
physical view:
A
B
C
D
E
F
IPv6
IPv6
IPv4
IPv4
IPv6
IPv6
flow: X
src: A
dest: F
data
src:B
dest: E
Flow: X
Src: A
Dest: F
data
src:B
dest: E
Flow: X
Src: A
Dest: F
data
flow: X
src: A
dest: F
data
A-to-B: IPv6
B-to-C: IPv6 inside IPv4
B-to-C: IPv6 inside IPv4
E-to-F: IPv6