Redes de Computadores

The Network Layer

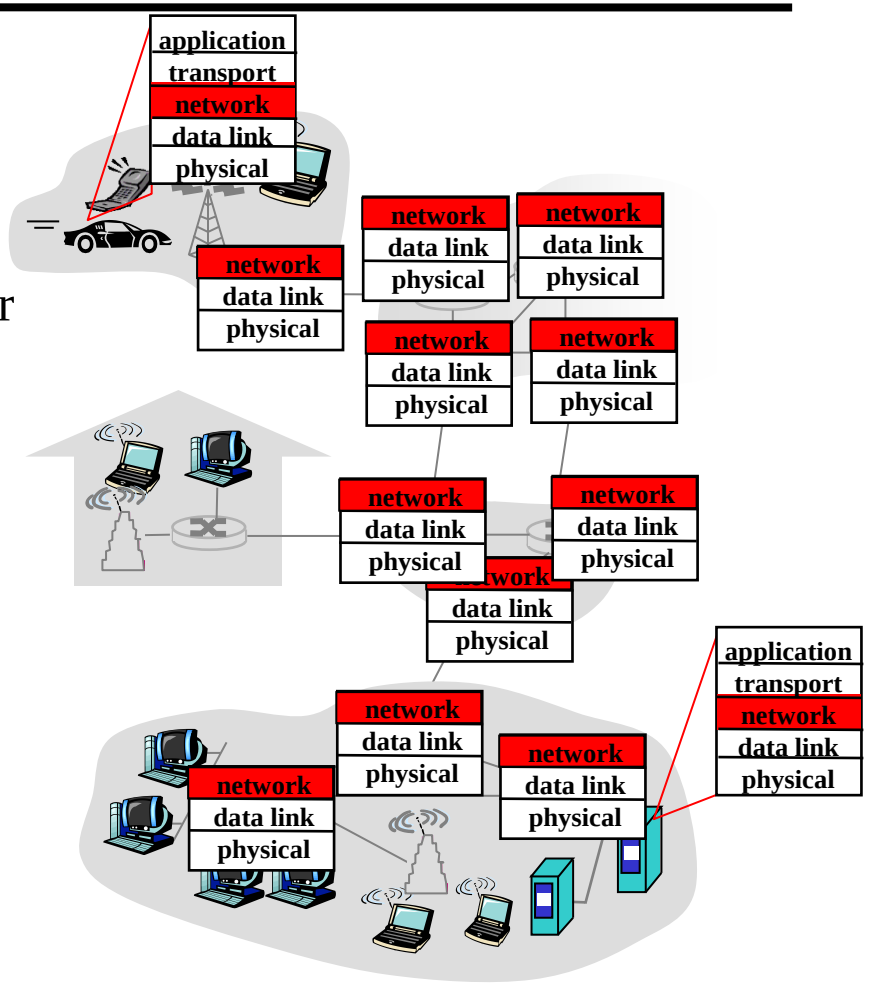
Manuel Ricardo, Rui Prior

Universidade do Porto

- What are the main functions of the network layer?
- What are the differences between virtual circuit and datagram networks?
- How is forwarding handled in both type of networks?
- What are the main functions of a router?
- What are the formats of IP addresses?
- How to form subnets?
- What services are provided by ARP, ICMP, DHCP and NAT? How do these protocols work?
- What are differences the between IPv4 and IPv6?

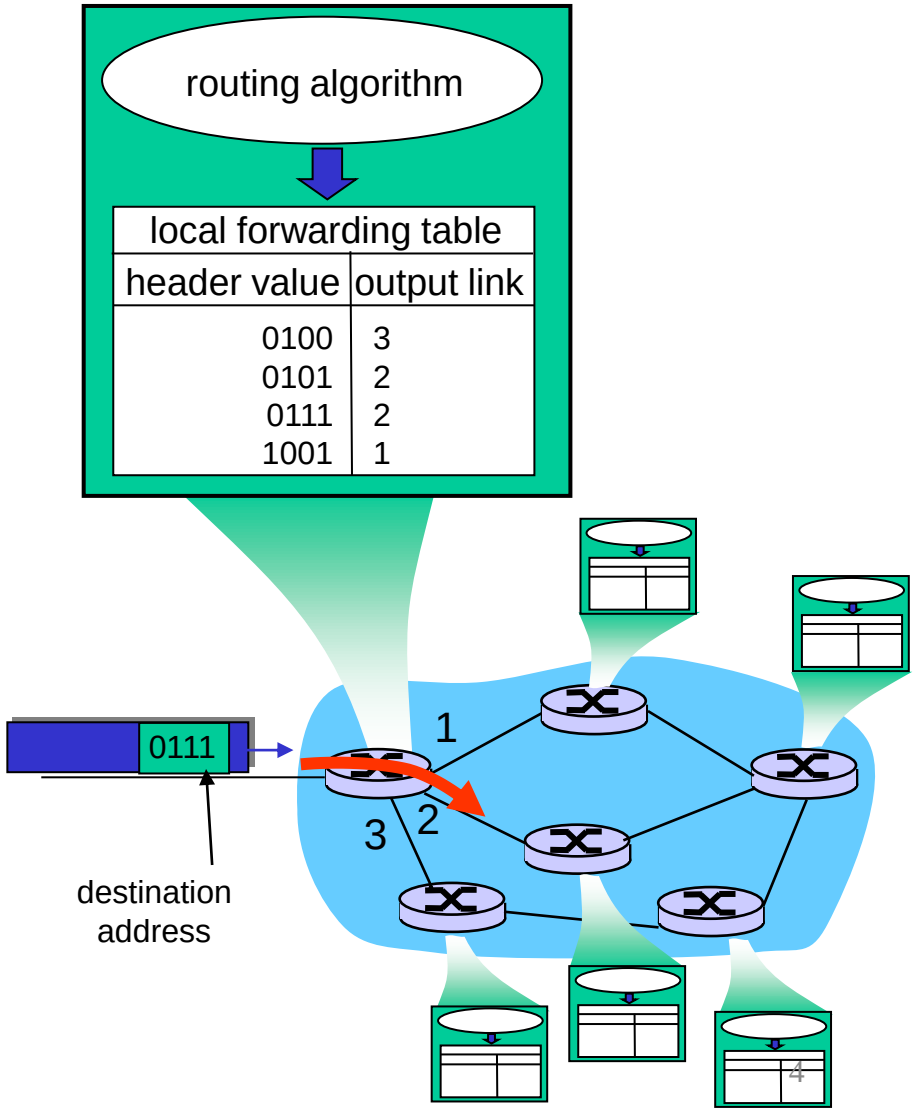
Network Layer Overview

- Network layer
  - » transports packets (datagrams)
  - » from sending host to receiving host
  - » functions located in every host and router
- Sender
  - » encapsulates transport data into packets
  - » generates packets
- Receiver
  - » receives packets
  - » delivers data to transport layer
- Router
  - » receives packets from input line
  - » examines network layer header
  - » forwards packets through adequate output link(s)



Network Layer – Main Functions

- Forwarding
  - » router forwards packet
    from input port to output port
- Routing
  - » determine route
    taken by packets,
    from source to destination
  - » algorithms, shortest path

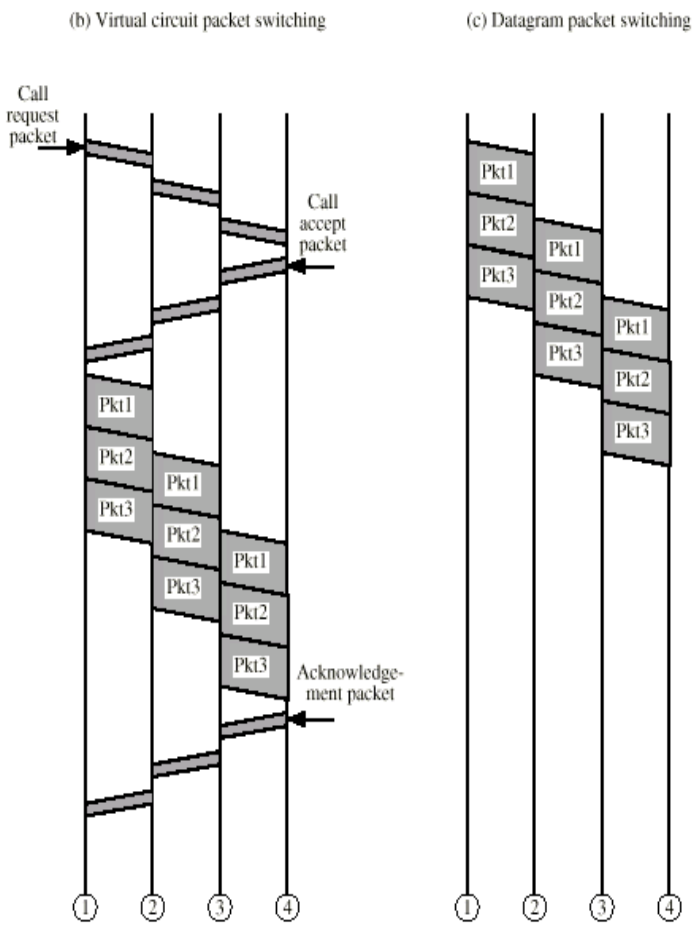


Virtual Circuits and Datagram Networks

Network Layer – ***Connection*** *and* ***Connectionless*** *Service*

Services provided by network layer

- » Datagram network
  - ➔ connectionless service
- » Virtual Circuit network
  - ➔ connection-oriented service

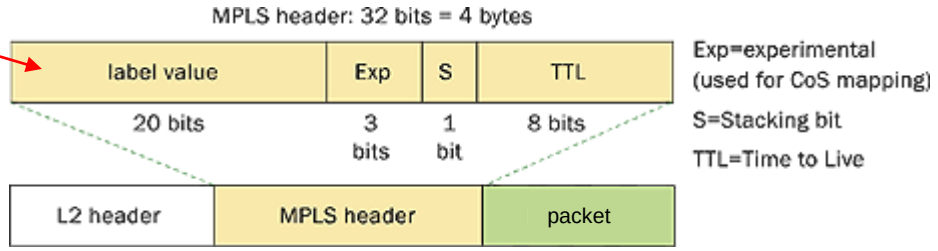


Virtual Circuit (VC)

- Phases

  circuit establishment ➔ data transfer ➔ circuit termination

- Packet carries identifier of Virtual Circuit

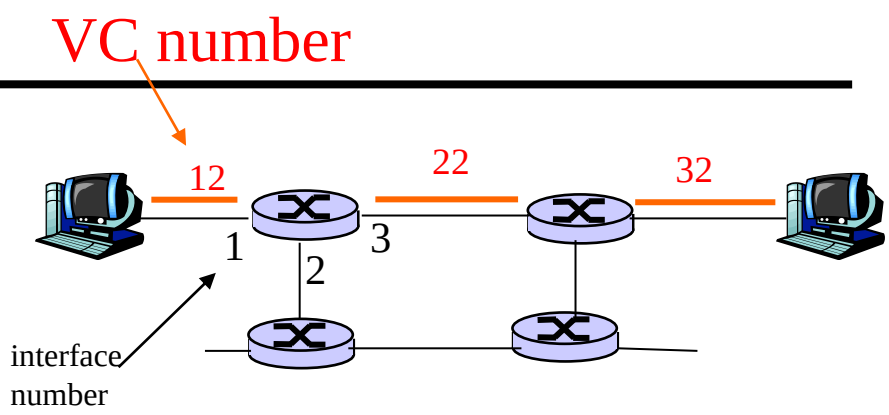


- Path defined from source to destination

  sequence of VC identifiers, one for each link along path

- Router
  - » maintains "state" for every established circuit
  - » may allocate resources (bandwidth, buffers) per Virtual Circuit

VC - Forwarding Table



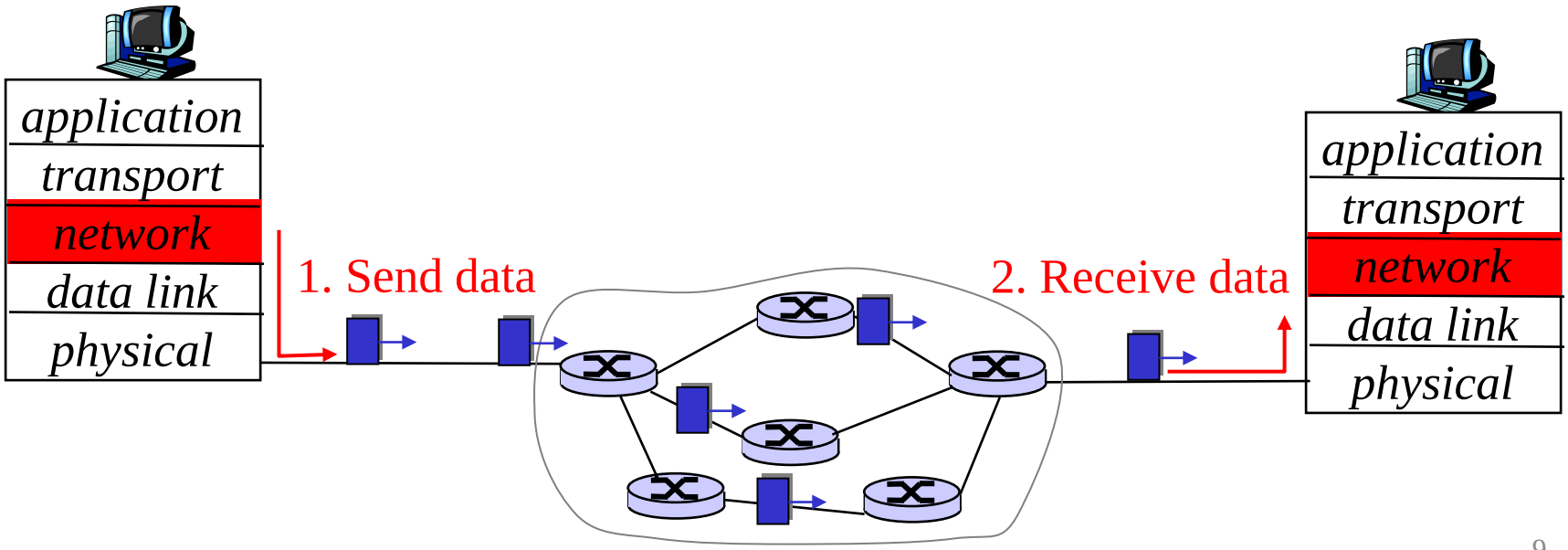
Forwarding table in northwest router:

Incoming interface	Incoming VC #	Outgoing interface	Outgoing VC #
1	12	3	22
2	63	1	18
3	7	2	17
1	97	3	87
…	…	…	…

Routers maintain connection state information!

Datagram Networks

- No circuit establishment; no circuit concept
- Packets
  - » forwarded using destination host address
  - » packets between same source-destination pair may follow different paths



To Think…

- How to reduce the number of entries in the forwarding table?

Forwarding Table

Destination Address Range	Output 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

2^{32} possible entries in IPv4

Longest Prefix Matching

Prefix Match	Link Interface
11001000 00010111 00010	0
11001000 00010111 00011000	1
11001000 00010111 00011	2
otherwise	3

Examples. Which Interface?

DA: 11001000 00010111 00010110 10100001 → 0

DA: 11001000 00010111 00011000 10101010 → 1,2 → 1

longest prefix

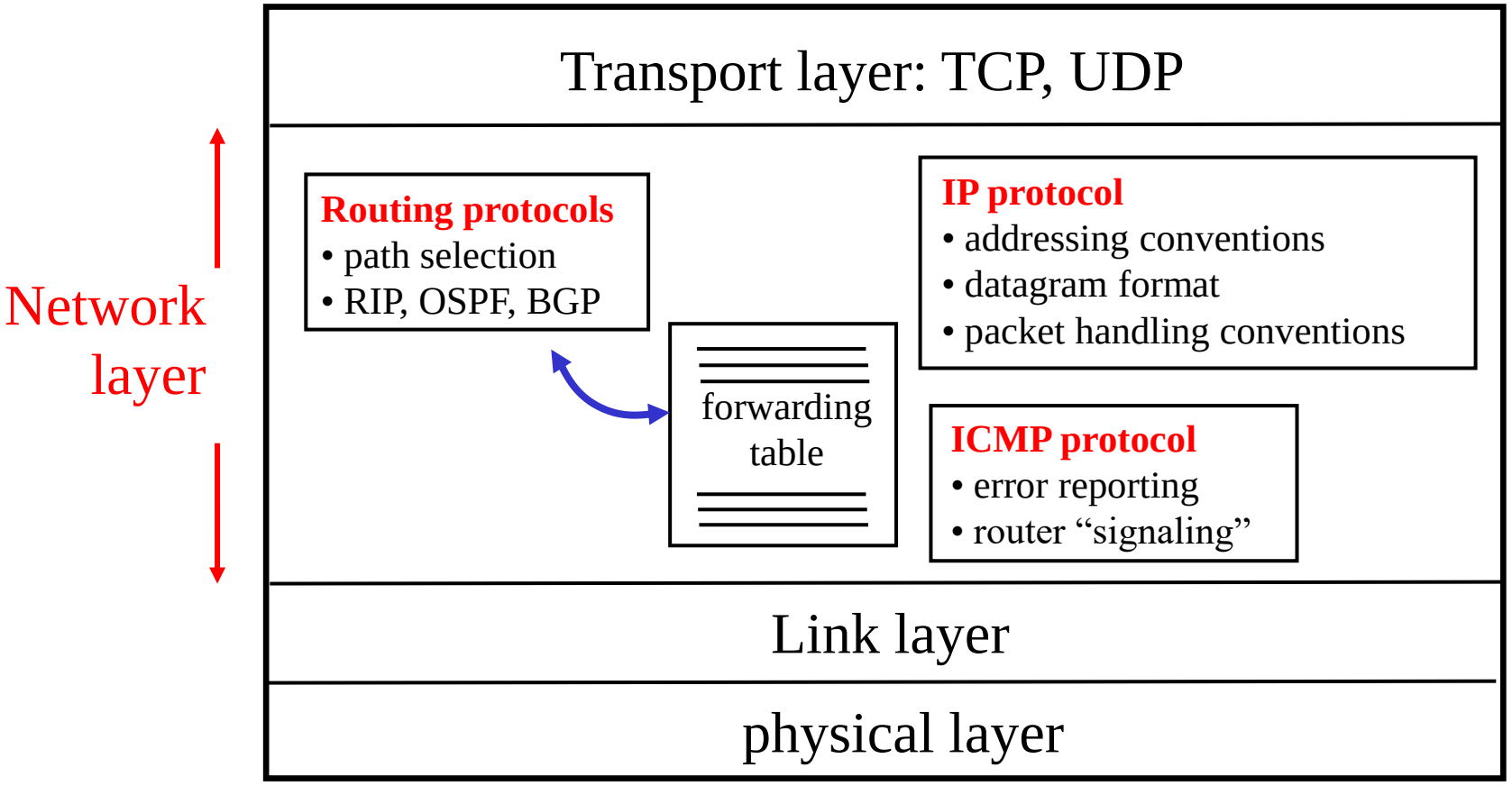
Virtual-Circuit versus Datagram Networks

Issue	Datagram subnet	Virtual-circuit subnet
Circuit setup	Not needed	Required
Addressing	Each packet contains the full source and destination address	Each packet contains a short VC number
State information	Routers do not hold state information about connections	Each VC requires router table space per connection
Routing	Each packet is routed independently; packets may arrive out of order	Route chosen when VC is set up; all packets follow it; no reordering
Effect of router failures	None, except for packets lost during the crash	All VCs that passed through the failed router are terminated
Quality of service	Difficult	Easy if enough resources can be allocated in advance for each VC
Congestion control	Difficult	Easy if enough resources can be allocated in advance for each VC

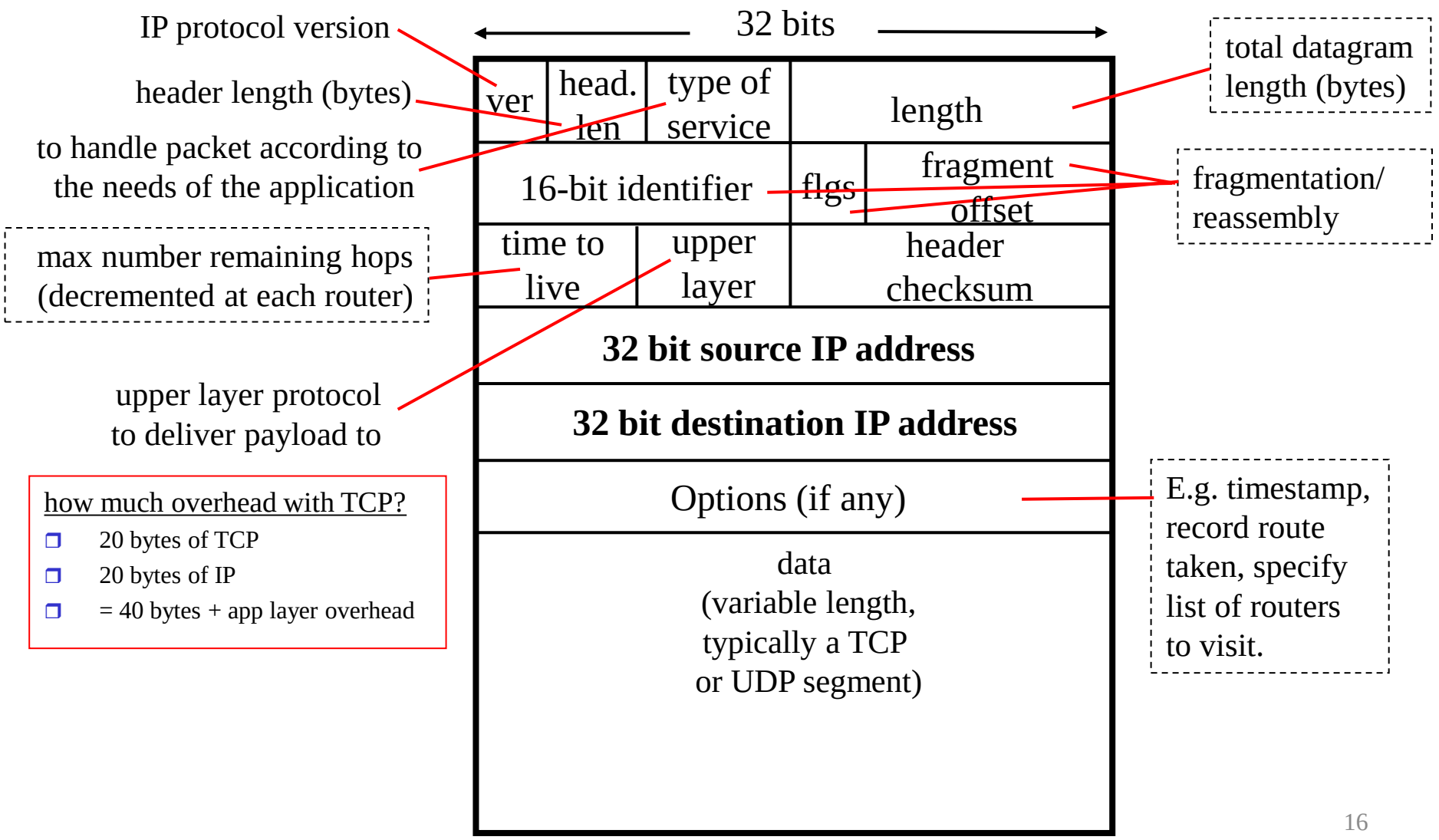
Internet Protocol

The Internet Network layer

Host, router network layer functions



IP Datagram Format

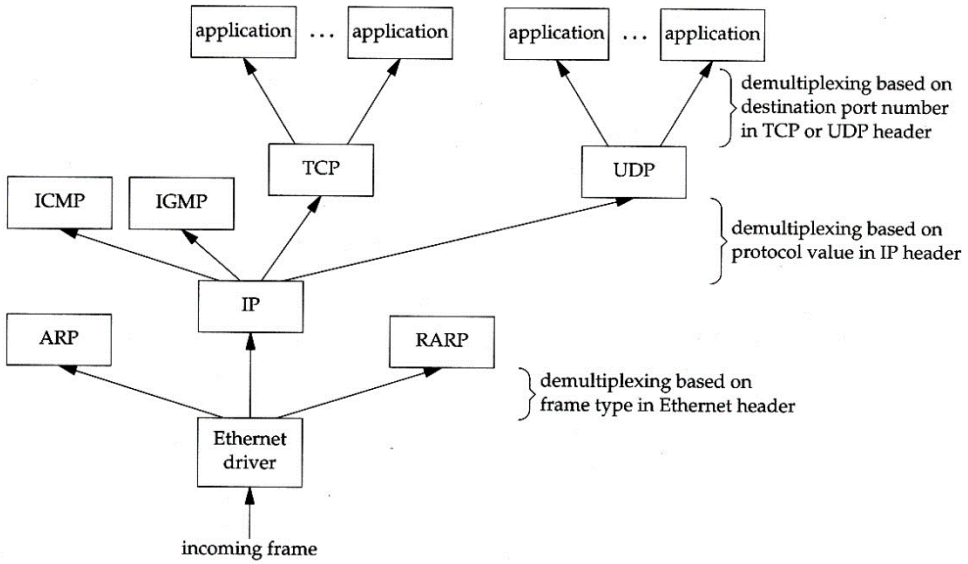


how much overhead with TCP?

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

Demultiplexing

- Ethernet header (type)
  - » IP - 0x0800
  - » ARP - 0x0806
  - » RARP - 0x8035
  - » IPX- 0x8037
  - » IPv6 - 0x86DD
  - » MPLS - 0x8847
- IP header (protocol)
  - » ICMP - 1
  - » IGMP - 2
  - » TCP - 6
  - » UDP – 17
- TCP/UDP header (port)
  - » FTP – 21
  - » Telnet - 23
  - » HTTP - 80
  - » SMTP - 25



Internet Checksum

- The Internet (not layer 2) uses a checksum
  - » easily implementable in software
  - » 1's complement sum of 16 bit words
  - » Performance: d=2

```
u_short
cksum(u_short *buf, int count)
{
    register u_long sum = 0;

    while (count--)
    {
        sum += *buf++;
        if (sum & 0xFFFF0000)
        {
            /* carry occurred,
               so wrap around */
            sum &= 0xFFFF;
            sum++;
        }
    }
    return ~(sum & 0xFFFF);
}
```

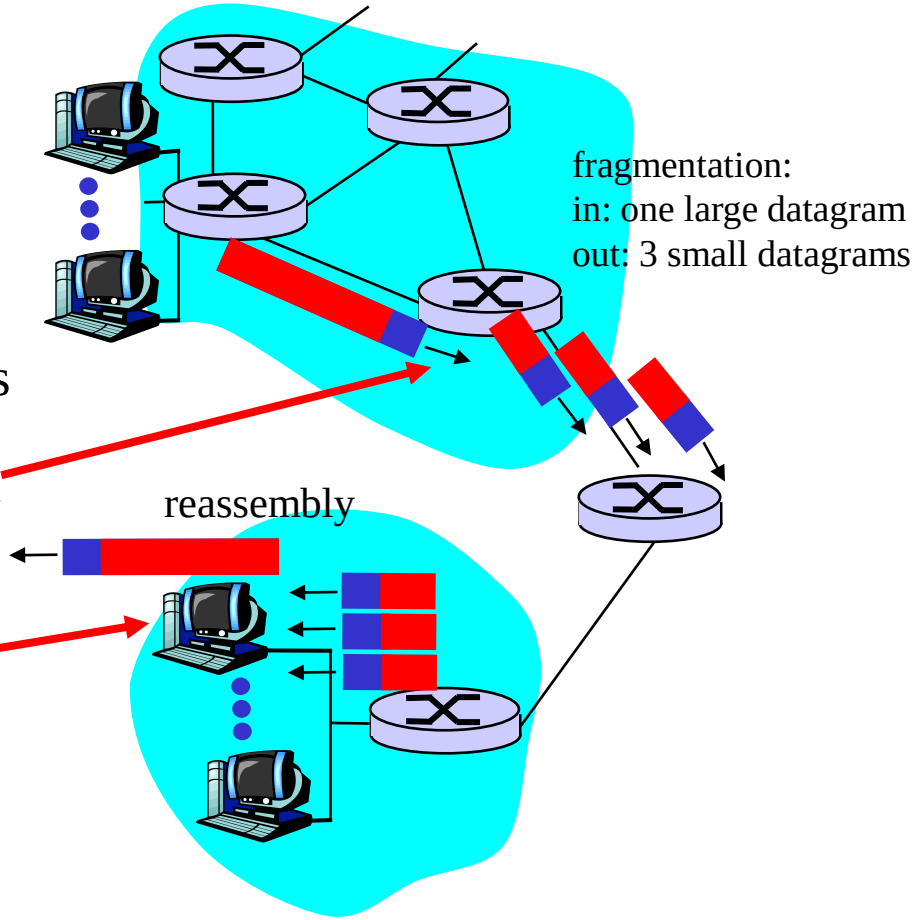
- Applied only to the header

- One's complement sum
  - » Mod-2 addition with carry-out
  - » Carry-out in the most-significant-bit is added to the least-significant bit
  - » Get one's complement of "one's complement sum"

```
                    1010011
                    0110110
carry-out       (1) 0001001
Carry wrap-around   0000001
                    0001010
One's complement = 1110101
```

IP Fragmentation and Reassembly

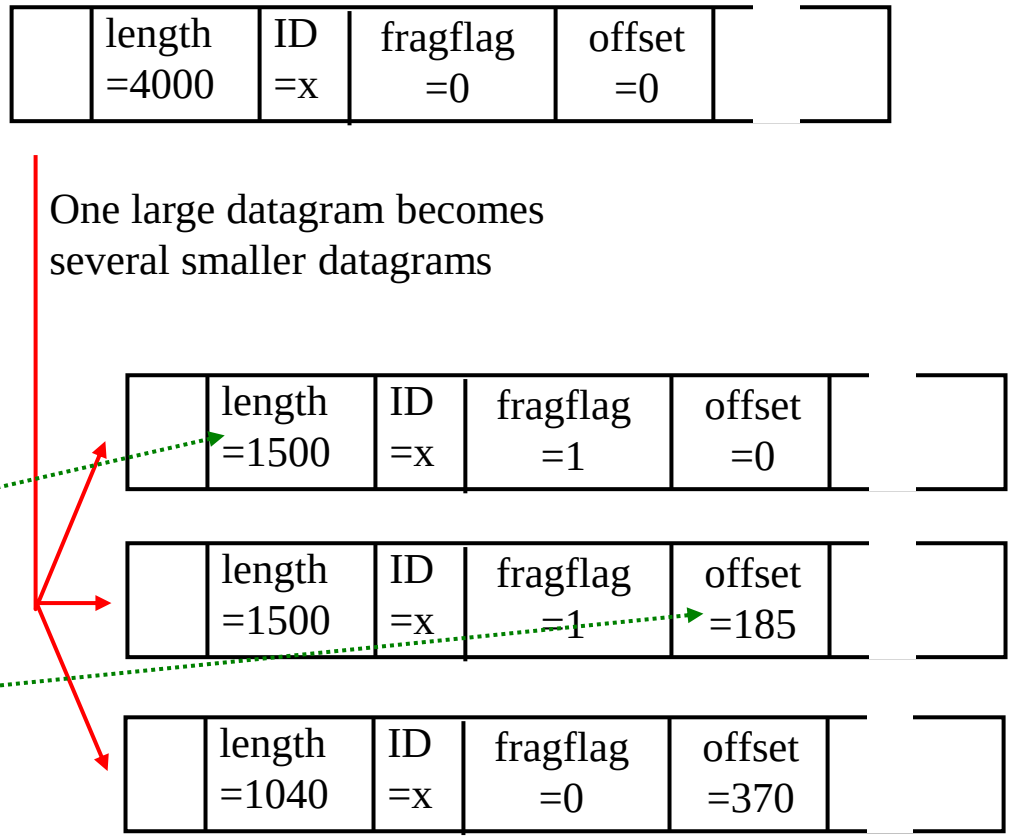
- Network links have MTU
  - » MTU - max. transfer size
  - » data field of largest possible link-level frame
  - » different link types, different MTUs
- Large IP datagram is fragmented
  - » one datagram ➔ *n* datagrams
  - » "reassembled" at final destination
  - » IP header bits used to identify and order related fragments



IP Fragmentation and Reassembly

Example

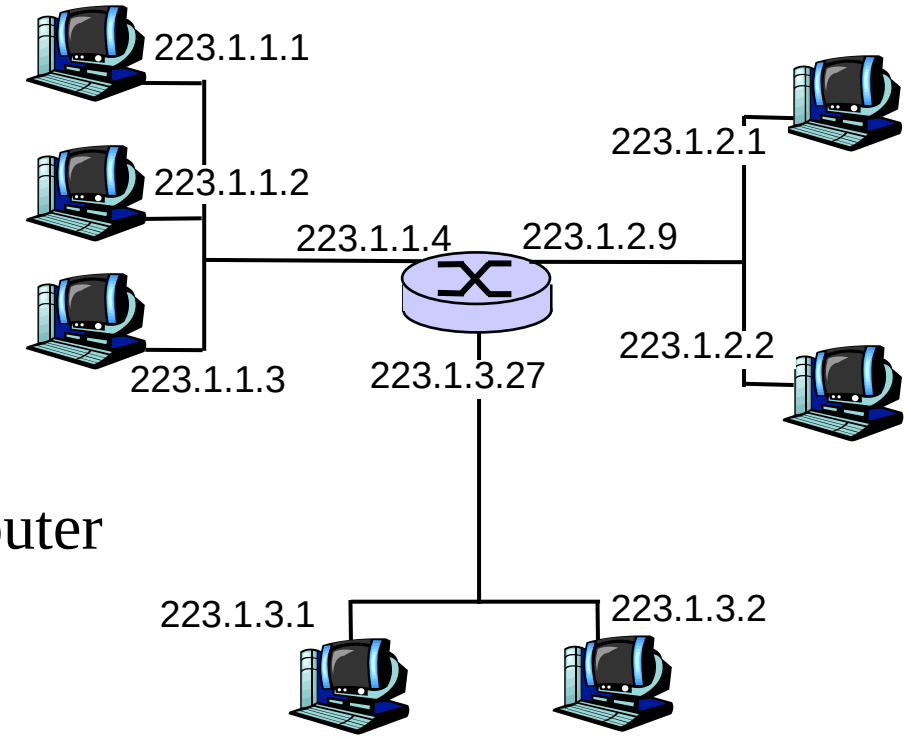
- 4000 byte datagram
- 3980 bytes data + 20 bytes IP header
- MTU = 1500 bytes



- 3 bits "stolen" from the *fragment offset* for *flags* in the IP header, assumed to be zeroes
  ⇒ data length must be a multiple of 8 in all fragments except the last one

IP Addressing - Introduction

- IP address
  - » 32-bit identifier for host/router interface
- Interface
  - » connection between host/router and physical link
  - » routers have multiple interfaces
  - » IP addresses associated with interfaces

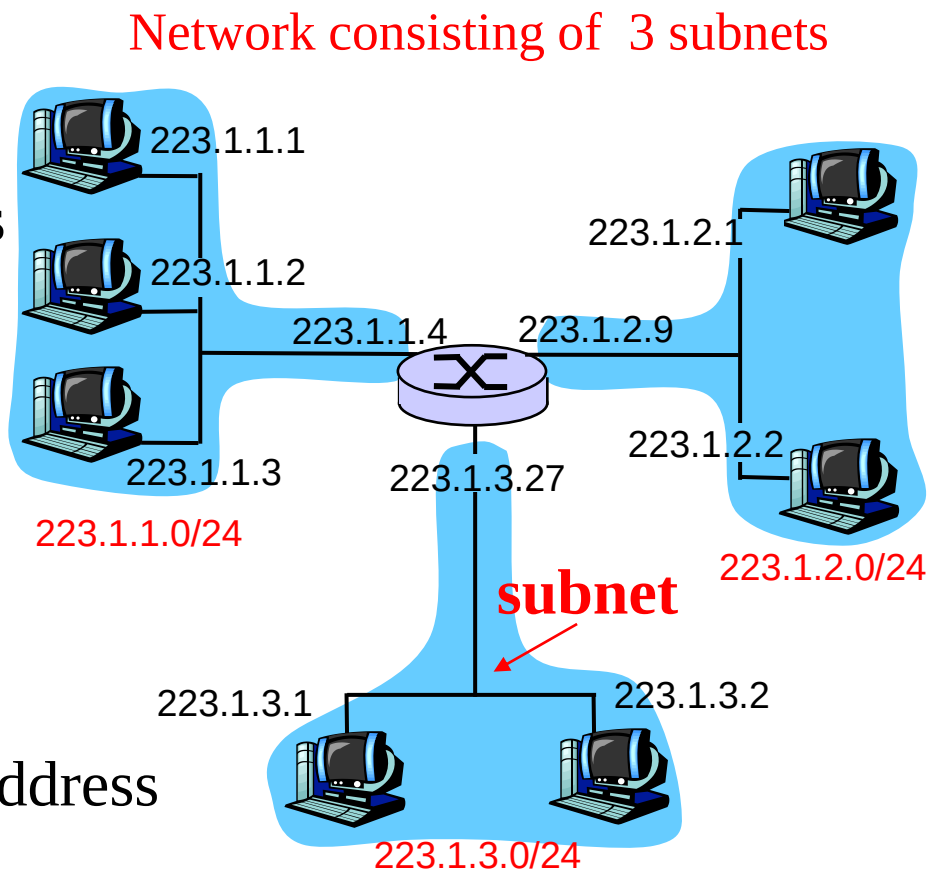


223.1.1.1 = 11011111 00000001 00000001 00000001

223 **1** **1** **1**

Subnets

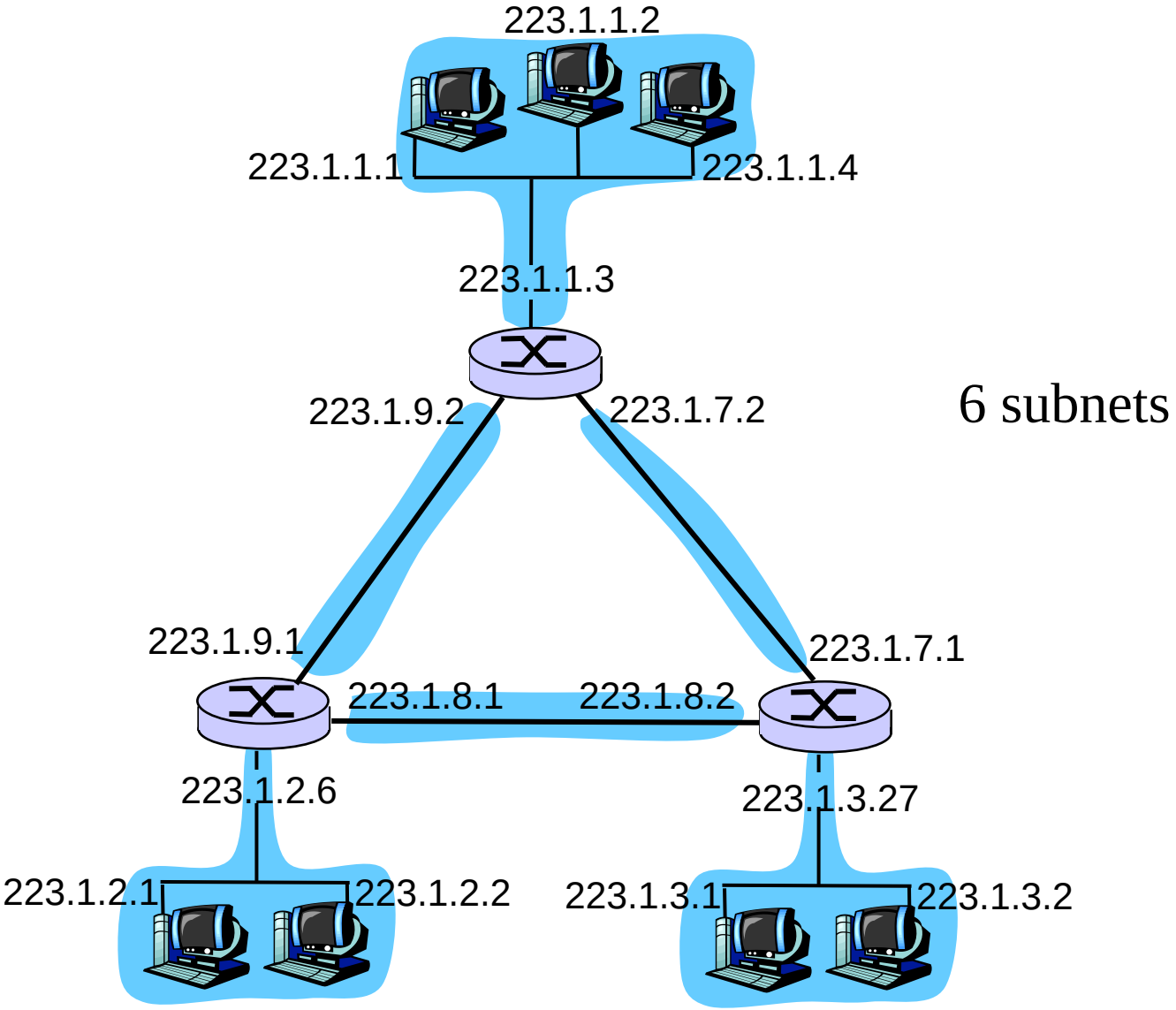
- IP address
  - » subnet part ➔ high order bits
  - » host part ➔ low order bits



- Subnet ➔ set of interfaces
  - » with same subnet part of IP address
  - » can reach each other directly

    without router intervention

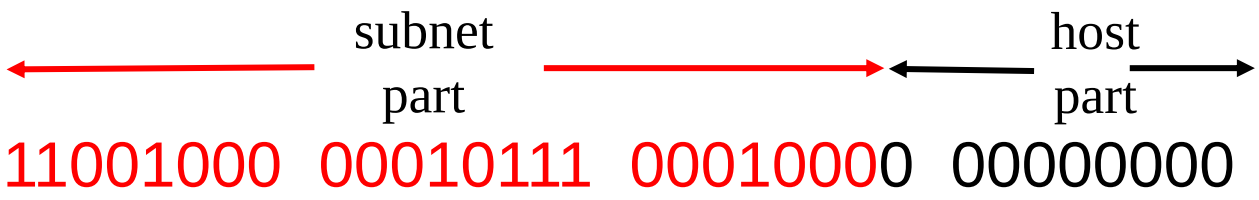
How many subnets?



IP Addressing - CIDR

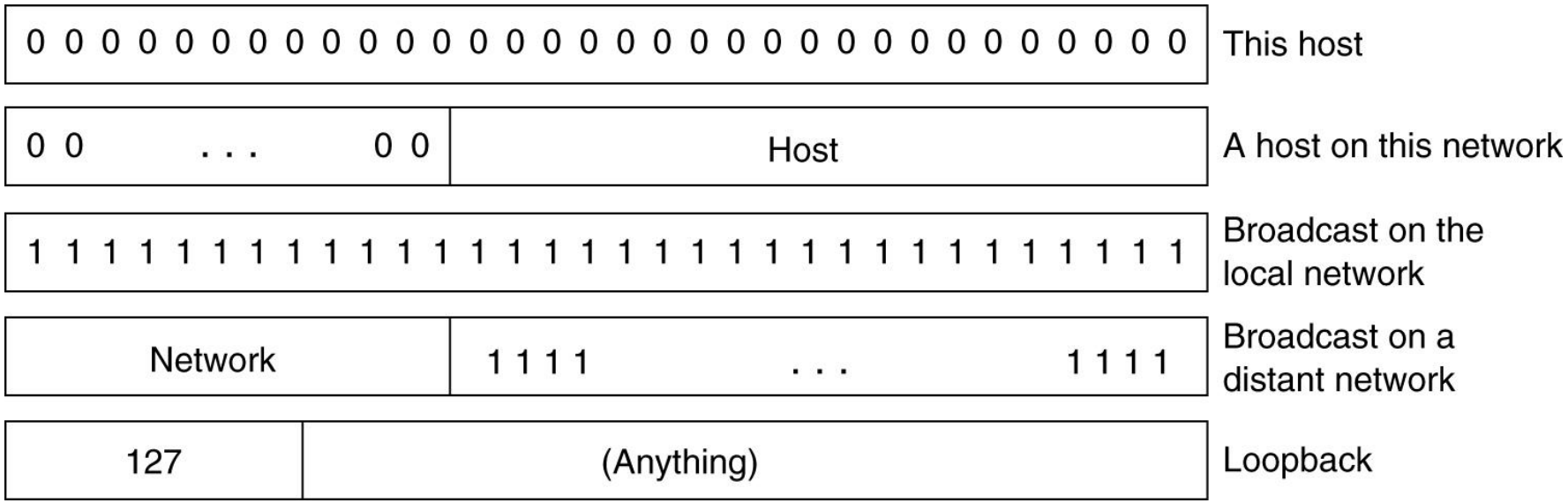
CIDR: Classless InterDomain Routing

- » subnet portion of address has arbitrary length
- » address format ➔ a.b.c.d/x
  - – where x is # bits in subnet portion of address



200.23.16.0/23

Special IP Addresses

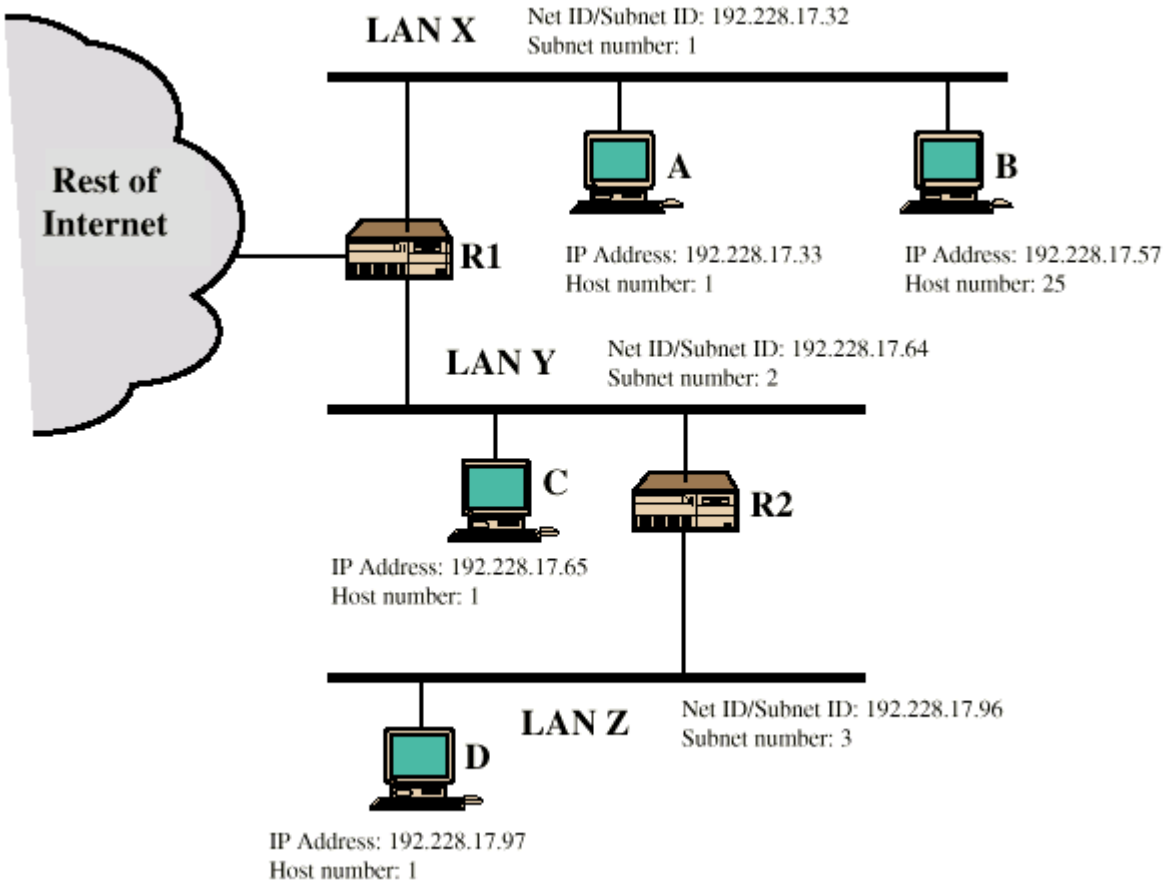


Looback Interface

- What is a loopback interface?
- Why have a loopback interface?
- What is its IP address?

Subnetting

Network **192.228.17.0/24** is divided in 8 subnetworks ➔ masks of 27 bits



Subnetwork mask – 27 bits

subnetid – 3 bits (8 subnetworks)
becomes part of the prefix

11000000 11100100 00010001 01100000

192.228.17.96/27

hostid – 5 bits

30 hosts per subnet supported
all 0 – identifies subnet
all 1 – broadcast address

Example of subnetworks

192.228.17.0/27 **(.00000000)**
192.228.17.32/27 **(.00100000)**
192.228.17.64/27 **(.01000000)**
192.228.17.96/27 **(.01100000)**
…..
192.228.17.224/27 **(.11100000)**

Subnetting

- A subnet with prefix length n can be divided into 2^{m-n} subnets of prefix length m $(m > n)$
- Example:

A /24 subnetwork

```
192.228.17.0/24 (.00000000)
```

can be divided into

2 /25 subnets

```
192.228.17.0/25   (.00000000)
192.228.17.128/25 (.10000000)
```

or

4 /26 subnets

```
192.228.17.0/26   (.00000000)
192.228.17.64/26  (.01000000)
192.228.17.128/26 (.10000000)
192.228.17.192/26 (.11000000)
```

or

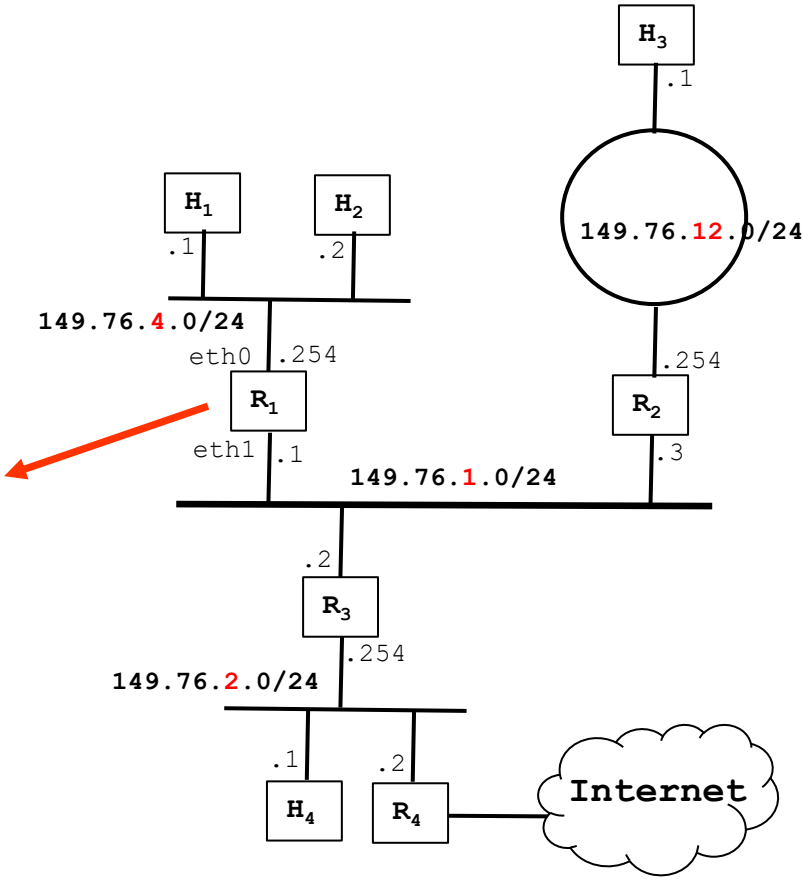
8 /27 subnets

```
192.228.17.0/26   (.00000000)
192.228.17.32/26  (.00100000)
192.228.17.64/26  (.01000000)
192.228.17.96/26  (.01100000)
192.228.17.128/26 (.10000000)
192.228.17.192/26 (.10100000)
192.228.17.192/26 (.11000000)
192.228.17.224/26 (.11100000)
```

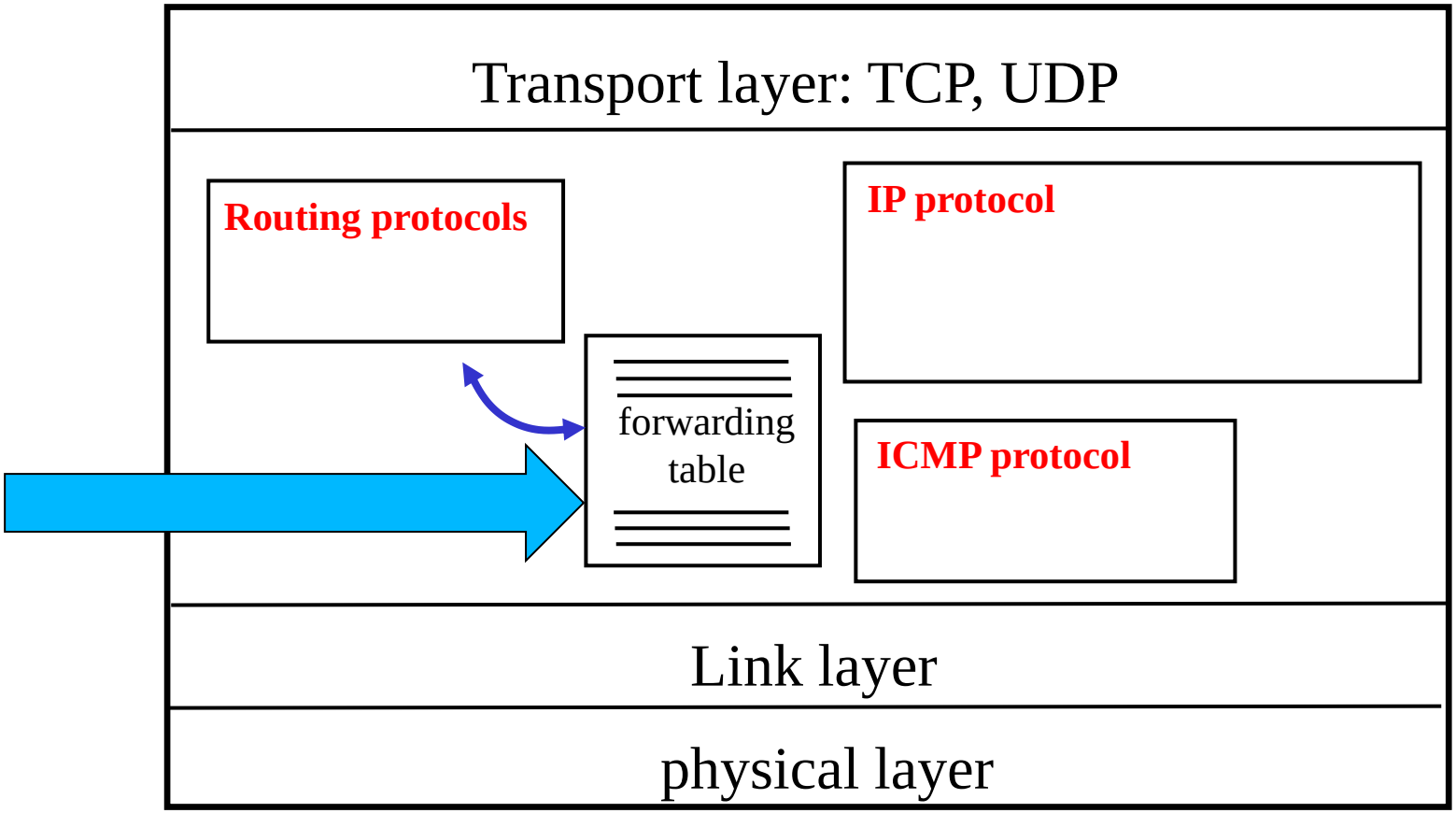
…

Forwarding Table at R1

Destination	Gateway	Interface
149.76.1.0/24	-	eth1
149.76.2.0/24	149.76.1.2	eth1
149.76.4.0/24	-	eth0
149.76.12.0/24	149.76.1.3	eth1
0/0	149.76.1.2	eth1



Forwarding Table at R1



Network mask

- Network mask: alternative representation of the prefix length
- A prefix length /n corresponds to a network mask with
  - » *n* "1" bits to the left
  - » 32-*n* "0" bits to the right
- Example: /24 ➔ 11111111.11111111.11111111.00000000

  24 ones 8 zeroes

  or, in decimal, 255.255.255.0
- A bitwise AND of an IP address with the mask gives the prefix (subnet part)

IP Forwarding Function

- Forwarding table has entries in format

  `<prefix, mask → gateway, interface>`

- Forwarding function:
  - When a datagram arrives with destination address A, then
    - For each entry of the forwarding table
      - `val = A & mask` (e.g., /8 → mask = 255.0.0.0)
      - if `( val == prefix )`
        - add entry to the set of matches
    - From the set of matches, select the one with the longest mask
      - the most specific route
    - If the selected entry has a gateway
      - send the packet to that neighbor
    - Otherwise, we have a direct connection the destination subnet
      - send the packet directly to the destination terminal

Longest Prefix Matching

Destination	Gateway	Genmask	Flags	Metric	Ref	Use	Iface
192.168.5.20	192.168.10.7	255.255.255.255	UGH	1	0	180	eth1
192.168.1.0	192.168.10.5	255.255.255.128	UG	1	0	243	eth1
192.168.10.0	0.0.0.0	255.255.255.0	U	0	0	63311	eth1
192.168.0.0	192.168.10.7	255.255.254.0	UG	1	0	2132	eth1
192.168.18.0	0.0.0.0	255.255.254.0	U	0	0	753430	eth0
192.168.64.0	192.168.10.5	255.255.192.0	UG	1	0	47543	eth1
10.0.0.0	0.0.0.0	255.0.0.0	U	0	0	3123	ppp0
127.0.0.0	0.0.0.0	255.0.0.0	U	0	0	564	lo
0.0.0.0	192.168.10.20	0.0.0.0	UG	1	0	183436	eth1

- Example: find next hop for packet with destination IP address 192.168.1.234

Destination IP address:

Decimal: `192.168.1.234`

Binary: `11000000.10101000.00000001.11101010`

Longest Prefix Matching

Destination	Gateway	Genmask	Flags	Metric	Ref	Use	Iface
192.168.5.20	192.168.10.7	255.255.255.255	UGH	1	0	180	eth1
192.168.1.0	192.168.10.5	255.255.255.128	UG	1	0	243	eth1
192.168.10.0	0.0.0.0	255.255.255.0	U	0	0	63311	eth1
192.168.0.0	192.168.10.7	255.255.254.0	UG	1	0	2132	eth1
192.168.18.0	0.0.0.0	255.255.254.0	U	0	0	753430	eth0
192.168.64.0	192.168.10.5	255.255.192.0	UG	1	0	47543	eth1
10.0.0.0	0.0.0.0	255.0.0.0	U	0	0	3123	ppp0
127.0.0.0	0.0.0.0	255.0.0.0	U	0	0	564	lo
0.0.0.0	192.168.10.20	0.0.0.0	UG	1	0	183436	eth1

Prefix: Decimal: `192.168.5.20`

Binary: `11000000.10101000.00000101.00010100`

Mask: Decimal: `255.255.255.255`

Binary: `11111111.11111111.11111111.11111111`

Longest Prefix Matching

Destination	Gateway	Genmask	Flags	Metric	Ref	Use	Iface
192.168.5.20	192.168.10.7	255.255.255.255	UGH	1	0	180	eth1
192.168.1.0	192.168.10.5	255.255.255.128	UG	1	0	243	eth1
192.168.10.0	0.0.0.0	255.255.255.0	U	0	0	63311	eth1
192.168.0.0	192.168.10.7	255.255.254.0	UG	1	0	2132	eth1
192.168.18.0	0.0.0.0	255.255.254.0	U	0	0	753430	eth0
192.168.64.0	192.168.10.5	255.255.192.0	UG	1	0	47543	eth1
10.0.0.0	0.0.0.0	255.0.0.0	U	0	0	3123	ppp0
127.0.0.0	0.0.0.0	255.0.0.0	U	0	0	564	lo
0.0.0.0	192.168.10.20	0.0.0.0	UG	1	0	183436	eth1

```
    11000000.10101000.00000001.11101010     192.168.  1.234
AND 11111111.11111111.11111111.11111111     255.255.255.255
    11000000.10101000.00000001.11101010     192.168.  1.234
```

192.168.1.234 ≠ 192.168.5.20 ➔ No match

Longest Prefix Matching

Destination	Gateway	Genmask	Flags	Metric	Ref	Use	Iface
192.168.5.20	192.168.10.7	255.255.255.255	UGH	1	0	180	eth1
192.168.1.0	192.168.10.5	255.255.255.128	UG	1	0	243	eth1
192.168.10.0	0.0.0.0	255.255.255.0	U	0	0	63311	eth1
192.168.0.0	192.168.10.7	255.255.254.0	UG	1	0	2132	eth1
192.168.18.0	0.0.0.0	255.255.254.0	U	0	0	753430	eth0
192.168.64.0	192.168.10.5	255.255.192.0	UG	1	0	47543	eth1
10.0.0.0	0.0.0.0	255.0.0.0	U	0	0	3123	ppp0
127.0.0.0	0.0.0.0	255.0.0.0	U	0	0	564	lo
0.0.0.0	192.168.10.20	0.0.0.0	UG	1	0	183436	eth1

```
     11000000.10101000.00000001.11101010     192.168.  1.234
AND  11111111.11111111.11111111.10000000     255.255.255.128
     11000000.10101000.00000001.10000000     192.168.  1.128
```

192.168.1.128 ≠ 192.168.1.0 ➔ No match

Longest Prefix Matching

Destination	Gateway	Genmask	Flags	Metric	Ref	Use	Iface
192.168.5.20	192.168.10.7	255.255.255.255	UGH	1	0	180	eth1
192.168.1.0	192.168.10.5	255.255.255.128	UG	1	0	243	eth1
192.168.10.0	0.0.0.0	255.255.255.0	U	0	0	63311	eth1
192.168.0.0	192.168.10.7	255.255.254.0	UG	1	0	2132	eth1
192.168.18.0	0.0.0.0	255.255.254.0	U	0	0	753430	eth0
192.168.64.0	192.168.10.5	255.255.192.0	UG	1	0	47543	eth1
10.0.0.0	0.0.0.0	255.0.0.0	U	0	0	3123	ppp0
127.0.0.0	0.0.0.0	255.0.0.0	U	0	0	564	lo
0.0.0.0	192.168.10.20	0.0.0.0	UG	1	0	183436	eth1

```
     11000000.10101000.00000001.11101010      192.168.  1.234
AND  11111111.11111111.11111111.00000000      255.255.255.  0
     11000000.10101000.00000001.00000000      192.168.  1.  0
```

192.168.1.0 ≠ 192.168.10.0 ➔ No match

Longest Prefix Matching

Destination	Gateway	Genmask	Flags	Metric	Ref	Use	Iface
192.168.5.20	192.168.10.7	255.255.255.255	UGH	1	0	180	eth1
192.168.1.0	192.168.10.5	255.255.255.128	UG	1	0	243	eth1
192.168.10.0	0.0.0.0	255.255.255.0	U	0	0	63311	eth1
192.168.0.0	192.168.10.7	255.255.254.0	UG	1	0	2132	eth1
192.168.18.0	0.0.0.0	255.255.254.0	U	0	0	753430	eth0
192.168.64.0	192.168.10.5	255.255.192.0	UG	1	0	47543	eth1
10.0.0.0	0.0.0.0	255.0.0.0	U	0	0	3123	ppp0
127.0.0.0	0.0.0.0	255.0.0.0	U	0	0	564	lo
0.0.0.0	192.168.10.20	0.0.0.0	UG	1	0	183436	eth1

```
    11000000.10101000.00000001.11101010      192.168.  1.234
AND 11111111.11111111.11111110.00000000      255.255.254.  0
    11000000.10101000.00000000.00000000      192.168.  0.  0
```

192.168.0.0 = 192.168.0.0 ➔ Match!

Since entries are sorted by decreasing prefix length,
the first match is the longest prefix match

Longest Prefix Matching

Destination	Gateway	Genmask	Flags	Metric	Ref	Use	Iface
192.168.5.20	192.168.10.7	255.255.255.255	UGH	1	0	180	eth1
192.168.1.0	192.168.10.5	255.255.255.128	UG	1	0	243	eth1
192.168.10.0	0.0.0.0	255.255.255.0	U	0	0	63311	eth1
192.168.0.0	192.168.10.7	255.255.254.0	UG	1	0	2132	eth1
192.168.18.0	0.0.0.0	255.255.254.0	U	0	0	753430	eth0
192.168.64.0	192.168.10.5	255.255.192.0	UG	1	0	47543	eth1
10.0.0.0	0.0.0.0	255.0.0.0	U	0	0	3123	ppp0
127.0.0.0	0.0.0.0	255.0.0.0	U	0	0	564	lo
0.0.0.0	192.168.10.20	0.0.0.0	UG	1	0	183436	eth1

- The next hop for a packet destined to 192.168.1.234 is 192.168.10.7 ("gateway" column)
- Absence of a "gateway" (0.0.0.0) means a direct connection to the destination subnet
  - » In that case, the next hop would be the destination (192.168.1.234)
  - » No flag `G` flag => no gateway

Which IP addresses match a given prefix?

- The lowest address has zeroes in all host bits
  - » Equal to the prefix
- The highest one has ones in all host bits
  - » Directed broadcast address — send a packet to all hosts at once
  - » To obtain this address, find the ones complement of the mask and bitwise OR it with the prefix. Example: 192.168.8.0/22

```
Mask:          11111111.11111111.11111100.00000000   255.255.252.  0
Inv. mask:     00000000.00000000.00000011.11111111     0.  0.  3.255
Prefix:        11000000.10101000.00001000.00000000   192.168.  8.  0
I.M. OR pref.:11000000.10101000.00001011.11111111   192.168. 11.255
```

  - » Or, in decimal, subtract the mask from 255.255.255.255 and add it to the prefix
    - – 255.255.255.255 - 255.255.252.0 = 0.0.3.255
    - – 192.168.8.0 + 0.0.3.255 = **192.168.11.255**

Address Resolution Protocol

ARP – Address Resolution Protocol

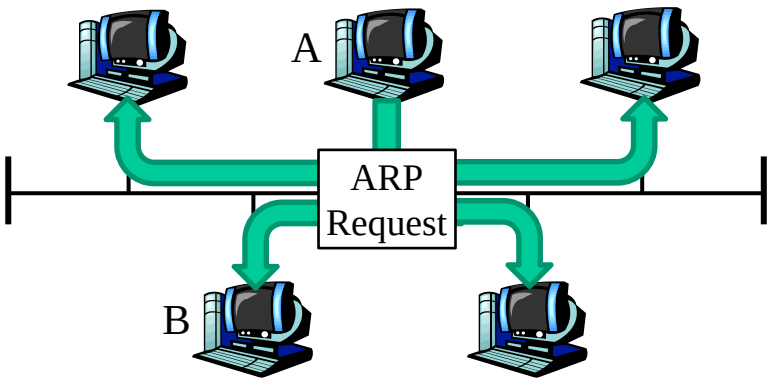
- A network interface has
  - » one MAC address
  - » one (or more) IP address(es)
- After discovering the next hop, a node must encapsulate the packet in a frame
  - » needs the next hop's MAC address to build the frame
- ARP: Address Resolution Protocol
  - » protocol used to

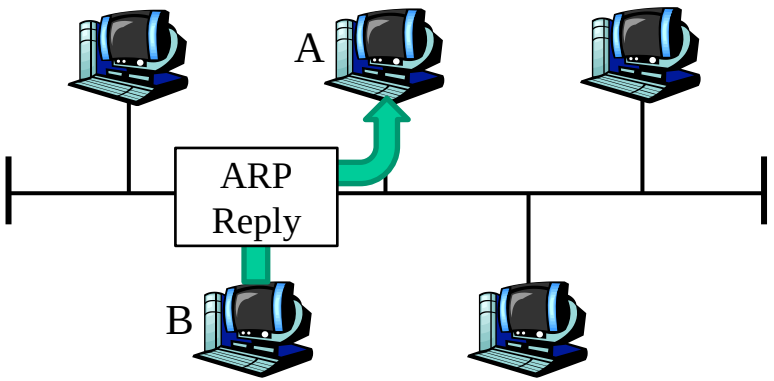
    obtain the MAC address

    associated to a given IP address
  - » neighbor IP ➔ MAC address translation stored temporarily in ARP cache

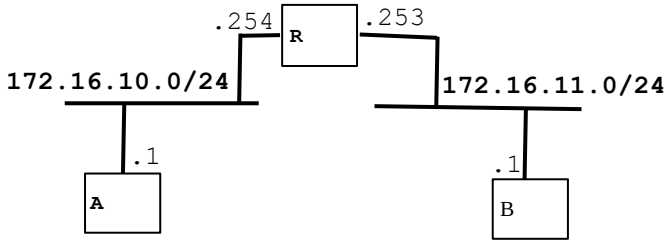
ARP in Action

- Node A wants to send a packet to neighbor B, but its IP ➔ MAC mapping is not found in the ARP cache
- Node A broadcasts an ARP Request:
  - » My IP is 192.168.1.12 and my MAC is 5A-6C-25-D5-41-80. Who has IP 192.168.1.38?
- Node B unicasts an ARP Reply:
  - » Hi, I have IP 192.168.1.38 and my MAC is B0-91-9B-51-42-55.
- Node A stores the mapping in the ARP cache and encapsulates the packet in a frame with Destination MAC address B0-91-9B-51-42-55





To Think…



- Assume host A sends an IP packet to host B and that this packet is forwarded by router R. What are the MAC and IP addresses (source and destination) observed?

- What roles does ARP play in this scenario?

Obtaining IP Addresses

How to Obtain IP Addresses

How does network get subnet part of IP addresss?

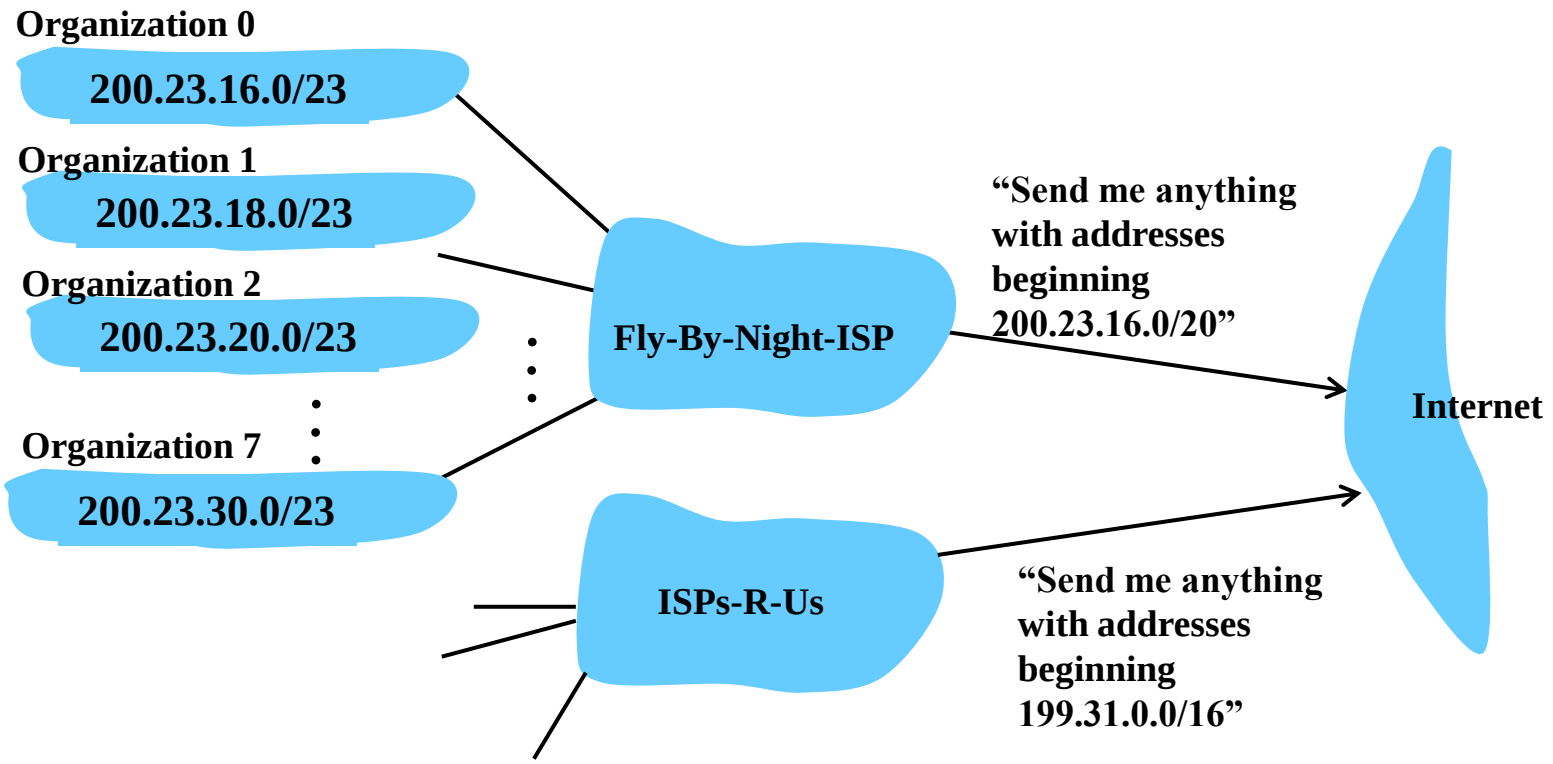
» Gets allocated portion of its provider ISP's address space

ISP's block	11001000 00010111 00010000	00000000	200.23.16.0/20
Organization 0	11001000 00010111 0001000**0**	00000000	200.23.16.0/23
Organization 1	11001000 00010111 0001001**0**	00000000	200.23.18.0/23
Organization 2	11001000 00010111 0001010**0**	00000000	200.23.20.0/23
...	.....	....	....
Organization 7	11001000 00010111 0001111**0**	00000000	200.23.30.0/23

Hierarchical Addressing - Route Aggregation

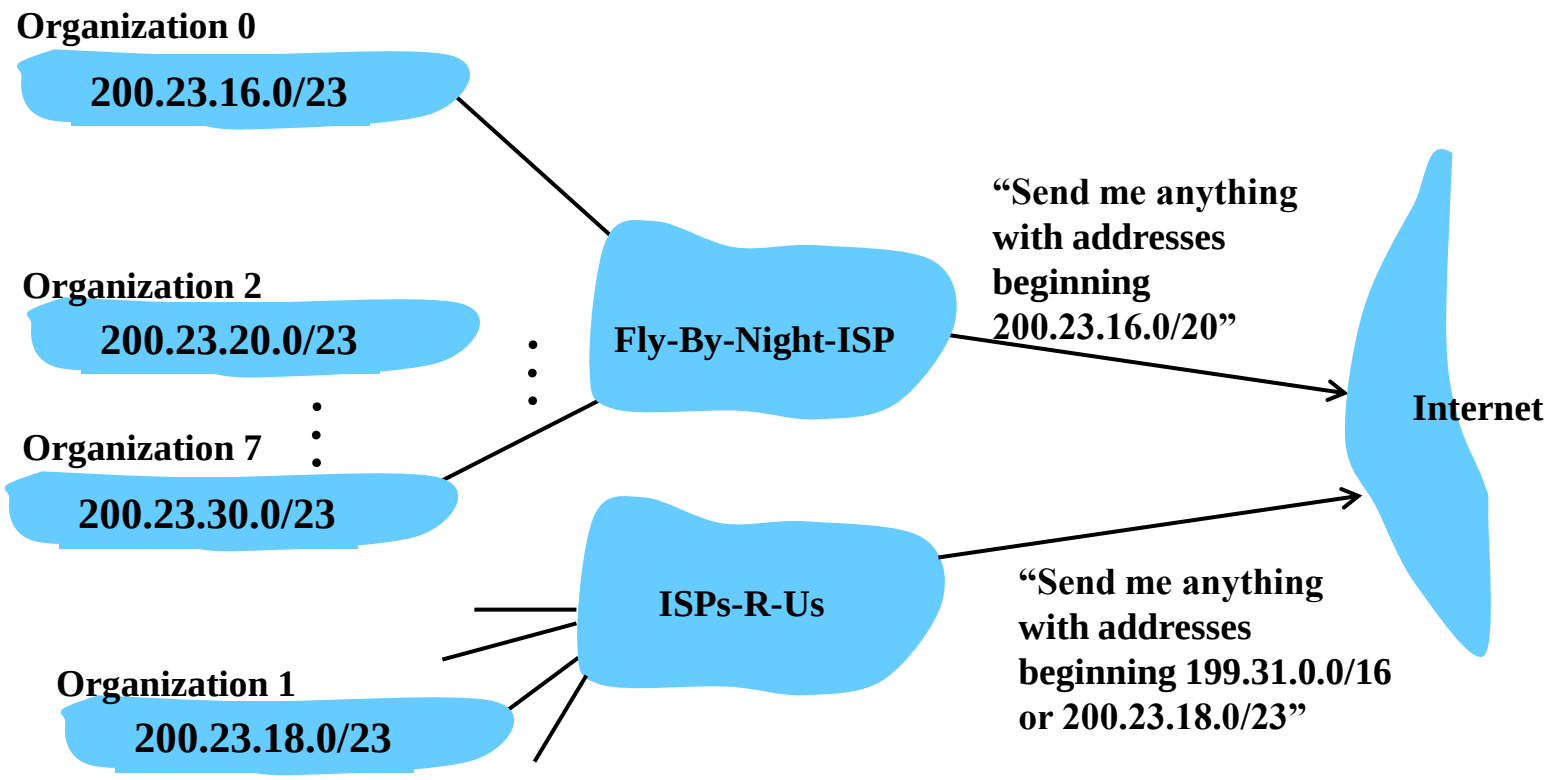
- Hierarchical addressing

  allows efficient advertisement of routing information



Hierarchical Addressing - More Specific Routes

ISPs-R-Us has a more specific route to Organization 1



IP Addressing

How does an ISP get block of addresses?

- From ICANN: Internet Corporation for Assigned Names and Numbers
- ICANN
  - allocates addresses
  - manages Domain Name Service (DNS)
  - assigns domain names, resolves disputes

IP Addresses

How does a host obtain an IP address?

- Hand configured by the system administrator
  - Windows:
    - old way: control panel → network → configuration → tcp/ip → properties
    - new way: settings → network & internet → *interface* → IP assignment
  - UNIX/Linux:
    - /etc/sysconfig/newtork-scripts/ifcfg-*itf*
    - /etc/network/interfaces
    - command line tools: ifconfig, ip addr

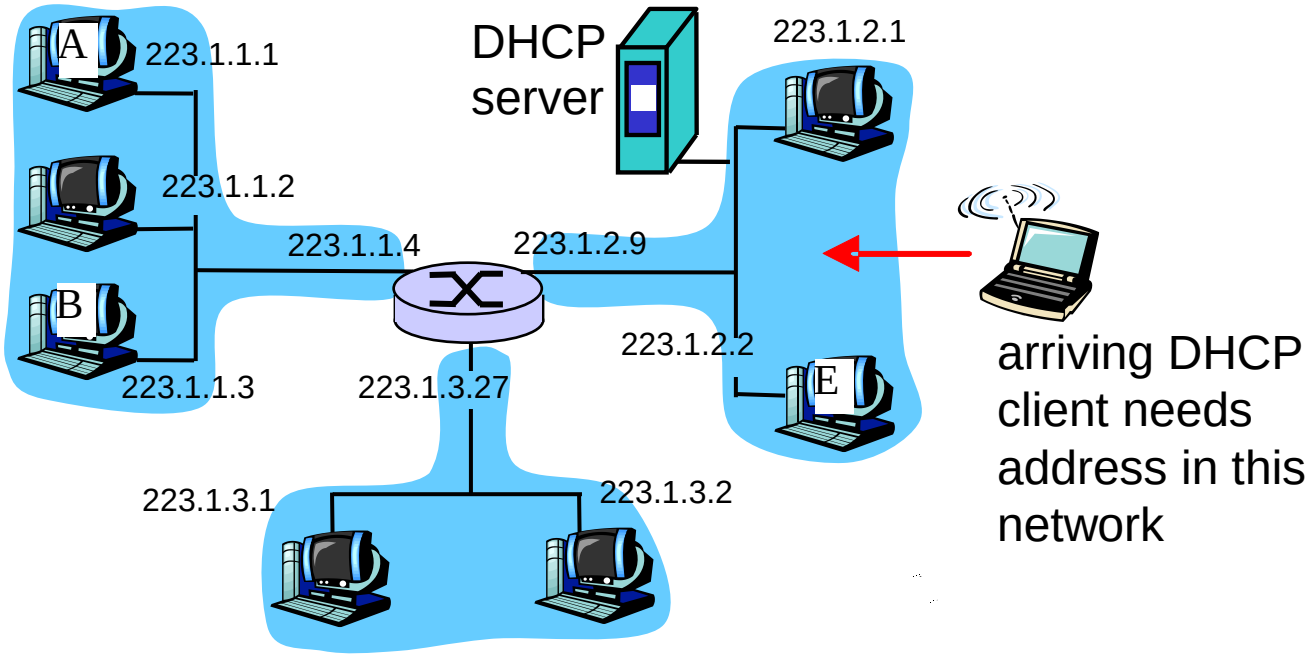
- DHCP: Dynamic Host Configuration Protocol
  - dynamically get address from a server
  - “plug-and-play”

DHCP - Dynamic Host Configuration Protocol

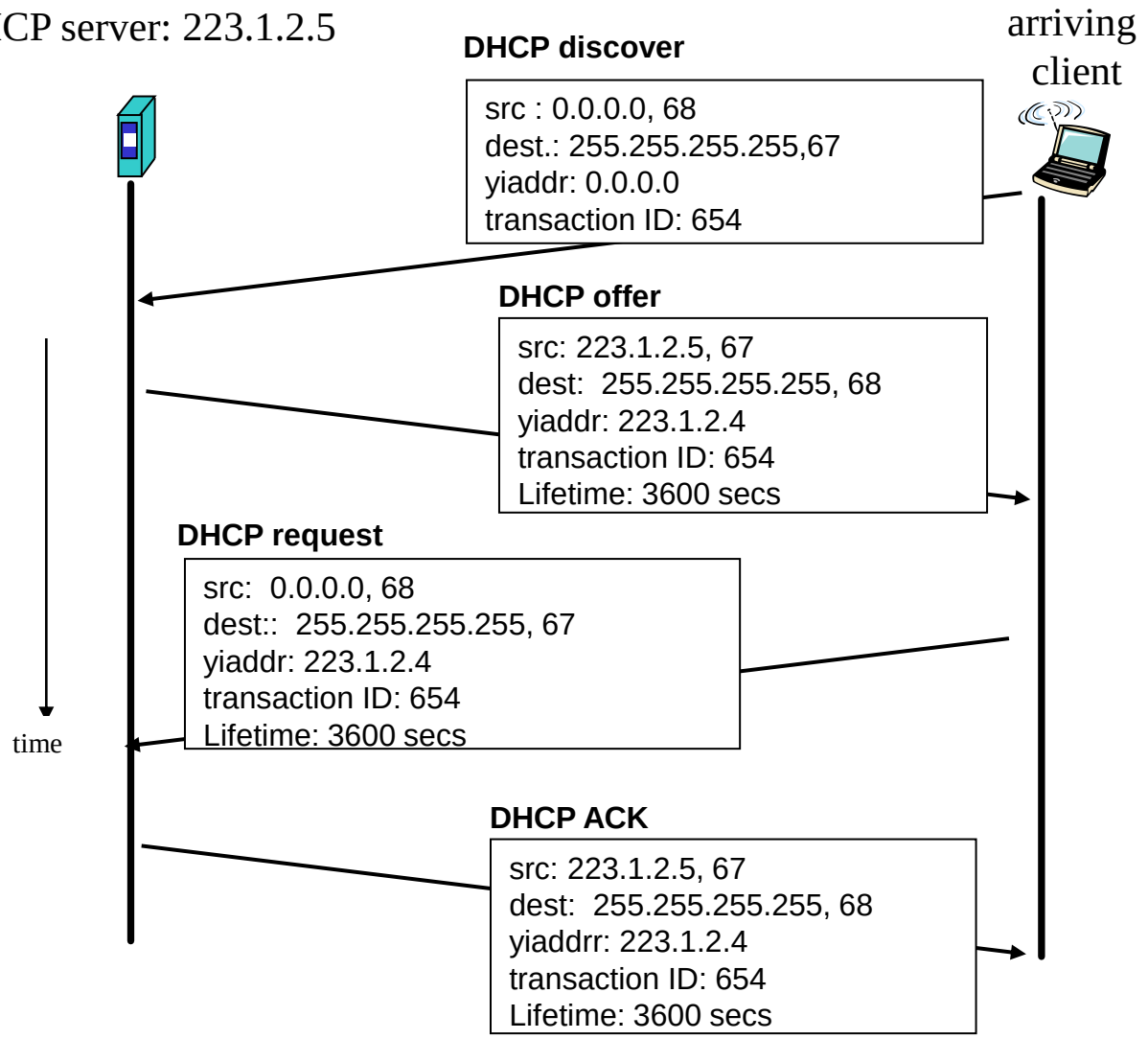
- DHCP allows
  - » host to dynamically obtain its IP address from network server
    when it joins network
  - » reuse of addresses

- DHCP overview
  - » host broadcasts "**DHCP discover**" msg
  - » DHCP server responds with "**DHCP offer**" msg
  - » host requests IP address "**DHCP request**" msg
  - » DHCP server sends address "**DHCP ack**" msg

DHCP - Client-server Scenario



DHCP Client-server



To Think…

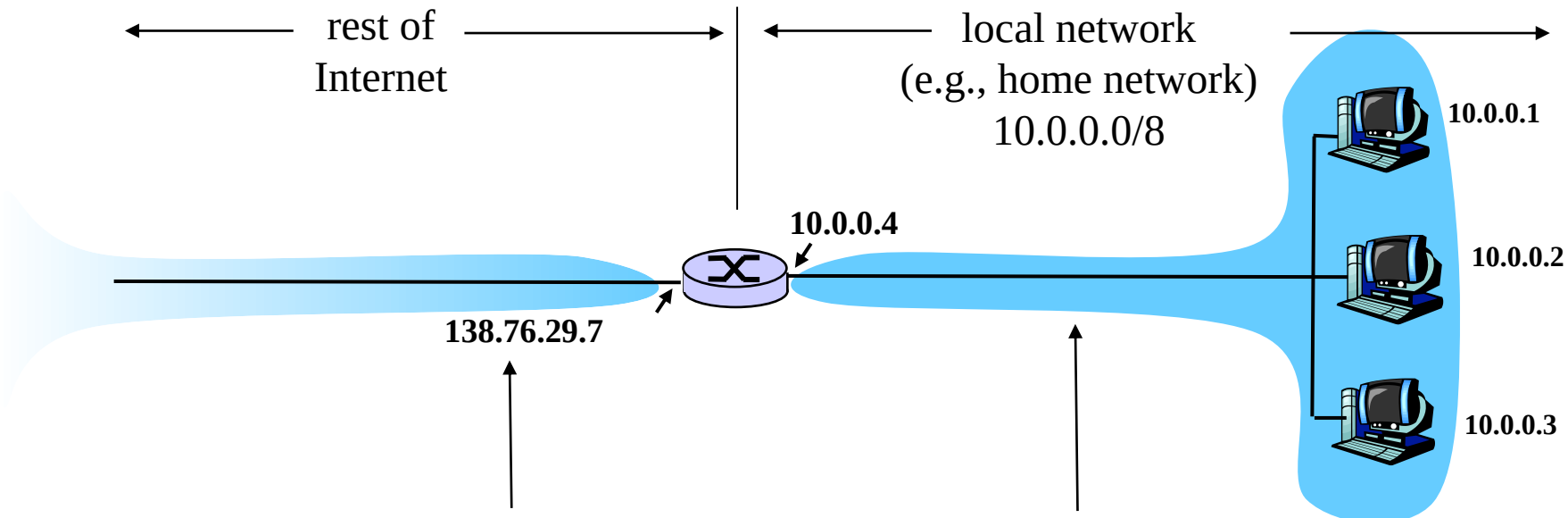
- Is it sufficient for an arriving client to acquire an IP address?
- What other relevant information shall this client obtain in order to start working with full functionality?

Network Address Translation

NAT - Network Address Translation

- Problem: shortage of IP addresses
  - » Initial design of the Internet did not foresee the later explosive growth
- Interim solution
  - » Configure many hosts with private IP addresses
    - – not routable on the Internet
  - » Translate those addresses to a public IP address on the gateway
- Private addressing blocks
  - » 10.0.0.0/8
  - » 172.16.0.0/12
  - » 192.168.0.0/16

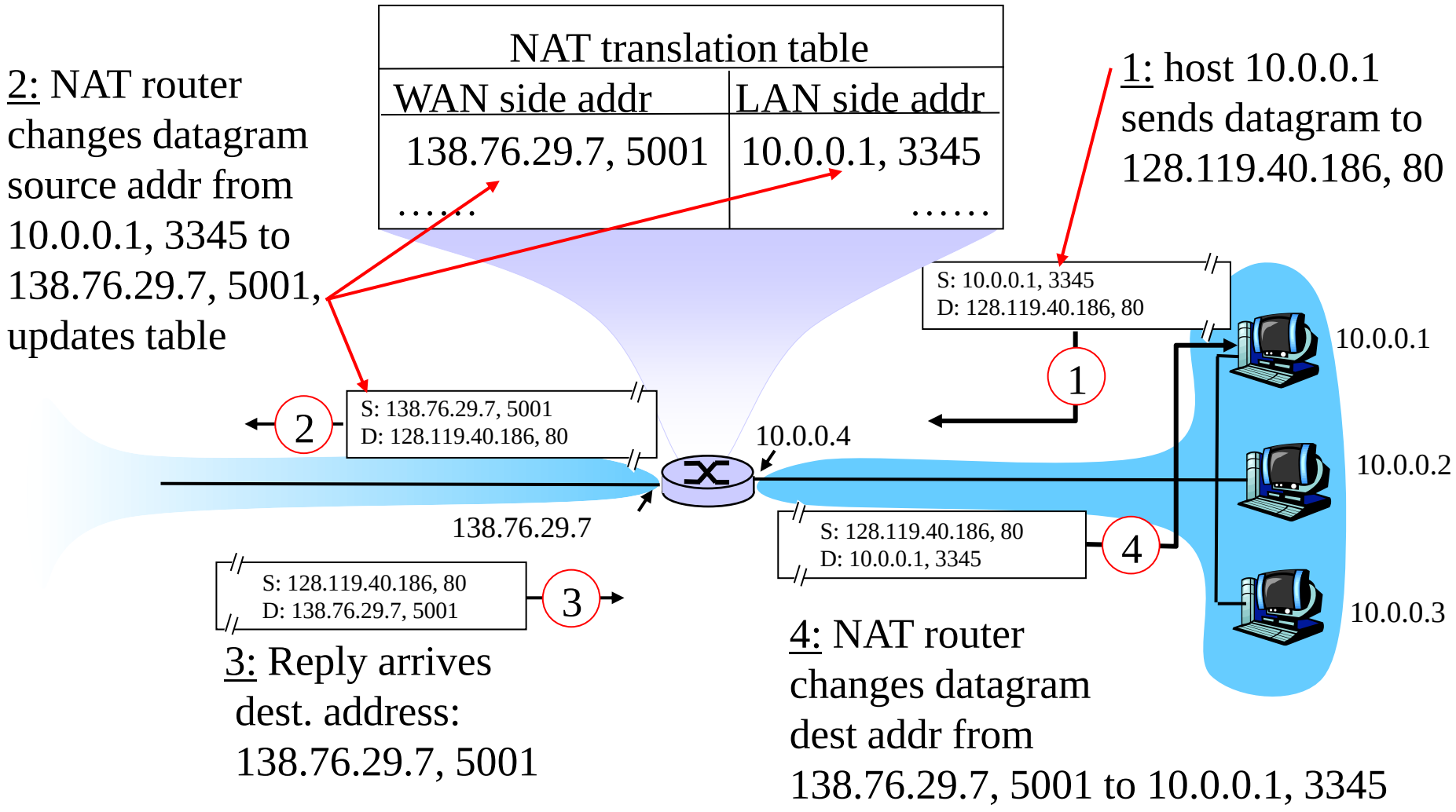
NAT - Network Address Translation



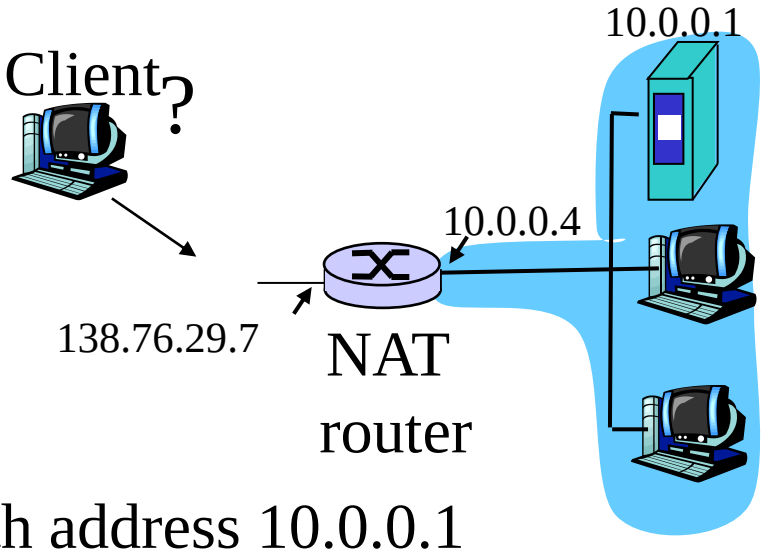
All datagrams *leaving* local network have same single source NAT IP address: 138.76.29.7, different source port numbers

Datagrams with source or destination in this network have 10.0.0.0/8 address for source, destination (as usual)

NAT - Network Address Translation



NAT Traversal



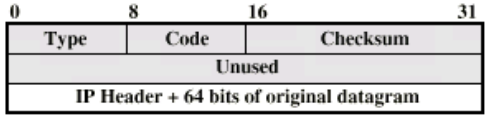
- Client wants to connect to server with address 10.0.0.1
  - » but server address 10.0.0.1 is private
  - » only one externally visible NATed address: 138.76.29.7
- Possible solution – Port forwarding
  - » statically configure NAT

    to forward incoming connection requests at given port to server
  - » e.g., (138.76.29.7, port 2500) always forwarded to 10.0.0.1 port 25000

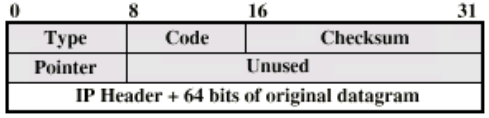
Internet Control Message Protocol

ICMP - Internet Control Message Protocol

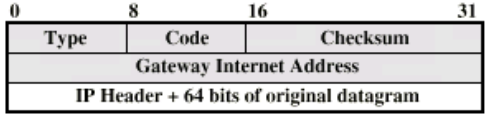
- Used by router or host
  - » to send layer 3 error or control messages
  - » to other hosts or routers
- Carried in IP datagrams



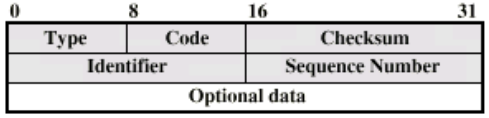
(a) Destination Unreachable; Time Exceeded; Source Quench



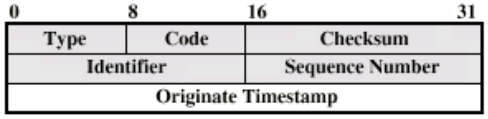
(b) Parameter Problem



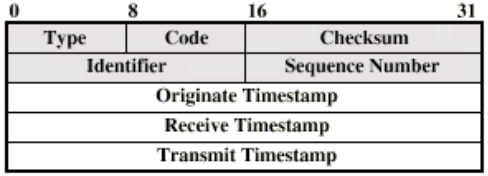
(c) Redirect



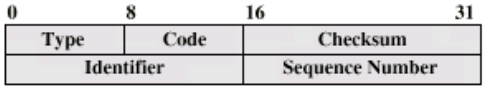
(d) Echo, Echo Reply



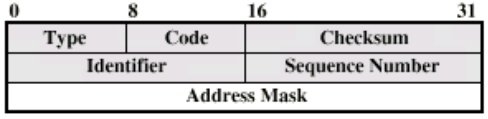
(e) Timestamp



(f) Timestamp Reply



(g) Address Mask Request



(h) Address Mask Reply

Type	Code	Description
0	0	echo reply (ping)
3	0	dest. network unreachable
3	1	dest host unreachable
3	2	dest protocol unreachable
3	3	dest port unreachable
3	6	dest network unknown
3	7	dest host unknown
4	0	source quench (congestion control - not used)
5	0	redirect for network
5	1	redirect for host
8	0	echo request (ping)
9	0	route advertisement
10	0	router discovery
11	0	TTL expired
12	0	bad IP header

Ping – Echo Request, Echo Reply

- Example 1: success

```
$ ping 193.137.55.13
PING 193.137.55.13 (193.137.55.13) 56(84) bytes of data.
64 bytes from 193.137.55.13: icmp_seq=1 ttl=251 time=0.975 ms
64 bytes from 193.137.55.13: icmp_seq=2 ttl=251 time=1.77 ms
64 bytes from 193.137.55.13: icmp_seq=3 ttl=251 time=1.64 ms
64 bytes from 193.137.55.13: icmp_seq=4 ttl=251 time=1.58 ms
64 bytes from 193.137.55.13: icmp_seq=5 ttl=251 time=1.63 ms
64 bytes from 193.137.55.13: icmp_seq=6 ttl=251 time=1.36 ms

--- 193.137.55.13 ping statistics ---
6 packets transmitted, 6 received, 0% packet loss, time 5004ms
rtt min/avg/max/mdev = 0.975/1.496/1.779/0.263 ms
```

- Example 2: failure

```
$ ping 193.137.55.1
PING 193.137.55.1 (193.137.55.1) 56(84) bytes of data.
```

(nothing; user gets tired of waiting and hits Ctrl-C)

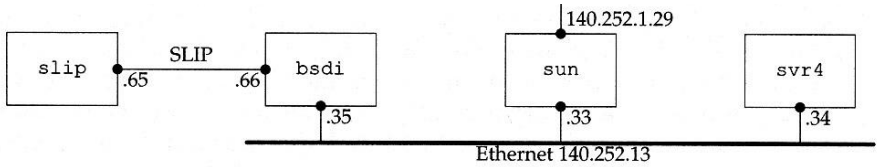
```
--- 193.137.55.1 ping statistics ---
20 packets transmitted, 0 received, 100% packet loss, time 18997ms
```

Traceroute and ICMP

- Source sends series of UDP segments to destination
  - » first segment has TTL =1
  - » second segment has TTL=2, …
  - » unlikely dst port number
- TTL decremented at every router
- At n^{th} router, TTL reaches 0
  - » router discards datagram
  - » sends to source: ICMP TTL expired
  - » message includes router name & IP address
- When ICMP message arrives, source calculates RTT
- Traceroute does this 3 times for each TTL value
- Stop criterion
  - » UDP segment eventually arrives at destination host
  - » destination returns ICMP “dest port unreachable” packet
  - » source stops

```
svr4% traceroute slip
traceroute to slip (140.252.13.65), 30 hops max. 40 byte packets
1 bsdi (140.252.13.35) 20 ms 10 ms 10 ms
2 slip (140.252.13.65) 120 ms 120 ms 120 ms

slip% traceroute svr4
traceroute to svr4 (140.252.13.34), 30 hops max, 40 byte packets
1 bsdi (140.252.13.66) 110 ms 110 ms 110 ms
2 svr4 (140.252.13.34) 110 ms 120 ms 110 ms
```



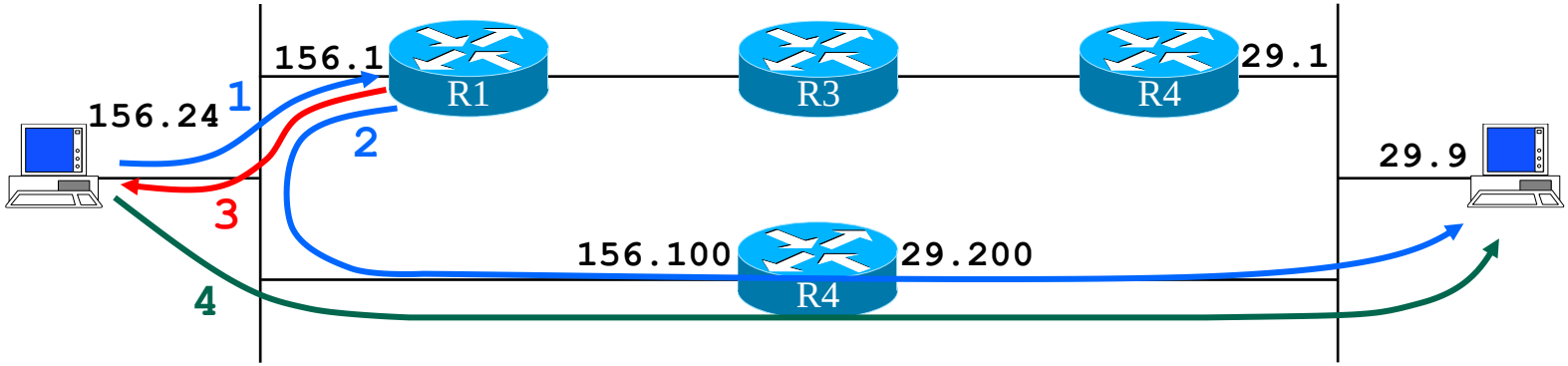
ICMP Redirect

- General routing principle of the TCP/IP architecture
  - » routers have extensive knowledge of routes
  - » hosts have minimal routing information ➔ learn routes also from ICMP redirects
- ICMP redirect message
  - » Sent by router R1 to source host A
    - – when R1 receives a packet from A with destination = B, and R1
      - finds that the next hop is R2 and
      - A is on-link with R2
  - » R1 sends ICMP redirect to A saying next hop for destination B is R2
  - » A stores the new next hop in the routing cache

ICMP Redirect Format:

Type=5	Code	Checksum
IP address of router that should be used		
IP header and first 8 bytes of data from the original datagram …		

ICMP Redirect Example



	src IP addr	dst IP addr	proto	data part
1:	193.154.156.24	193.154.29.9	udp	xxxxxxx
2:	193.154.156.24	193.154.29.9	udp	xxxxxxx
3:	193.154.156.1	193.154.156.24	icmp	type=redir code=host chksm 193.154.156.100 xxxxxxx (28 bytes of 1)
4:	193.154.156.24	193.154.29.9	udp	.....

ICMP Redirect Example

- The redirection does not appear in the forwarding table

```
$ netstat -nr
Destination     Gateway         Genmask         Flags  Metric  Ref    Use  Iface
193.154.156.0   0.0.0.0         255.255.255.0   U      0       0    63311  eth0
0.0.0.0         193.154.156.1   0.0.0.0         UG     1       0   183436  eth0
```

- However, it can be seen in the cache

```
$ ip route show cache
193.154.29.9 via 192.154.156.100 dev eth0
    cache <redirected> expires 279sec
```

- Some hosts reject redirects for security reasons

IPv6

Why IPv6?

- Lack of IPv4 addresses – most pressing reason
  - » Explosive Internet growth unforeseen by original designers
  - » NAT is a temporary band-aid
- IETF developed new IP version: **IPv6**
  - » Huge number of addresses
  - » 128-bit addresses (16 octets, 8 *shorts*) – huge number of addresses
  - » Better QoS support (native flow label)
  - » Better multicast support
  - » Native security functions (peer authentication, data encryption)
  - » Host autoconfiguration (*plug-n-play*)
  - » Prefix aggregation easier ➔ fewer entries in routing tables
  - » More efficient routing

IPv6 Addresses

- 128-bit addresses ➔ 3.4×10^{38} possible addresses
  - » 665 570 793 348 866 943 898 599 addresses per m^2 on earth!
- Represented in hexadecimal
  - » 2528:8653:294c:0000:0000:90af:0900:7654
- Consecutive zeroes may be abbreviated by ::
  - » ff01:0:0:0:0:0:0:43 ➔ ff01::43
  - » Only one :: in an address to avoid ambiguity
- Prefix length similar to IPv4 CIDR
  - » ff01::43/64 ➔ prefix is 64 bits long
- Loopback address is ::1 /128
- IPv4 addresses can be represented as IPv6 addresses to simplify applications
  - » ::ffff:5.6.7.8 (IPv4-mapped address)

Adresses – Link-Local, Global Unicast, Anycast, Multicast

- *Link-Local*
  - » Used for communication between hosts in the same LAN/link
  - » Automatically created from MAC address
  - » Routers do not foward packets having *Link-Local* destination addresses
- *Global Unicast*
  - » Internet routable global addresses
  - » Can be generated automatically from prefix and MAC address
- *Anycast*
  - » Assigned to several devices; a packet is received by any (only one) of them
  - » Allocated from the unicast address space
  - » Special case: subnet device anycast addresses
- *Multicast*
  - » Group address: packet received by all the members of the group
  - » Scoped

Address Formats

n bits	m bits	128-n-m bits
001 global route prefix	subnet ID	interface ID

Global Unicast Address
(2000::/3)

10 bits	54 bits	64 bits
1111111010	0	interface ID

Link-Local Unicast address
(fe80::/10)

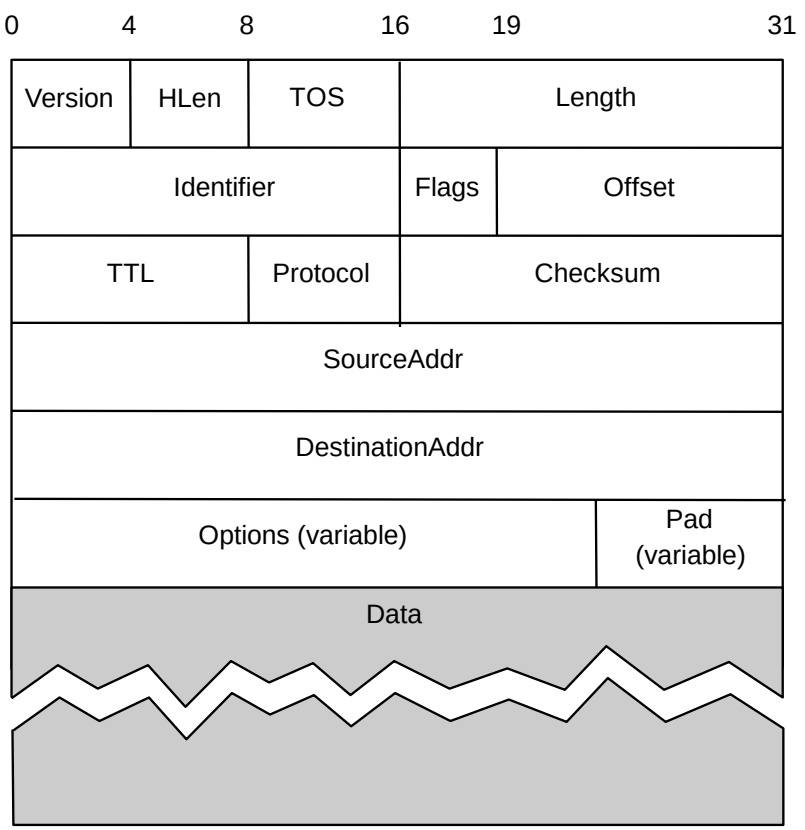
n bits	128-n bits
subnet prefix	00000000000000

Subnet device anycast address

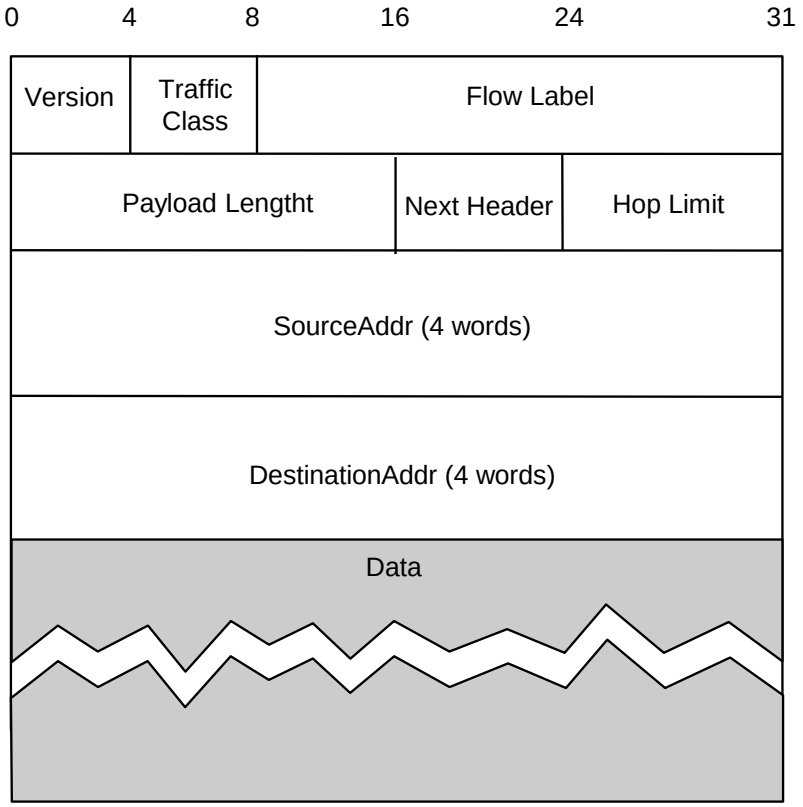
8	4	4	112 bits
11111111	flgs	scop	group ID

Multicast address
Scope – link, site, global, ...
(ff::/8)

Headers IPv4 and IPv6



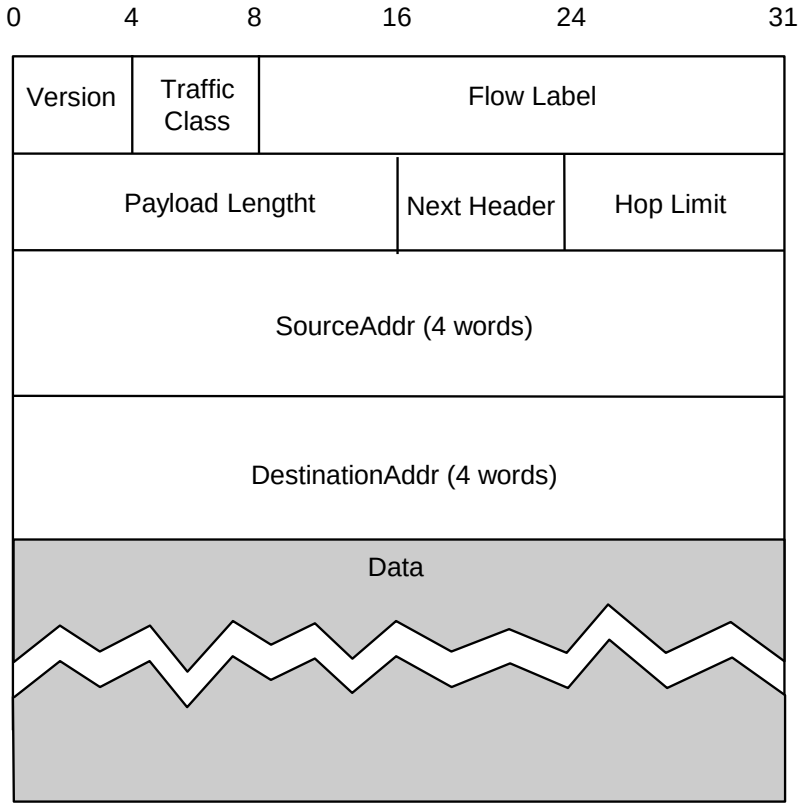
IPv4



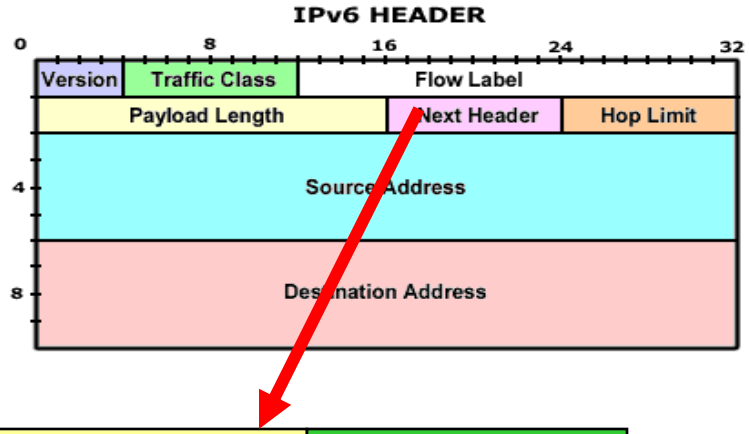
IPv6

IPv6 Header

- Traffic class
  - » Similar to IPv4 ToS / DSCP
- Flow label → identifies packet flow
  - » QoS, resource reservation
  - » Packets receive same service
- Payload length
  - » Header not included
- Next header
  - » Identifies next header/extension
- Hop limit = TTL (v4)
  - » But defined properly
- No checksum
  - » Rely on stronger layer 2 error detection
- No options
  - » Uses extension headers instead



Extension Headers



Nested headers, akin to encapsulation

IPv6 Header Next Header = TCP	TCP header + data		
IPv6 Header Next Header = Routing	Routing Header Next Header = TCP	TCP header + data	
IPv6 Header Next Header = Routing	Routing Header Next Header = Fragment	Fragment Header Next Header = TCP	Fragment of TCP header + data

IPv6	Hop-by-hop	Routing	Destination	Fragment	Authenticat.	ESP	TCP

Extension Headers

- Hop-by-hop
  additional information, inspected by every node traversed by the packet
  other headers are inspected only at the destination or at pre-defined nodes
- Routing: Options that may affect the route
- Destination: Information for the destination node
- Fragmentation: Only by the source; source can find the Path MTU
- Authentication: Authentication (signature) of packet header
- ESP: Data encryption

Configuration examples in Linux

```
tux13:~# ifconfig eth0 inet6 add 2000:0:0:1::1/64
tux13:~# ifconfig eth0
eth0      Link encap:Ethernet  HWaddr 00:C0:DF:08:D5:99
          inet addr:172.16.1.13  Bcast:172.16.1.255  Mask:255.255.255.0
          inet6 addr: 2000:0:0:1::1/64 Scope:Global
          inet6 addr: fe80::2c0:dfff:fe08:d599/10 Scope:Link
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:81403 errors:0 dropped:0 overruns:0 frame:0
          TX packets:2429 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:100
          RX bytes:4981344 (4.7 MiB)  TX bytes:260692 (254.5 KiB)
          Interrupt:5
tux13:~# route -A inet6 add 2000::/3 gw 2000:0:0:1::aa
tux13:~# route -A inet6
Kernel IPv6 routing table
Destination                    NextHop         Flags Metric Ref Use Iface
::1/128                        ::              U     0      0   0    lo
2000:0:0:1::1/128              ::              U     0      0   0    lo
2000:0:0:1::/64                ::              UA    256    0   0    eth0
2000::/3                       2000:0:0:1::aa  UG    1      0   0    eth0
fe80::2c0:dfff:fe08:d599/128   ::              U     0      0   0    lo
fe80::/10                      ::              UA    256    0   0    eth0
ff00::/8                       ::              UA    256    0   0    eth0
::/0                           ::              UDA   256    0   0    eth0
```

Protocol Neighbor Discovery (ND)

- IPv6 node uses ND to
  - » find other nodes in the same link / LAN
  - » find a neighbor's MAC address from its IPv6 address
    - – Replaces ARP, but much more efficient
  - » find router(s) in its network
  - » mantain information about neighbor nodes

- ND uses ICMPv6 messages
  - » using *Link Local* addresses

ND Messages

- ***Neighbor Solicitation***

  Sent by a host to obtain MAC address of a neighbor / verify its presence

  sent to a multicast address built from the neighbor's IPv6 address (instead of broadcast)

- ***Neighbor Advertisement***

  Reply to the solicitation

- ***Router Advertisement***

  Used by routers to send information about the network prefix, periodically or under request

  sent to the *Link Local multicast* IPv6 address

- ***Router Solicitation***

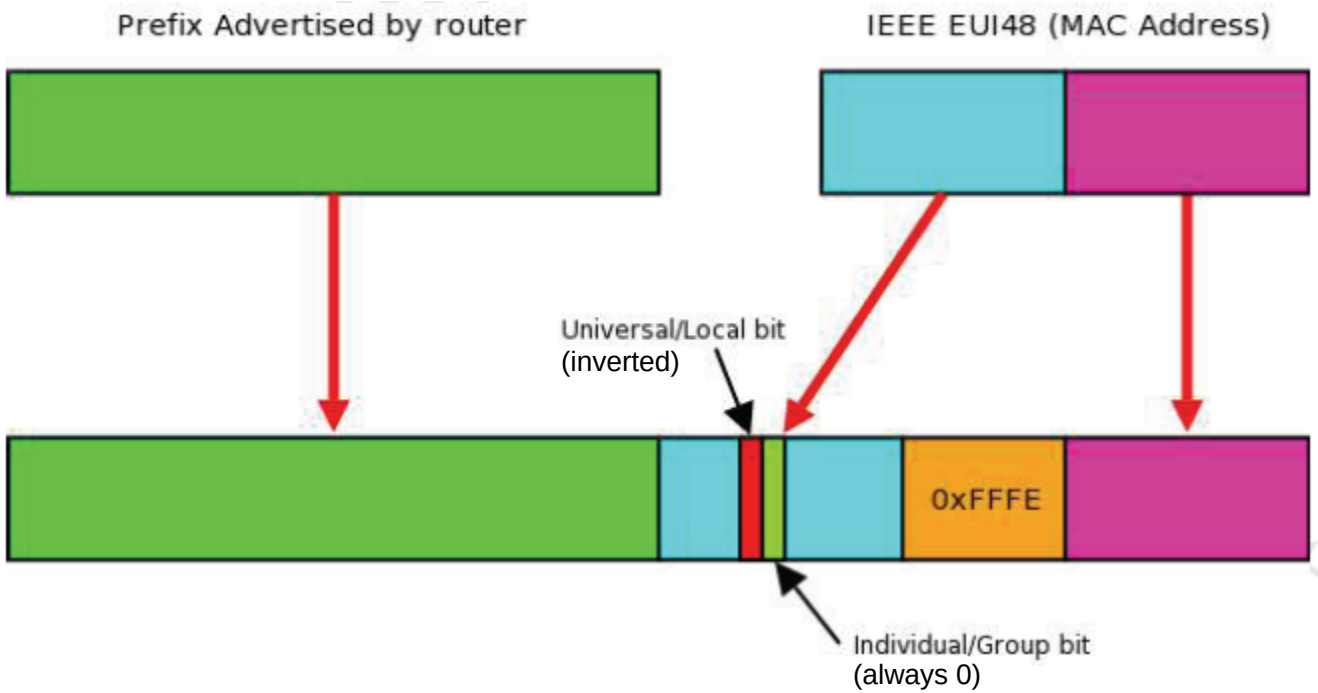
  Request a *Router Advertisment* from router(s) on the link

- ***Redirect***

  Used by a router to inform a host about the best route to a destination

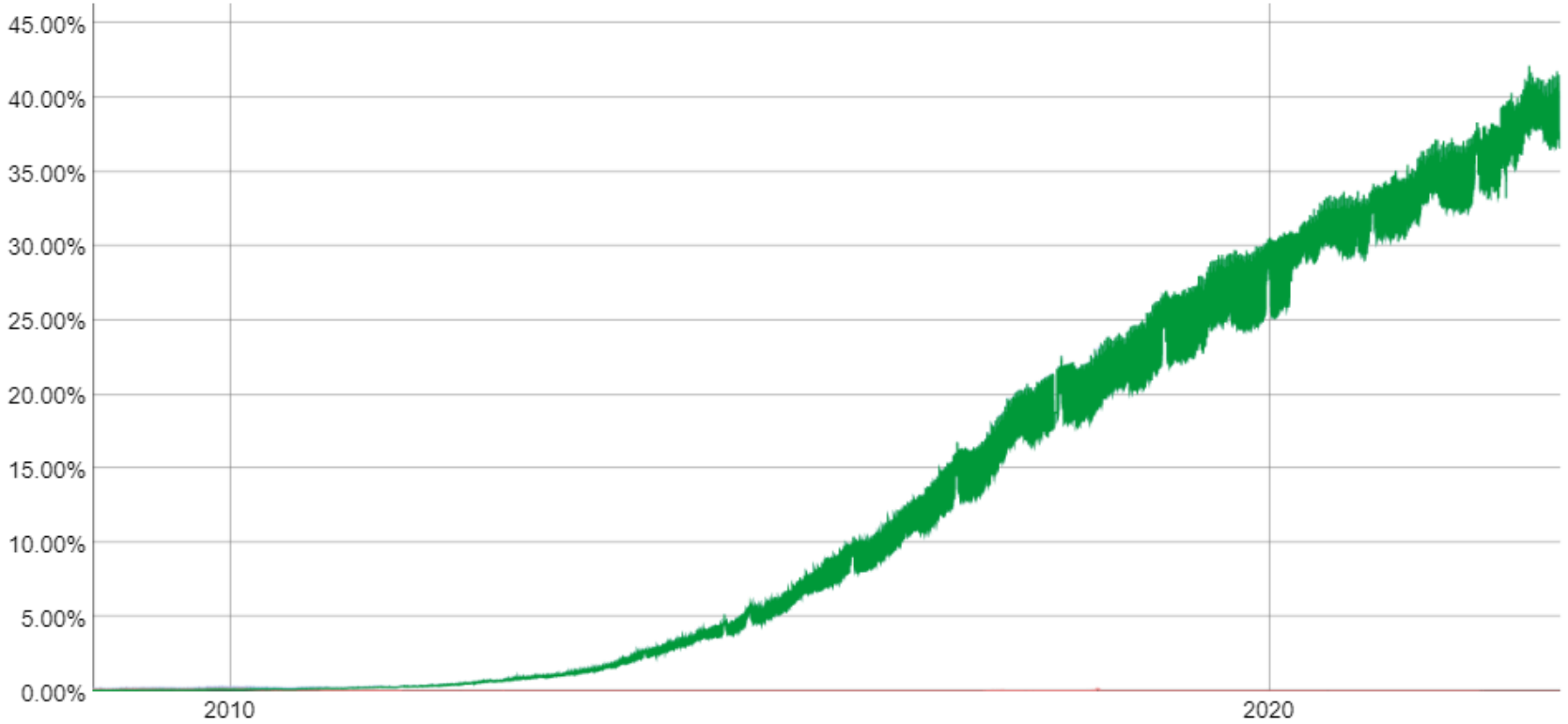
Stateless Address Autoconfiguration (SLAAC)

- Automatic IPv6 address configuration based on *Router Advertisements*
- Combines advertised prefix and EUI-64 identifier MAC address of interface
  - » Ex.: prefix 2000:0:0:1::/64 + MAC addr. 00:c0:df:08:d5:99 ➔ 2000::1:2c0:dfff:fe08:d599



IPv6 — Current Usage

- As of October 2022, nearly 40% of accesses to Google use IPv6



https://www.google.com/intl/en/ipv6/statistics.html

Homework

1. Review slides

2. Read from Kurose&Ross
   - » Chapter 4 – The Network Layer
     (this set of slides follows mainly Kurose&Ross)

3. Or, from Tanenbaum,
   - » Chapter 5 – The Network Layer

4. Answer questions at moodle