

source: computer-networks-webdesign.com



CSCI 6760 - Computer Networks Spring 2017

Instructor: Prof. Roberto Perdisci
perdisci@cs.uga.edu

These slides are adapted from the textbook slides by J.F. Kurose and K.W. Ross

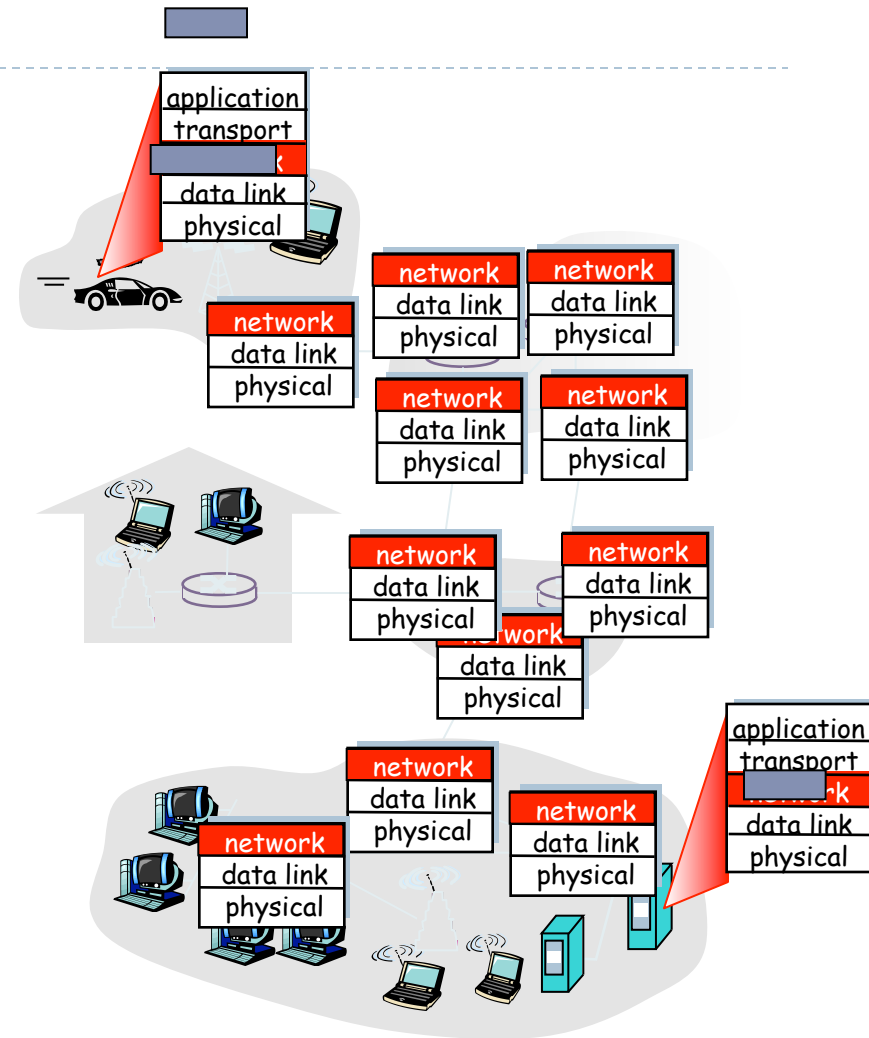
Chapter 4: Network Layer

Chapter goals:

- ▶ understand principles behind network layer services:
 - ▶ network layer service models
 - ▶ forwarding versus routing
 - ▶ how a router works
 - ▶ routing (path selection)
 - ▶ dealing with scale
 - ▶ advanced topics: IPv6, mobility
- ▶ instantiation, implementation in the Internet

Network layer

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



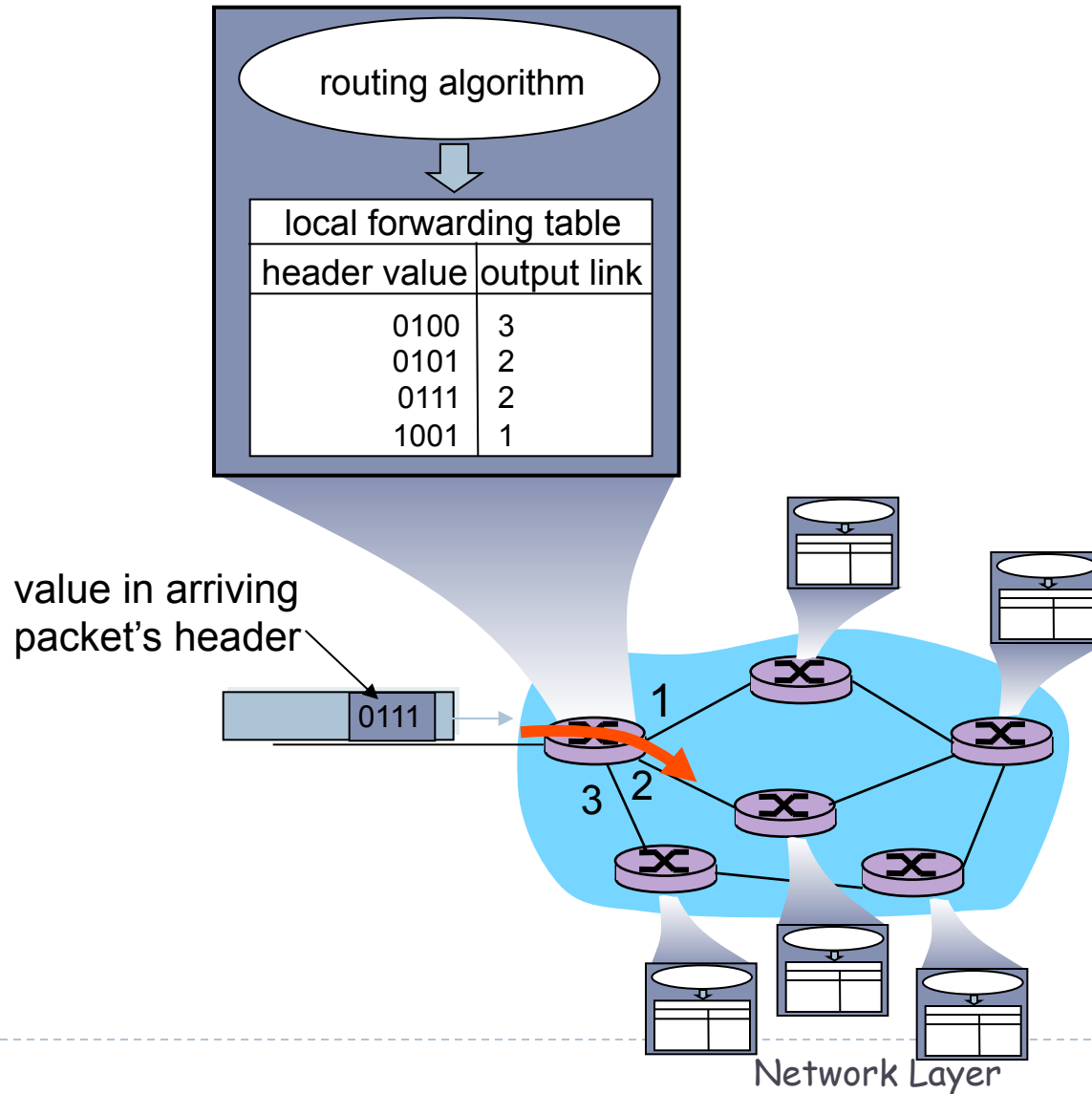
Two Key Network-Layer Functions

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

analogy:

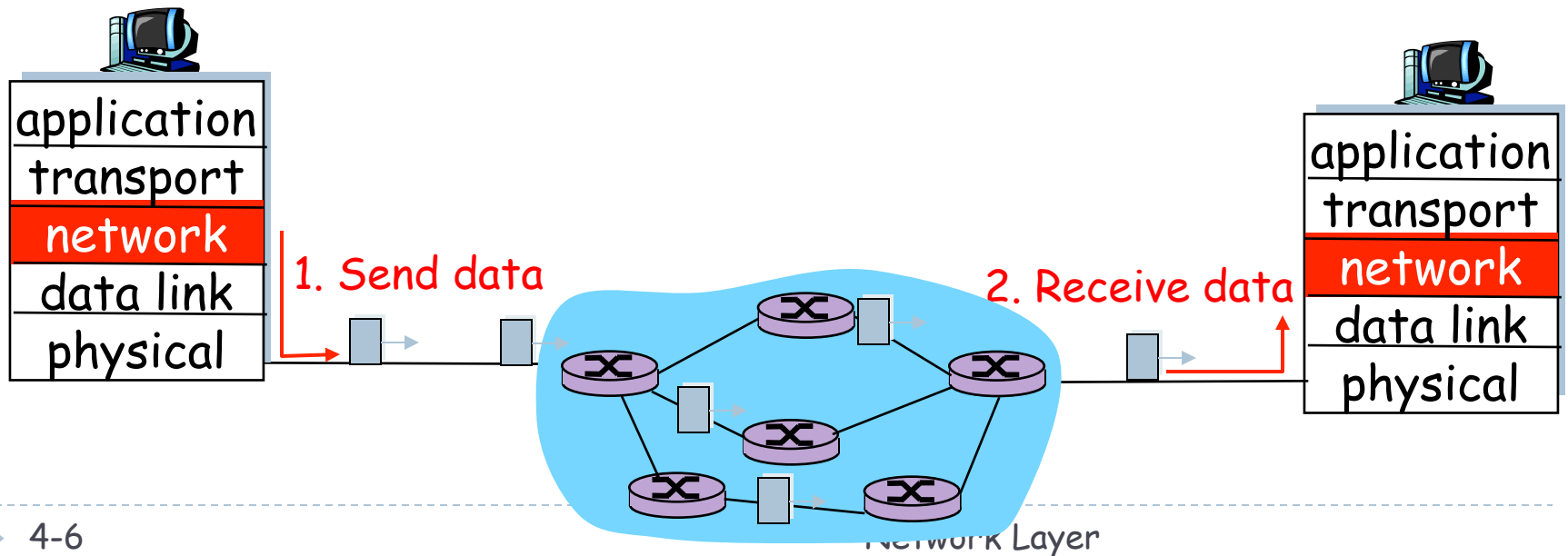
- *routing*: process of planning trip from source to dest
- *forwarding*: process of getting through single interchange

Interplay between routing and forwarding



Datagram networks

- ▶ no call setup at network layer
- ▶ routers: no state about end-to-end connections
 - ▶ no network-level concept of “connection”
- ▶ packets forwarded using destination host address
 - ▶ packets between same source-dest pair may take different paths



IPv4 forwarding table

$2^{32} = 4$ billion
possible entries

<u>Destination Address Range</u>	<u>Link Interface</u>
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

<u>Prefix Match</u>	<u>Link Interface</u>
11001000 00010111 00010	0
11001000 00010111 00011000	1
11001000 00010111 00011	2
otherwise	3

Examples

DA: 11001000 00010111 00010110 10100001

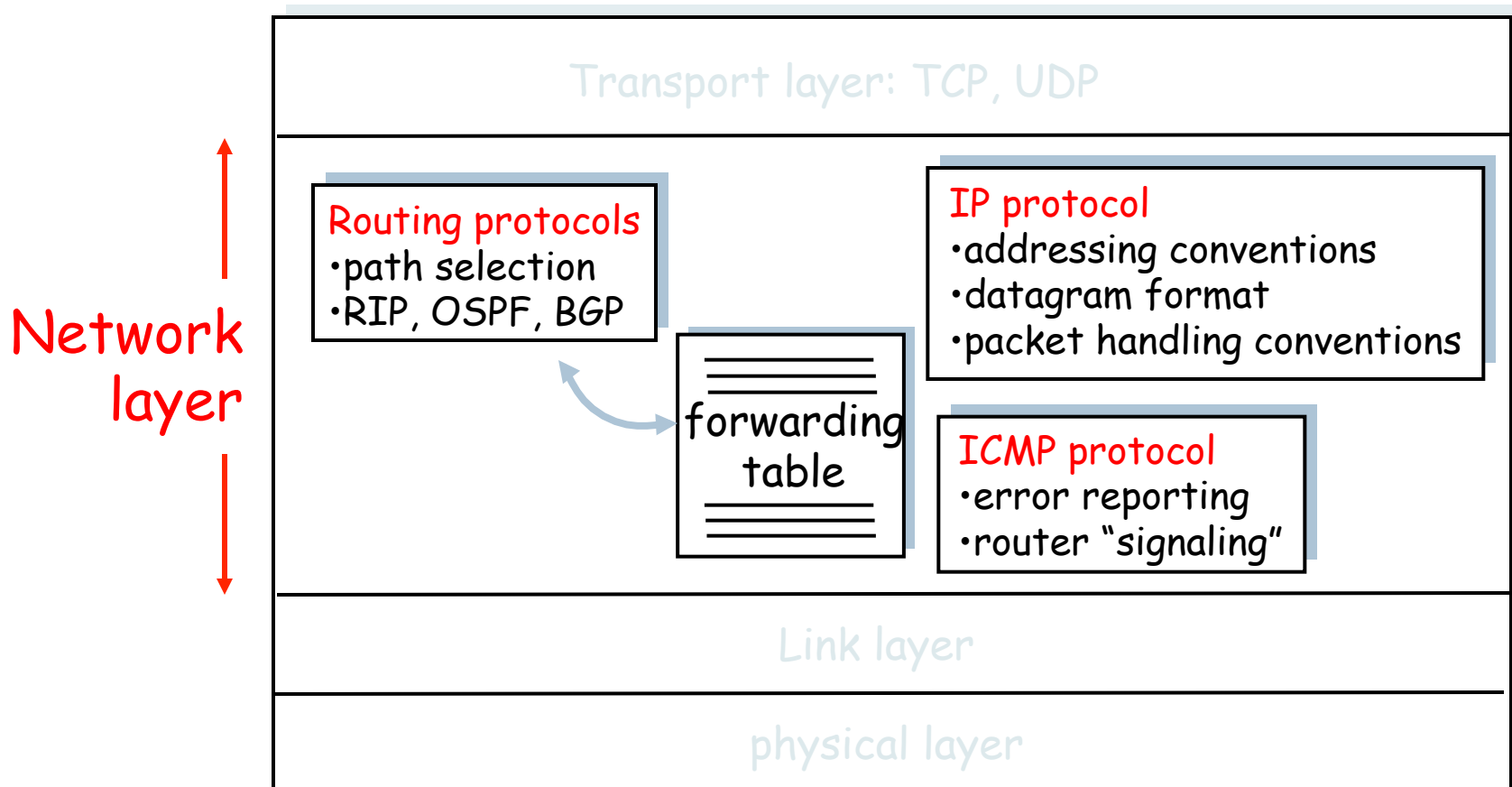
Which interface?

DA: 11001000 00010111 00011000 10101010

Which interface?

The Internet Network layer

Host, router network layer functions:



IPv4 datagram format

IP protocol version number

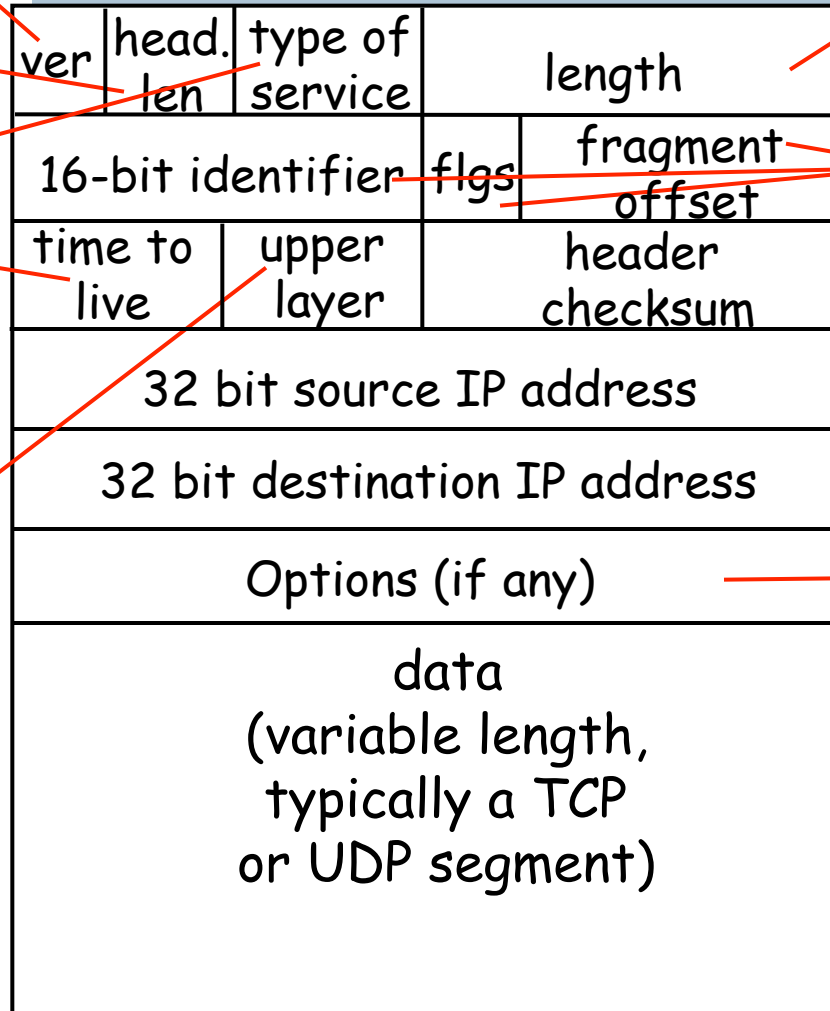
header length (bytes)

"type" of data

max number remaining hops (decremented at each router)

upper layer protocol to deliver payload to

32 bits



total datagram length (bytes)

for fragmentation/reassembly

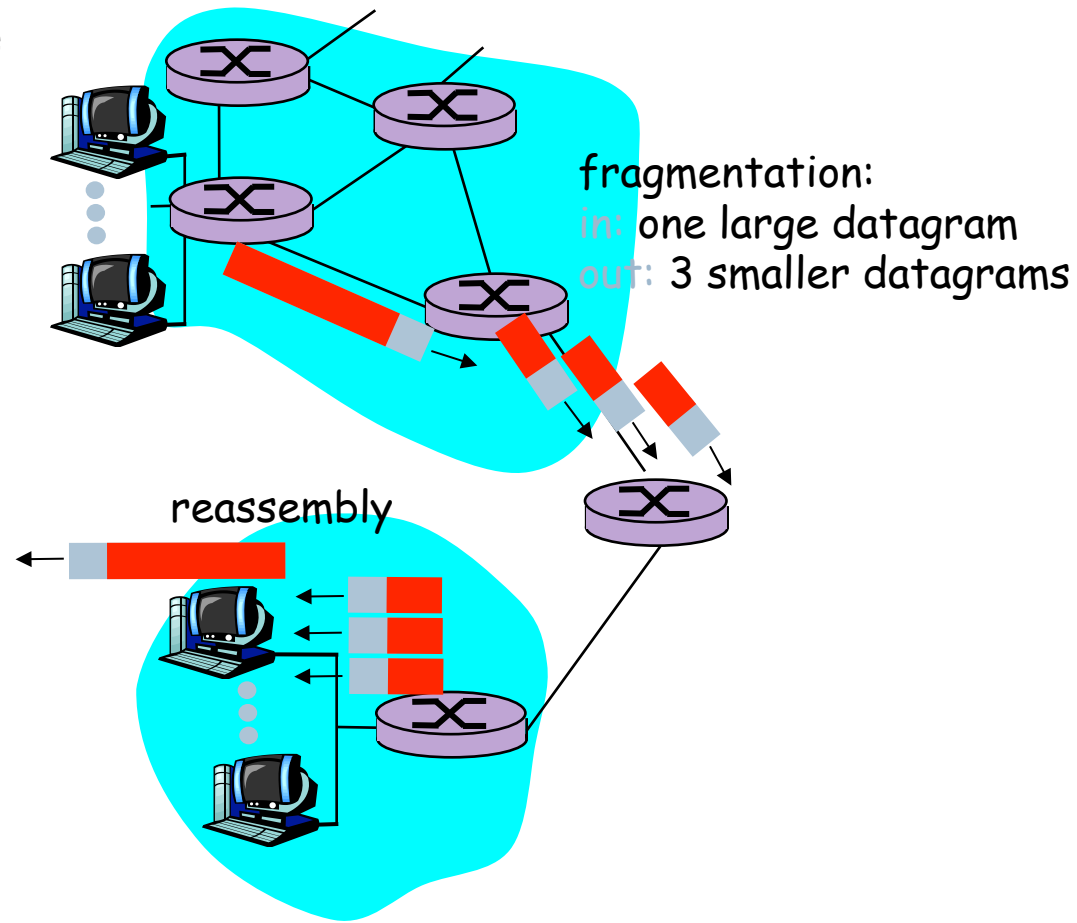
E.g. timestamp, record route taken, specify list of routers to visit.

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



IP Fragmentation and Reassembly

Example

- ❑ 4000 byte datagram (3980 Bytes for payload)
- ❑ MTU = 1500 bytes

1480 bytes in data field

offset =
 $1480/8 = 185$

	length	ID	fragflag	offset	
	=4000	=x	=0	=0	

One large datagram becomes several smaller datagrams

	length	ID	fragflag	offset	
	=1500	=x	=1	=0	

	length	ID	fragflag	offset	
	=1500	=x	=1	=185	

	length	ID	fragflag	offset	
	=1040	=x	=0	=370	

IP Fragmentation - Another Example

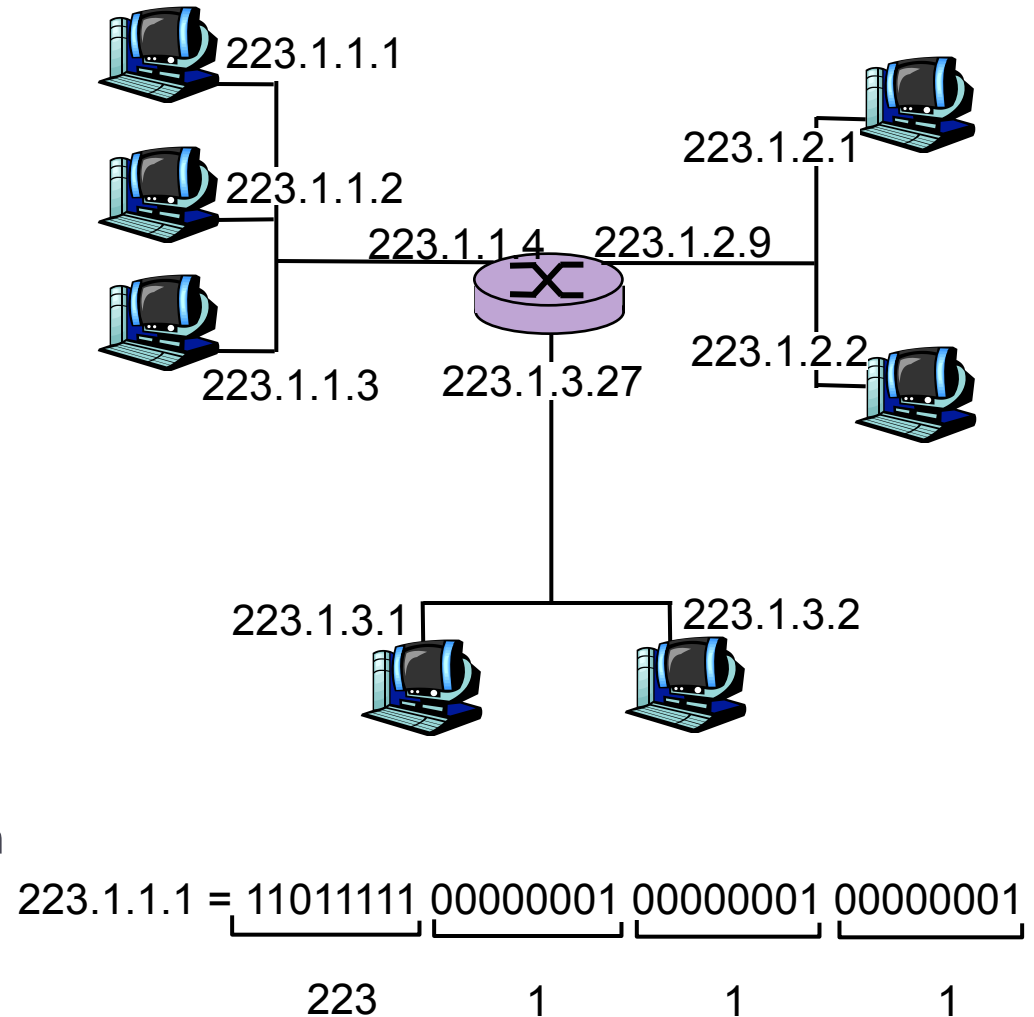
- ▶ Initial MTU = 3100 bytes (=3080 payload bytes)
- ▶ As packet is routed, it encounters a link with MTU = 820 bytes (=800 payload bytes)
- ▶ How will the fragments look like?
 - ▶ ID = 4325, Flag = 1, offset = 0, length = 820
 - ▶ ID = 4325, Flag = 1, offset = 100, length = 820
 - ▶ ID = 4325, Flag = 1, offset = 200, length = 820
 - ▶ ID = 4325, Flag = 0, offset = 300, length = 700

IP Fragmentation - Another Example

- ▶ Initial MTU = 3100 bytes (=3080 payload bytes)
- ▶ As packet is routed, it encounters a link with MTU = 930 bytes (=910 payload bytes)
- ▶ How will the fragments look like?
 - ▶ ID = 4325, Flag = 1, offset = 0, length = 924
 - ▶ ID = 4325, Flag = 1, offset = 113, length = 924
 - ▶ ID = 4325, Flag = 1, offset = 226, length = 924
 - ▶ ID = 4325, Flag = 0, offset = 339, length = 388

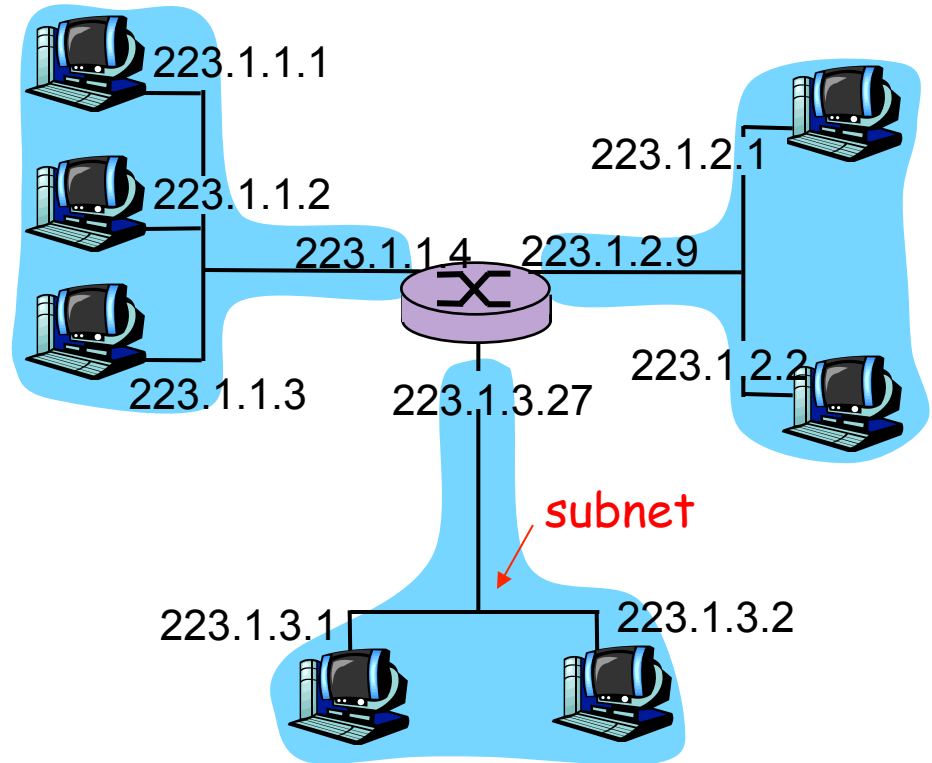
IPv4 Addressing: introduction

- ▶ IPv4 address: 32-bit identifier for host, router *interface*
- ▶ *interface*: connection between host/router and physical link
 - ▶ router's typically have multiple interfaces
 - ▶ host typically has one interface
 - ▶ IP addresses associated with each interface



Subnets

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

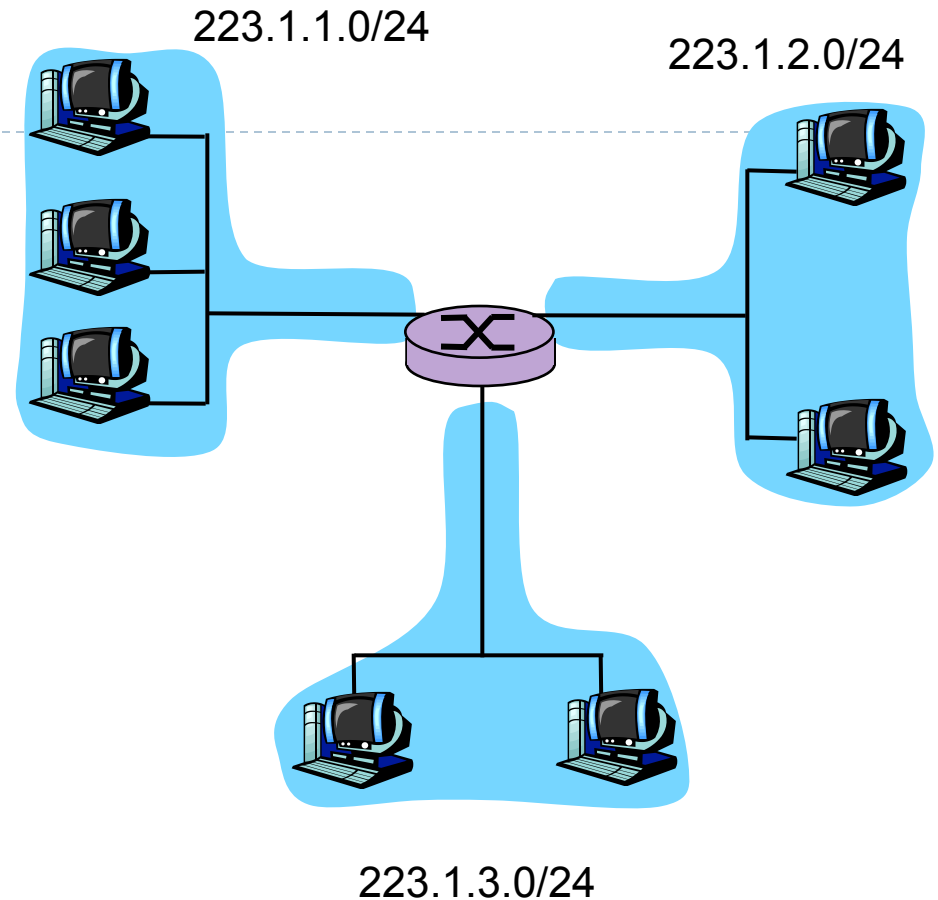


network consisting of 3 subnets

Subnets

Recipe

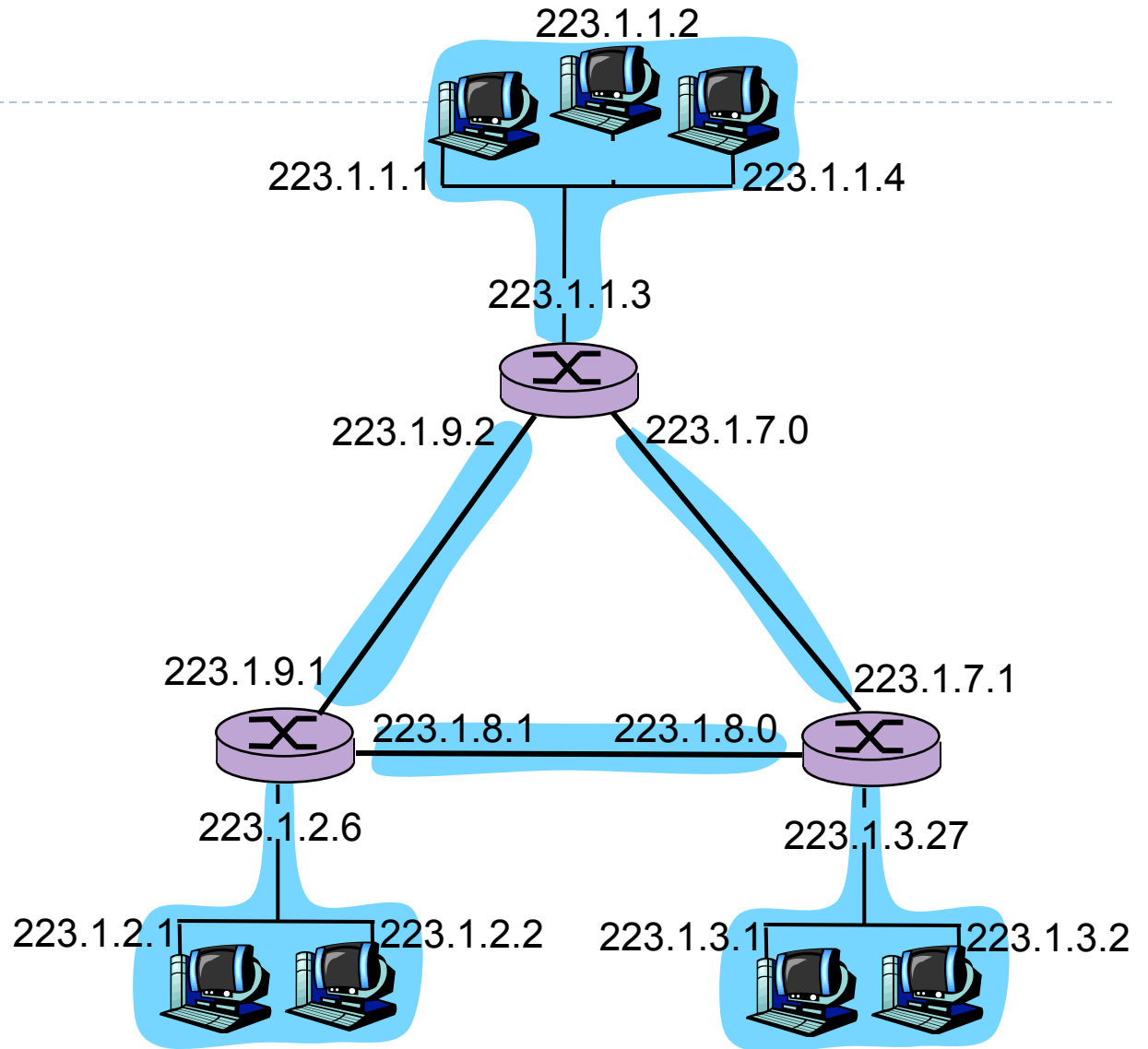
- ▶ To determine the subnets, detach each interface from its host or router, creating islands of isolated networks. Each isolated network is called a **subnet**.



Subnet mask: /24

Subnets

How many?



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

Computing the longest common CIDR

- ▶ **192.168.6.98**

- ▶ 192.168.(00000110).(01100010)

- ▶ **192.168.65.3**

- ▶ 192.168.(01000001).(00000011)

- ▶ **CIDR**

- ▶ 192.168.(00000000).(00000000)/17

- ▶ 192.168.0.0/17

- ▶ **Subnet Mask**

- ▶ 255.255.(10000000).(00000000)

- ▶ 255.255.128.0

- ▶ IP & SM = CIDR

- ▶ **172.18.5.215**

- ▶ 172.18.5.(11010111)

- ▶ **172.18.5.210**

- ▶ 172.18.5.(11010010)

- ▶ **CIDR**

- ▶ 172.18.5.(11010000)/29

- ▶ 172.18.5.208/29

- ▶ **Subnet Mask**

- ▶ 255.255.255.(11111000)

- ▶ 255.255.255.248

- ▶ IP & SM = CIDR

Computing the CIDR

- ▶ Assume we have the following IP addresses, what is their longest common CIDR?
 - ▶ 10.35.25.102, 10.35.27.23, 10.35.28.203, 10.35.30.124
 - ▶ CIDR = 10.35.24.0/21
 - ▶ Subnet Mask = 255.255.248.0

- ▶ Assume we have the following IP addresses, what is their longest common CIDR?
 - ▶ 172.17.2.102, 172.17.2.65, 172.17.2.87, 172.17.2.124
 - ▶ CIDR = 172.17.2.64/26
 - ▶ Subnet Mask = 255.255.255.192

Reserved Address Blocks

- ▶ **10.0.0.0/8** **Private network** RFC 1918
- ▶ **127.0.0.0/8** **Loopback** RFC 5735
- ▶ 169.254.0.0/16 Link-Local RFC 3927
- ▶ **172.16.0.0/12** **Private network** RFC 1918
- ▶ 192.0.0.0/24 Reserved (IANA) RFC 5735
- ▶ 192.0.2.0/24 TEST-NET-1, Documentation and example code RFC 5735
- ▶ 192.88.99.0/24 IPv6 to IPv4 relay RFC 3068
- ▶ **192.168.0.0/16** **Private network** RFC 1918
- ▶ 198.18.0.0/15 Network benchmark tests RFC 2544
- ▶ 198.51.100.0/24 TEST-NET-2, Documentation and examples RFC 5737
- ▶ 203.0.113.0/24 TEST-NET-3, Documentation and examples RFC 5737
- ▶ 224.0.0.0/4 Multicasts (former Class D network) RFC 3171
- ▶ 240.0.0.0/4 Reserved (former Class E network) RFC 1700
- ▶ **255.255.255.255** **Broadcast** RFC 919

IP addresses: how to get one?

Q: How does a *host* get IP address?

- ▶ hard-coded by system admin in a file
 - ▶ Windows: control-panel->network->configuration->tcp/ip->properties
 - ▶ UNIX: /etc/rc.config
- ▶ **DHCP: Dynamic Host Configuration Protocol**: dynamically get address from as server
 - ▶ “plug-and-play”

DHCP:

Dynamic Host Configuration Protocol

Goal: allow host to *dynamically* obtain its IP address from network server when it joins network

Can renew its lease on address in use

Allows reuse of addresses (only hold address while connected an “on”)

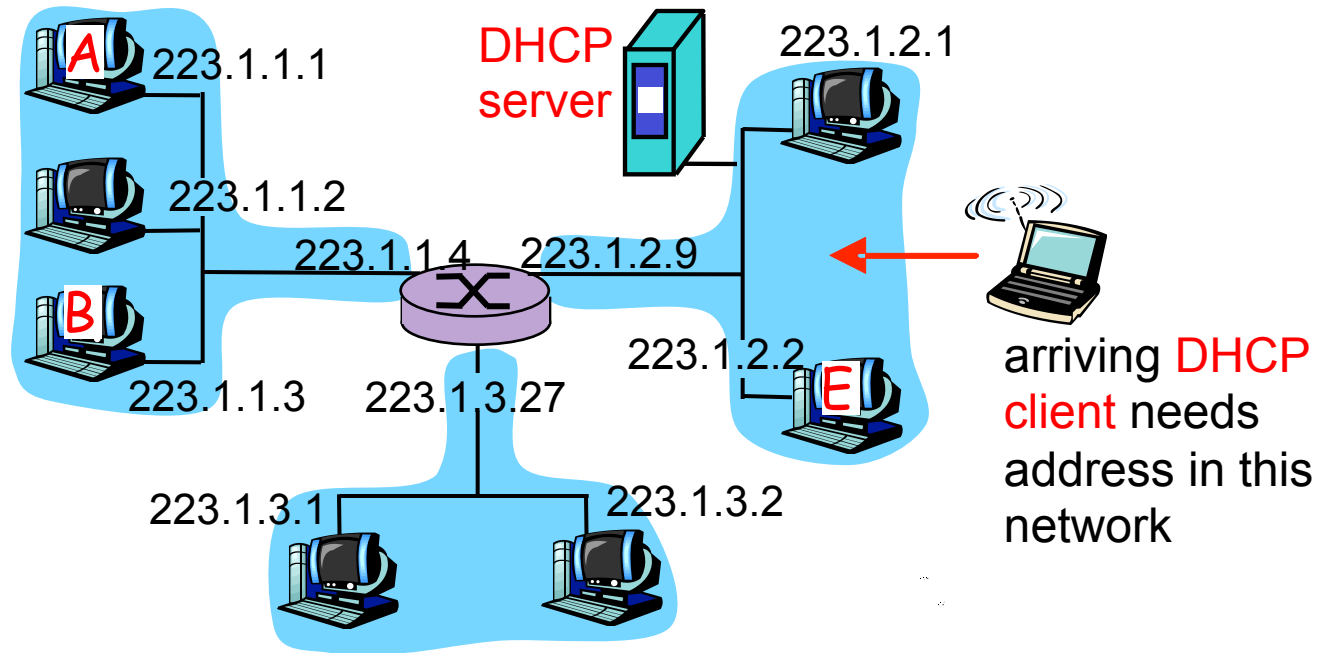
Support for mobile users who want to join network (more shortly)

DHCP overview:

- ▶ host broadcasts “DHCP discover” msg
 - ▶ Src=0.0.0.0 Dst=255.255.255.255
- ▶ DHCP server responds with “DHCP offer” msg
 - ▶ Src=DHCP Server’s IP Dst=255.255.255.255
- ▶ host requests IP address: “DHCP request” msg
 - ▶ Src=0.0.0.0 Dst=255.255.255.255
- ▶ DHCP server sends address: “DHCP ack” msg
 - ▶ Src=DHCP Server’s IP Dst: 255.255.255.255

RFC1531, Section 3.1 “Client-server interaction - allocating a network address”

DHCP client-server scenario



DHCP client-server scenario

DHCP server: 223.1.2.5

DHCP discover

src : 0.0.0.0, 68
dest.: 255.255.255.255,67
yiaddr: 0.0.0.0
transaction ID: 654

arriving
client



DHCP offer

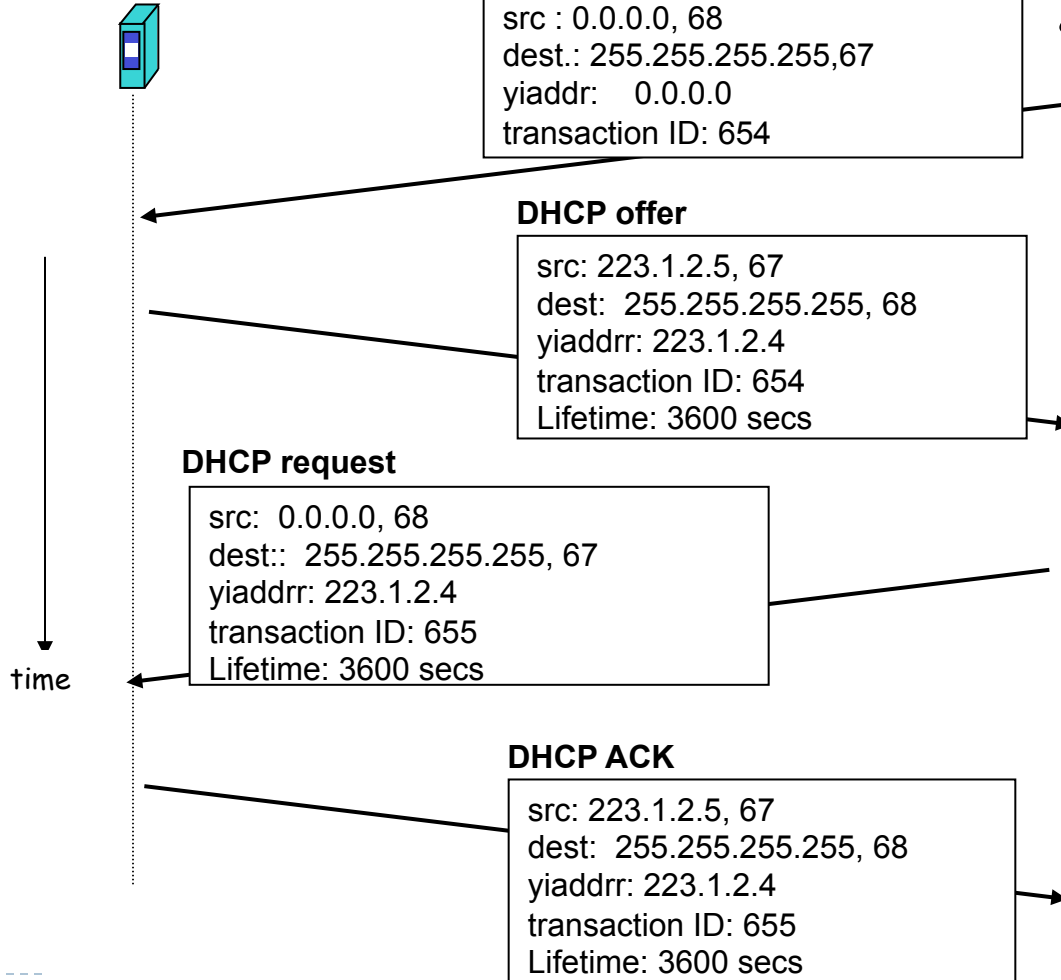
src: 223.1.2.5, 67
dest: 255.255.255.255, 68
yiaddr: 223.1.2.4
transaction ID: 654
Lifetime: 3600 secs

DHCP request

src: 0.0.0.0, 68
dest.: 255.255.255.255, 67
yiaddr: 223.1.2.4
transaction ID: 655
Lifetime: 3600 secs

DHCP ACK

src: 223.1.2.5, 67
dest: 255.255.255.255, 68
yiaddr: 223.1.2.4
transaction ID: 655
Lifetime: 3600 secs

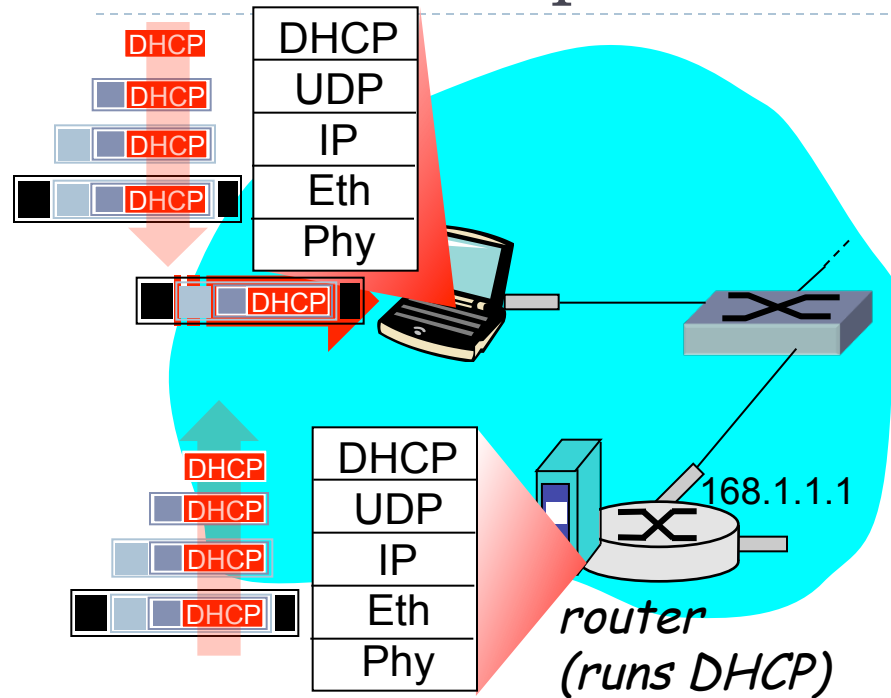


DHCP: more than IP address

DHCP can return more than just allocated IP address on subnet:

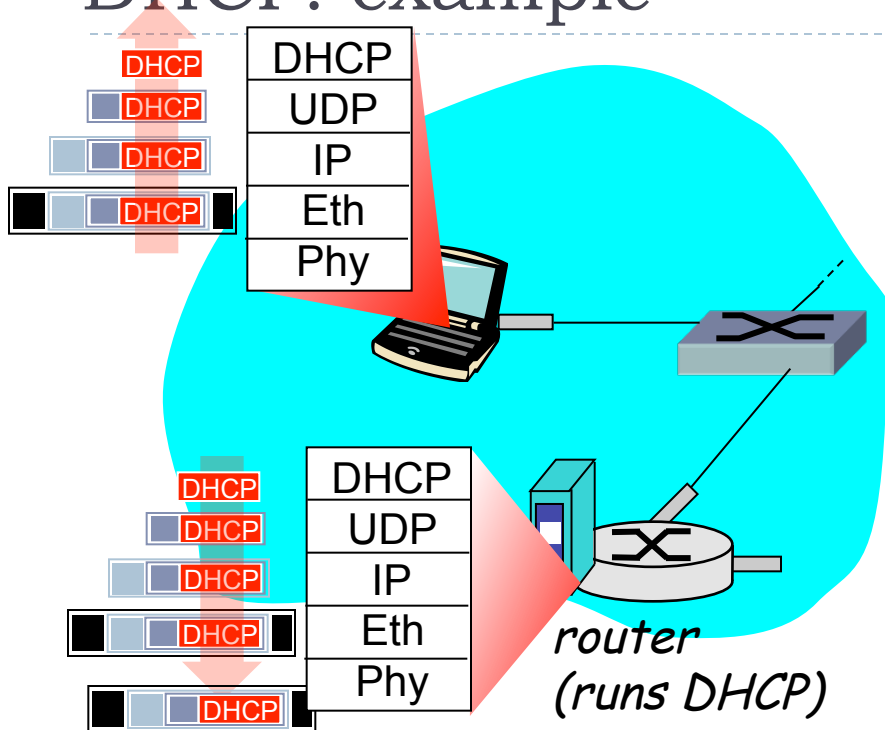
- ▶ address of first-hop router for client
- ▶ name and IP address of DNS sever
- ▶ network mask (indicating network versus host portion of address)

DHCP: example



- ▶ connecting laptop needs its IP address, addr of first-hop router, addr of DNS server: use DHCP
- DHCP request encapsulated in UDP, encapsulated in IP, encapsulated in 802.1 Ethernet
- Ethernet frame broadcast (dest: FFFFFFFFFFFFFFFF) on LAN, received at router running DHCP server
- Ethernet demux'ed to IP demux'ed, UDP demux'ed to DHCP

DHCP: example



- ▶ DCP server formulates DHCP ACK containing client's IP address, IP address of first-hop router for client, name & IP address of DNS server
- encapsulation of DHCP server, frame forwarded to client, demux'ing up to DHCP at client
- client now knows its IP address, name and IP address of DSN server, IP address of its first-hop router

DHCP: wireshark output (home LAN)

Message type: **Boot Request (1)**
Hardware type: Ethernet
Hardware address length: 6
Hops: 0
Transaction ID: 0x6b3a11b7
Seconds elapsed: 0
Bootp flags: 0x0000 (Unicast)
Client IP address: 0.0.0.0 (0.0.0.0)
Your (client) IP address: 0.0.0.0 (0.0.0.0)
Next server IP address: 0.0.0.0 (0.0.0.0)
Relay agent IP address: 0.0.0.0 (0.0.0.0)
Client MAC address: Wistron_23:68:8a (00:16:d3:23:68:8a)
Server host name not given
Boot file name not given
Magic cookie: (OK)
Option: (t=53,l=1) **DHCP Message Type = DHCP Request**
Option: (61) Client identifier
 Length: 7; Value: 010016D323688A;
 Hardware type: Ethernet
 Client MAC address: Wistron_23:68:8a (00:16:d3:23:68:8a)
Option: (t=50,l=4) Requested IP Address = 192.168.1.101
Option: (t=12,l=5) Host Name = "nomad"
Option: (55) Parameter Request List
 Length: 11; Value: 010F03062C2E2F1F21F92B
 1 = Subnet Mask; 15 = Domain Name
 3 = Router; 6 = Domain Name Server
 44 = NetBIOS over TCP/IP Name Server

request

Message type: **Boot Reply (2)**
Hardware type: Ethernet
Hardware address length: 6
Hops: 0
Transaction ID: 0x6b3a11b7
Seconds elapsed: 0
Bootp flags: 0x0000 (Unicast)
Client IP address: 192.168.1.101 (192.168.1.101)
Your (client) IP address: 0.0.0.0 (0.0.0.0)
Next server IP address: 192.168.1.1 (192.168.1.1)
Relay agent IP address: 0.0.0.0 (0.0.0.0)
Client MAC address: Wistron_23:68:8a (00:16:d3:23:68:8a)
Server host name not given
Boot file name not given
Magic cookie: (OK)
Option: (t=53,l=1) DHCP Message Type = DHCP ACK
Option: (t=54,l=4) Server Identifier = 192.168.1.1
Option: (t=1,l=4) Subnet Mask = 255.255.255.0
Option: (t=3,l=4) Router = 192.168.1.1
Option: (6) Domain Name Server
 Length: 12; Value: 445747E2445749F244574092;
 IP Address: 68.87.71.226;
 IP Address: 68.87.73.242;
 IP Address: 68.87.64.146
Option: (t=15,l=20) Domain Name = "hsd1.ma.comcast.net."

reply

Addressing and “Routing”

```
[nike]$ ifconfig
em1      Link encap:Ethernet  HWaddr 84:2B:2B:42:A5:7E
         inet addr:128.192.101.135  Bcast:128.192.101.191  Mask:255.255.255.192
         inet6 addr: fe80::862b:2bff:fe42:a57e/64  Scope:Link
         UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1

lo       Link encap:Local Loopback
         inet addr:127.0.0.1  Mask:255.0.0.0
         inet6 addr: ::1/128  Scope:Host
         UP LOOPBACK RUNNING  MTU:16436  Metric:1

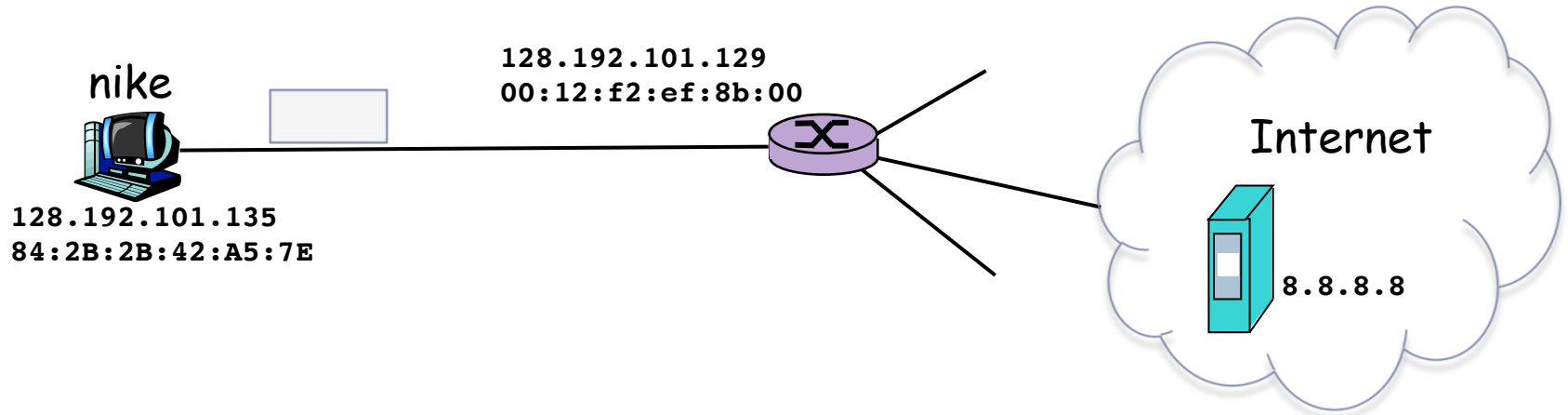
virbr0   Link encap:Ethernet  HWaddr 52:54:00:BC:51:80
         inet addr:192.168.122.1  Bcast:192.168.122.255  Mask:255.255.255.0
         UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
```

```
[nike]$ route -n
Kernel IP routing table
Destination      Gateway          Genmask          Flags Metric Ref    Use Iface
128.192.101.128  0.0.0.0         255.255.255.192 U        0      0      0 em1
192.168.122.0   0.0.0.0         255.255.255.0   U        0      0      0 virbr0
169.254.0.0     0.0.0.0         255.255.0.0     U       1002   0      0 em1
0.0.0.0         128.192.101.129 0.0.0.0         UG       0      0      0 em1
```

```
[nike]$ arp -n
Address          HWtype  HWaddress          Flags Mask          Iface
128.192.101.186 ether    c8:60:00:9b:5d:a3  C                em1
128.192.101.174 ether    00:26:b9:8e:10:56  C                em1
128.192.101.129 ether    00:12:f2:ef:8b:00  C                em1
```

```
[nike]$ cat /etc/resolv.conf
nameserver 128.192.1.9
nameserver 8.8.8.8
```

How packets get to the Internet



Src Eth: 84:2B:2B:42:A5:7E
Dst Eth: 00:12:f2:ef:8b:00

Src IP: 128.192.101.135
Dst IP: 8.8.8.8

Src Port: 54321
Dst Port: 53

www.microsoft.com ?

DNS

UDP

IPv4

Ethernet

IP addresses: how to get one?

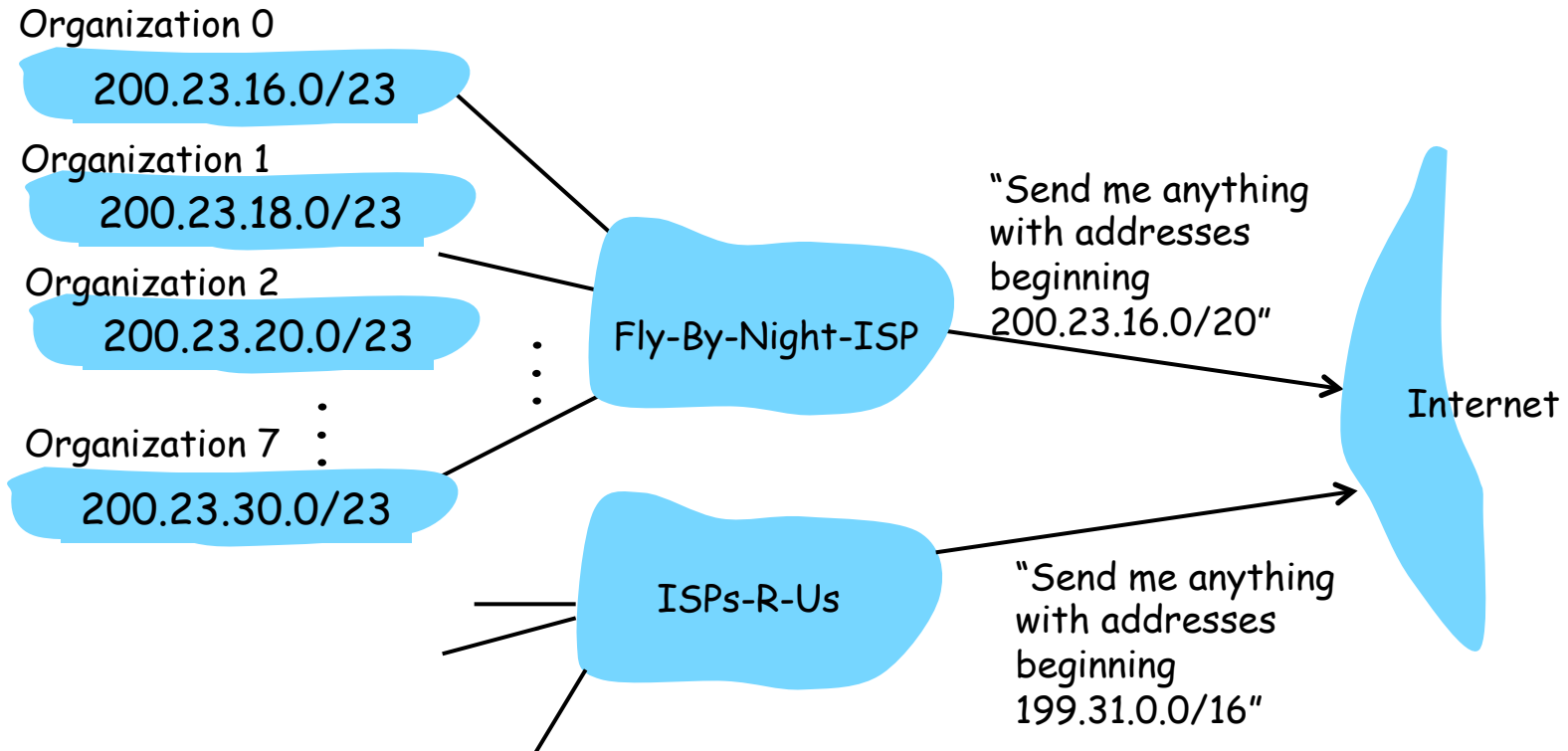
Q: How does a *network* get subnet part of IP addr?

A: gets allocated portion of its provider ISP's address space

ISP's block	<u>11001000</u>	<u>00010111</u>	<u>00010000</u>	00000000	200.23.16.0/20
Organization 0	<u>11001000</u>	<u>00010111</u>	<u>00010000</u>	00000000	200.23.16.0/23
Organization 1	<u>11001000</u>	<u>00010111</u>	<u>00010010</u>	00000000	200.23.18.0/23
Organization 2	<u>11001000</u>	<u>00010111</u>	<u>00010100</u>	00000000	200.23.20.0/23
...
Organization 7	<u>11001000</u>	<u>00010111</u>	<u>00011110</u>	00000000	200.23.30.0/23

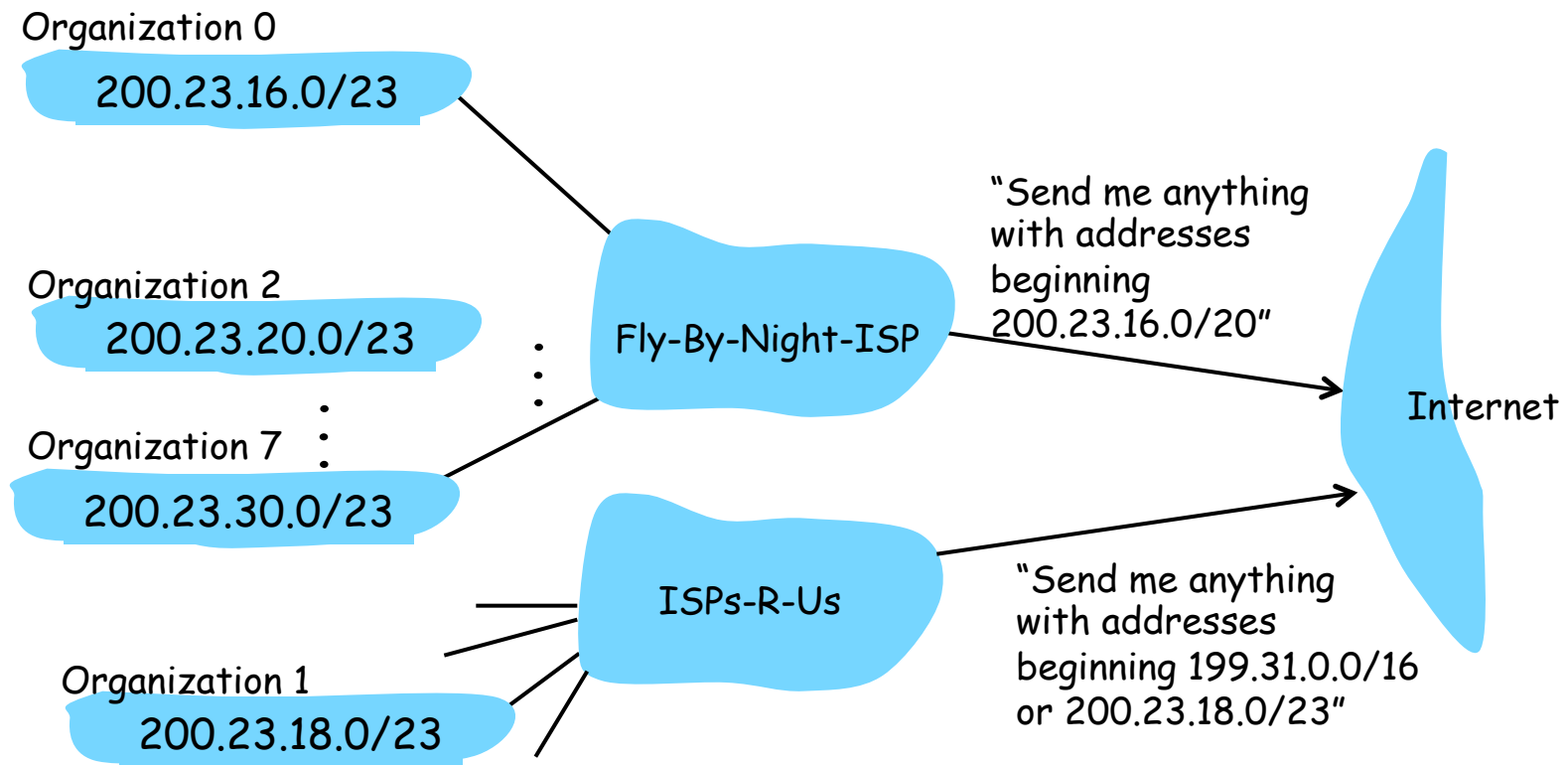
Hierarchical addressing: route aggregation

Hierarchical addressing allows efficient advertisement of routing information:



Hierarchical addressing: more specific routes

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



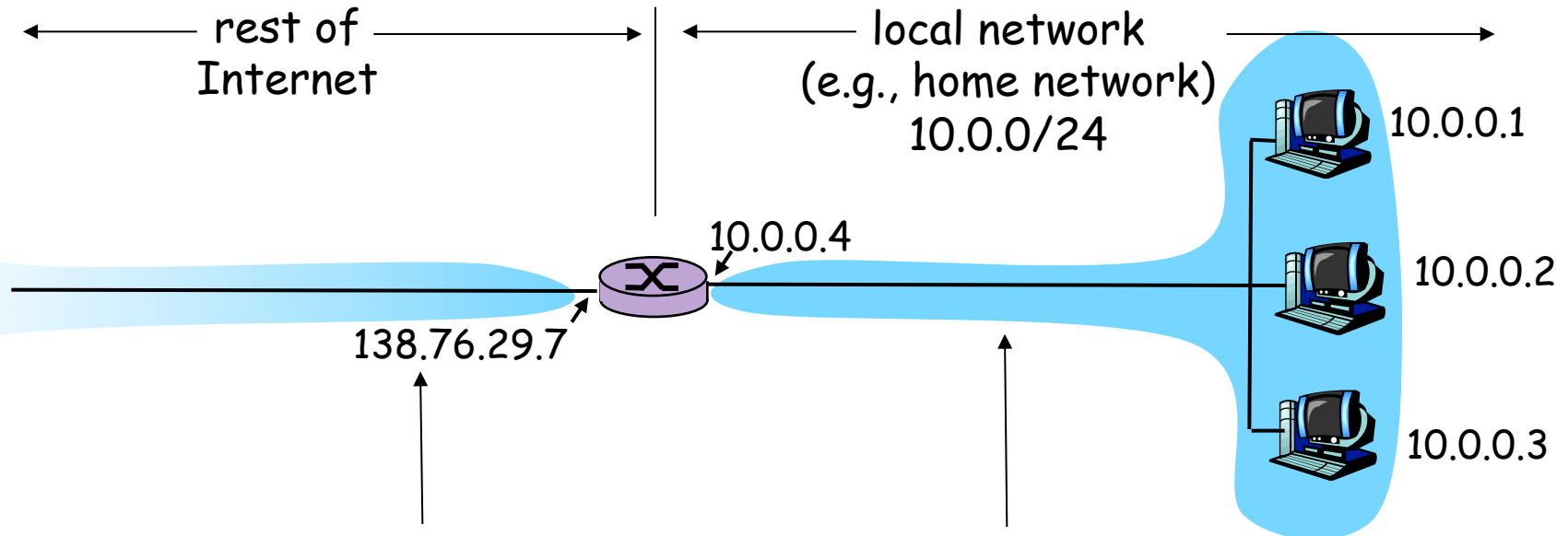
IP addressing: the last word...

Q: How does an ISP get an IP address block?

A: **ICANN**: Internet **C**orporation for **A**ssigned
Names and **N**umbers

- ▶ allocates addresses
- ▶ manages DNS
- ▶ assigns domain names, resolves disputes

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

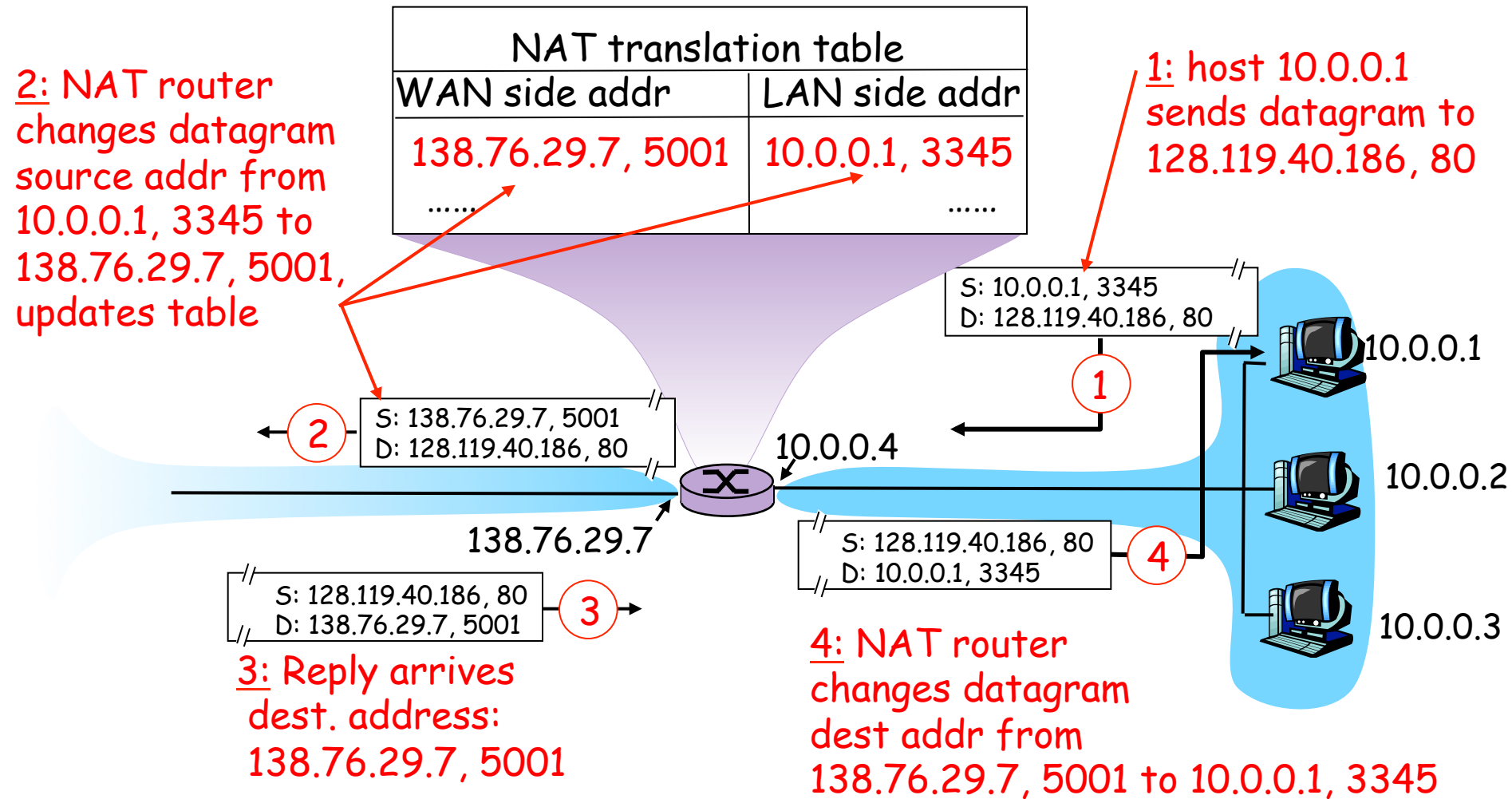
- ▶ **Motivation:** local network uses just one IP address as far as outside world is concerned:
 - ▶ range of addresses not needed from ISP: just one IP address for all devices
 - ▶ can change addresses of devices in local network without notifying outside world
 - ▶ can change ISP without changing 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

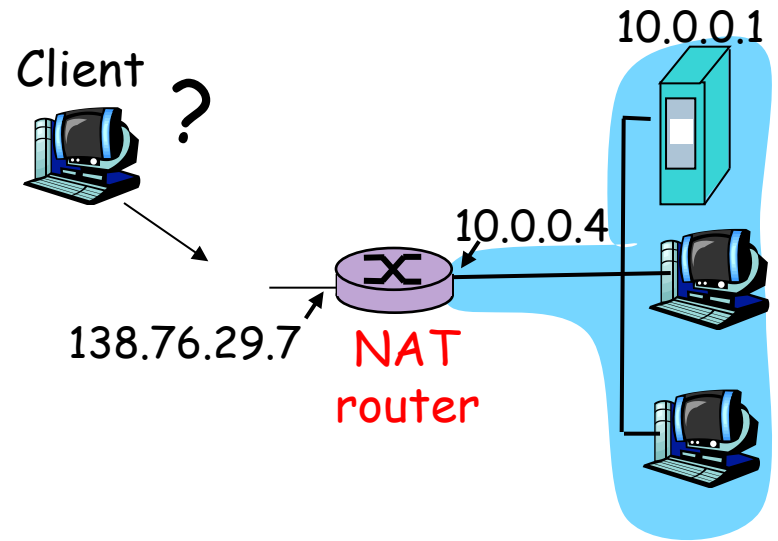


NAT: Network Address Translation

- ▶ 16-bit port-number field:
 - ▶ (65,536 – 1024) simultaneous connections with a single LAN-side address! (per transport layer protocol)
- ▶ 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, eg, P2P applications
 - ▶ address shortage should instead be solved by IPv6

NAT traversal problem

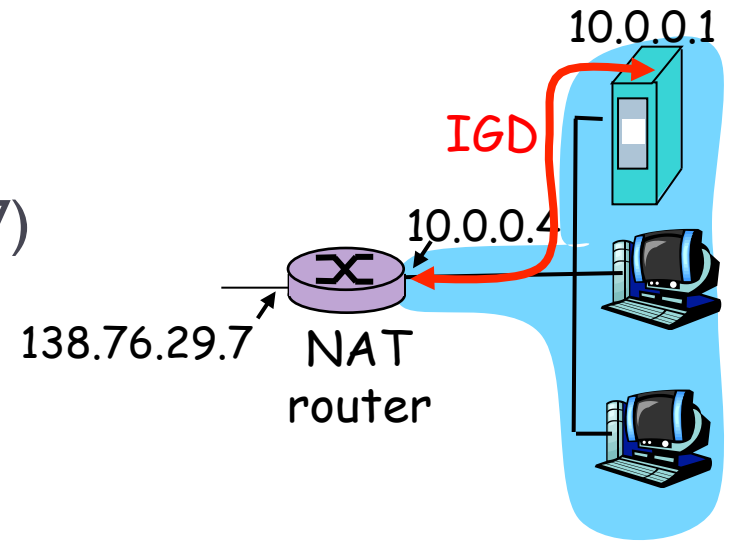
- ▶ client wants to connect to server with address 10.0.0.1
 - ▶ server address 10.0.0.1 local to LAN (client can't use it as destination addr)
 - ▶ only one externally visible NATted address: 138.76.29.7
- ▶ solution 1: statically configure NAT to forward incoming connection requests at given port to server
 - ▶ e.g., (138.76.29.7, port 2500) always forwarded to 10.0.0.1 port 25000



NAT traversal problem

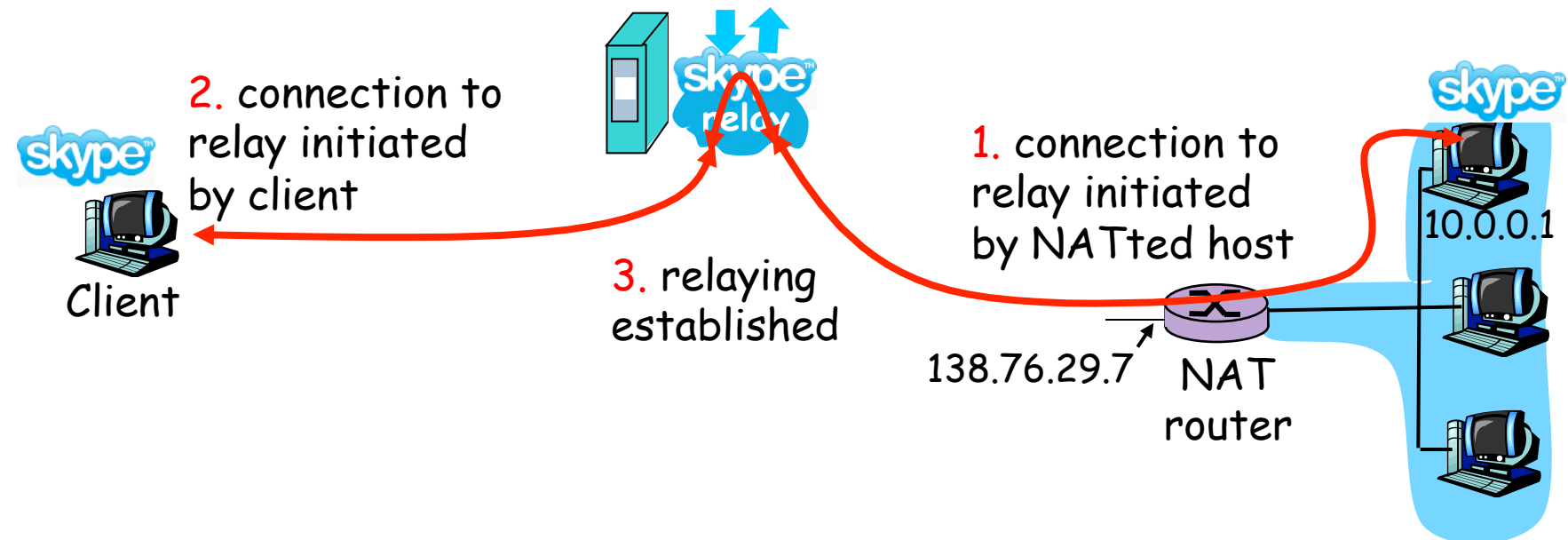
- ▶ solution 2: Universal Plug and Play (UPnP) Internet Gateway Device (IGD) Protocol. Allows NATted host to:
 - ❖ learn public IP address (138.76.29.7)
 - ❖ add/remove port mappings (with lease times)

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 two connections



ICMP:

Internet Control Message Protocol

- ▶ used by hosts & routers to communicate network-level information
 - ▶ error reporting: unreachable host, network, port, protocol
 - ▶ echo request/reply (used by ping)
- ▶ network-layer “above” IP:
 - ▶ ICMP msgs carried in IP datagrams
- ▶ **ICMP message**: type, code plus first 8 bytes of IP datagram causing error

<u>Type</u>	<u>Code</u>	<u>description</u>
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)
8	0	echo request (ping)
9	0	route advertisement
10	0	router discovery
11	0	TTL expired
12	0	bad IP header

Traceroute and ICMP

- ▶ Source sends series of UDP segments to dest
 - ▶ First has TTL = 1
 - ▶ Second has TTL=2, etc.
 - ▶ Unlikely port number
 - ▶ When nth datagram arrives to nth router:
 - ▶ Router discards datagram
 - ▶ And sends to source an ICMP message (type 11, code 0)
 - ▶ Message includes name of router & IP address
 - ▶ When ICMP message arrives, source calculates RTT
 - ▶ Traceroute does this 3 times
- Stopping criterion
- ▶ UDP segment eventually arrives at destination host
 - ▶ Destination returns ICMP “host unreachable” packet (type 3, code 3)
 - ▶ When source gets this ICMP, stops.

IPv6

- ▶ **Initial motivation:**
 - ▶ IPv4 offers only 32-bit addresses
 - ▶ space soon to be completely allocated
 - ▶ See IANA IPv4 Address Space Registry
- ▶ IPv6 => 128-bit addresses
 - ▶ $\sim 3.4E+38$, i.e., $\sim 6.7E+23$ addresses per m^2
- ▶ **Additional motivation:**
 - ▶ header format helps speed processing/forwarding
 - ▶ header changes to facilitate QoS
- ▶ **IPv6 datagram format:**
 - ▶ fixed-length 40 byte header
 - ▶ no fragmentation allowed (MTU discovery is used, instead)
 - ▶ *Fragmentation can happen only at the source, not at the routers*

IPv6 Addresses

- ▶ 16 bytes, hex format, ‘:’ separator
 - ▶ 2001:DB8:0000:0000:0202:B3FF:FE1E:8329
- ▶ Sequences of zeros can be abbreviated
 - ▶ 2001:DB8:0:0:0202:B3FF:FE1E:8329
 - ▶ 2001:DB8::0202:B3FF:FE1E:8329
- ▶ Prefix notation
 - ▶ Similar to CIDR notation for IPv4
 - ▶ 2E78:DA53:1200::/40

IPv6 Addresses

Allocation	Prefix binary	Prefix hex	Fraction of address space
Unassigned	0000 0000	::0/8	1/256
Reserved	0000 001		1/128
Global unicast	001	2000::/3	1/8
Link-local unicast	1111 1110 10	FE80::/10	1/1024
Reserved (formerly Site-local unicast)	1111 1110 11	FECO::/10* * deprecated	1/1024
Local IPv6 address	1111 110	FC00::/7	
Private administration	1111 1101	FD00::/8	
Multicast	1111 1111	FF00::/8	1/256

IPv6 loopback address = ::1/128
(IPv4 loopback address = 127.0.0.0/8)

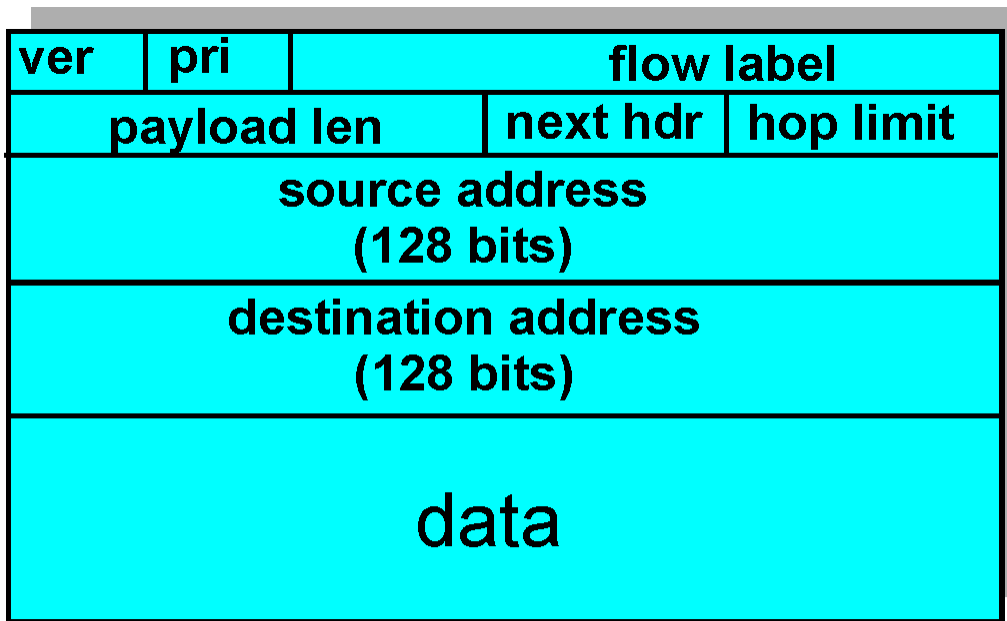
IPv6 Header ***

Priority: identify priority among datagrams in flow

Flow Label: identify datagrams in same "flow."

(concept of "flow" not well defined).

Next header: identify upper layer protocol for data



4-50 32 bits Network Layer

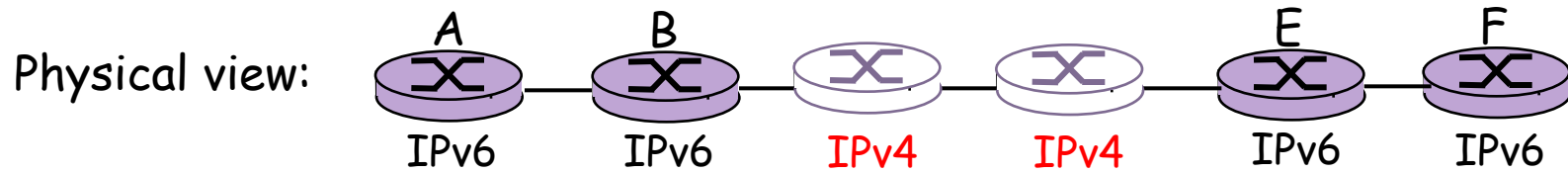
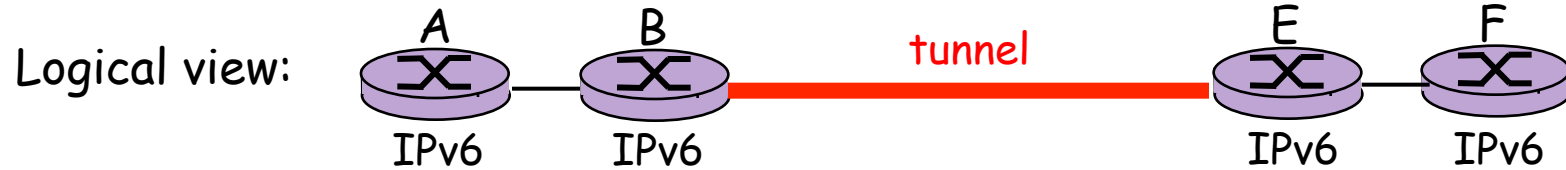
Other Changes from IPv4

- ▶ *Checksum*: removed entirely... **Why???**
- ▶ *Options*: allowed, but outside of header, indicated by “Next Header” field
- ▶ *ICMPv6*: new version of ICMP
 - ▶ additional message types, e.g. “Packet Too Big”
 - ▶ multicast group management functions

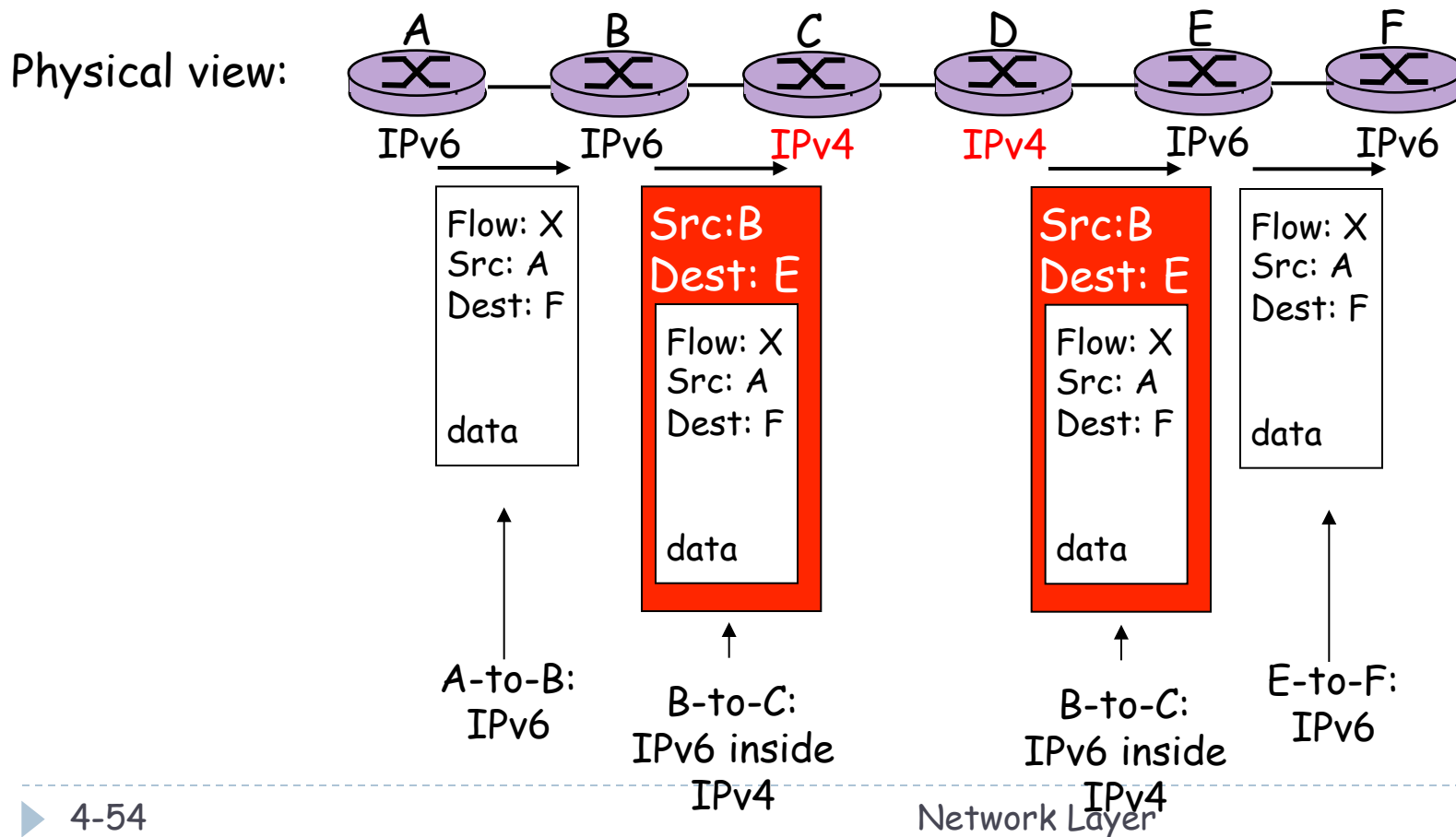
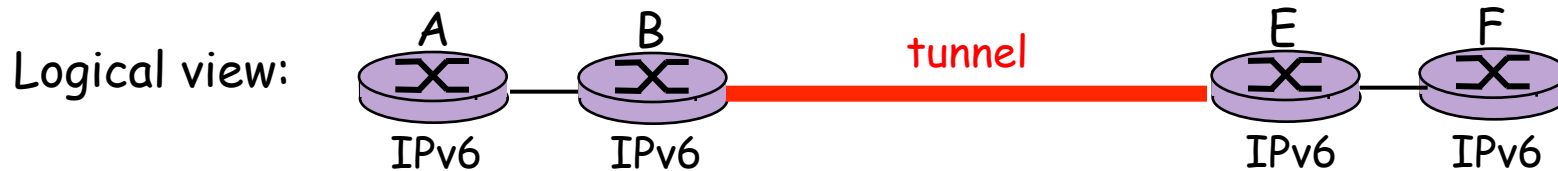
Transition From IPv4 To IPv6

- ▶ Not all routers can be upgraded simultaneous
 - ▶ no “flag days”
 - ▶ How will the network operate with mixed IPv4 and IPv6 routers?
- ▶ **Dual Stack + AAAA DNS RRs**
 - ▶ How does it work? (assuming core supports IPv6)
- ▶ *Tunneling*: IPv6 carried as payload in IPv4 datagram among IPv4 routers

Tunneling



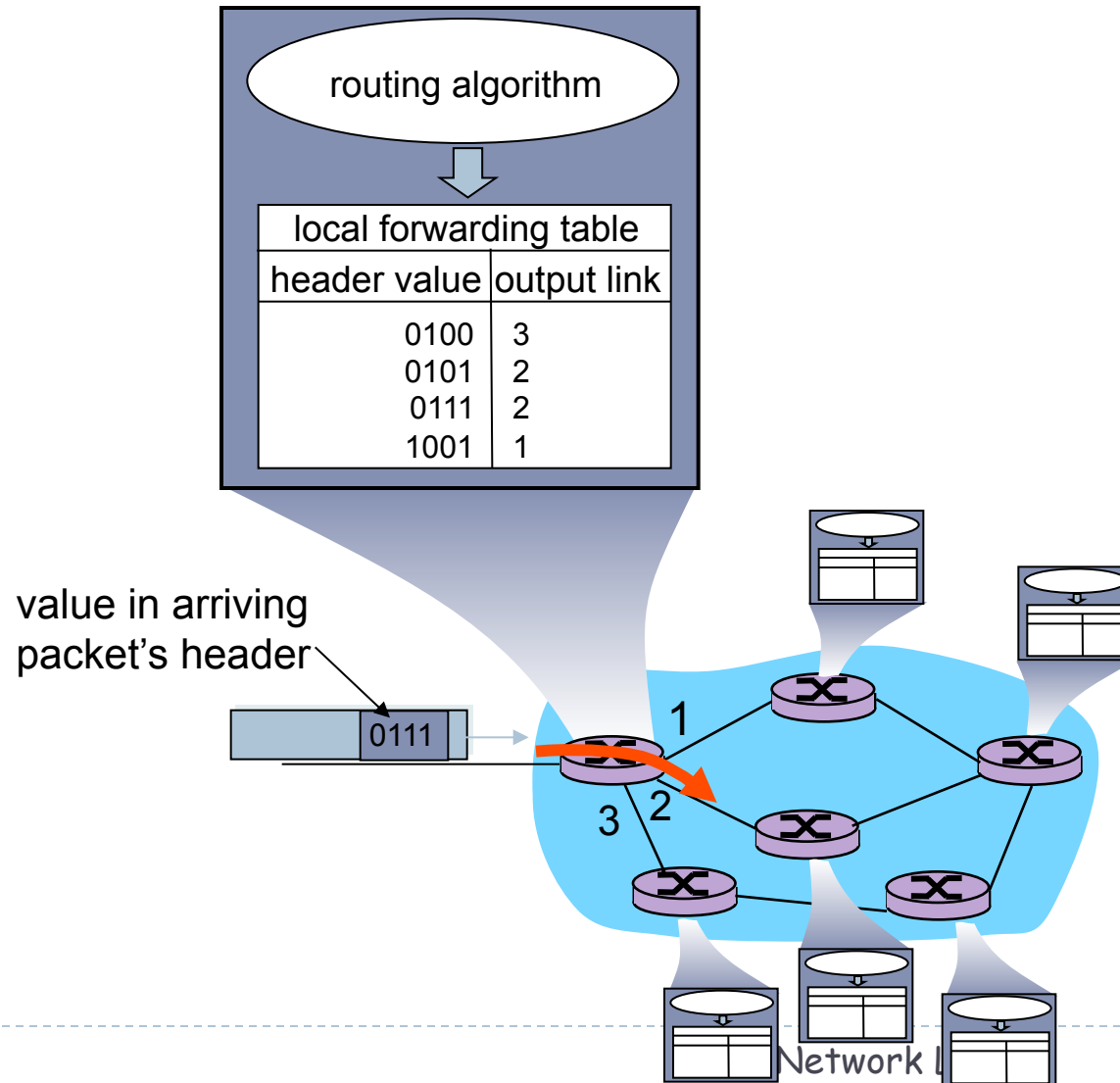
Tunneling



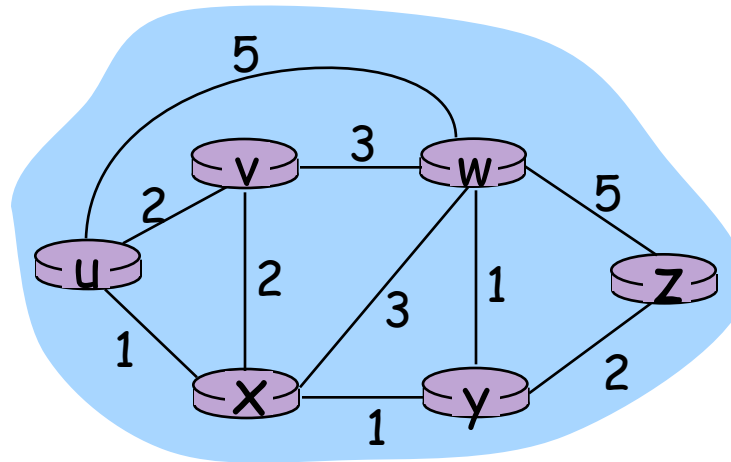
IPsec

- ▶ In between network and transport layer
- ▶ “Connection oriented”
- ▶ Functionalities
 - ▶ Cryptographic agreement (key exchange)
 - ▶ Encryption of payload
 - ▶ Data integrity
 - ▶ Origin authentication (no spoofing!)

Interplay between routing, forwarding



Graph abstraction



Graph: $G = (N,E)$

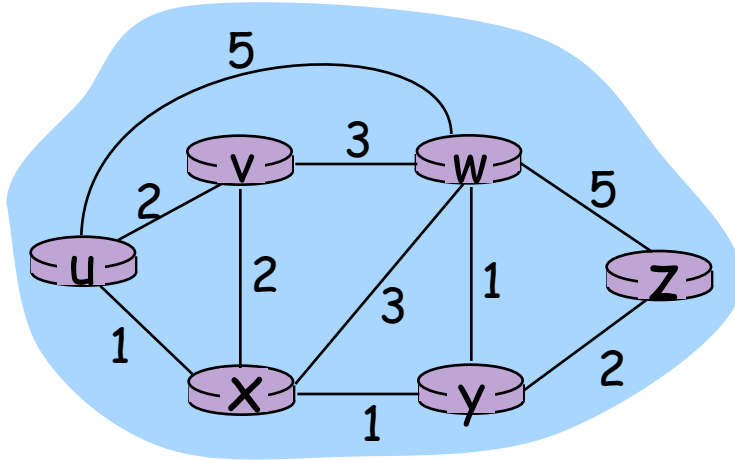
$N =$ set of routers = $\{ u, v, w, x, y, z \}$

$E =$ set of links = $\{ (u,v), (u,x), (v,x), (v,w), (x,w), (x,y), (w,y), (w,z), (y,z) \}$

Remark: Graph abstraction is useful in other network contexts

Example: P2P, where N is set of peers and E is set of TCP connections

Graph abstraction: costs



• $c(x,x') = \text{cost of link } (x,x')$

- e.g., $c(w,z) = 5$

• cost related to physical features of link or congestion or money

Cost of path $(x_1, x_2, x_3, \dots, x_p) = c(x_1, x_2) + c(x_2, x_3) + \dots + c(x_{p-1}, x_p)$

Question: What's the least-cost path between u and z ?

Routing algorithm: algorithm that finds least-cost path

Routing Algorithm classification

Global or decentralized information?

Global:

- ▶ all routers have complete topology, link cost info
- ▶ “link state” algorithms

Decentralized:

- ▶ router knows physically-connected neighbors, link costs to neighbors
- ▶ iterative process of computation, exchange of info with neighbors
- ▶ “distance vector” algorithms

Static or dynamic?

Static:

- ▶ routes change slowly over time

Dynamic:

- ▶ routes change more quickly
 - ▶ periodic update
 - ▶ in response to link cost changes (e.g., topology)
- ▶ Load Sensitive?

A Link-State Routing Algorithm

Dijkstra's algorithm

- ▶ net topology, link costs known to all nodes
 - ▶ accomplished via “link state broadcast”
 - ▶ all nodes have same info
- ▶ computes least cost paths from one node (“source”) to all other nodes
 - ▶ gives forwarding table for that node
- ▶ iterative: after k iterations, know least cost path to k dest.'s

Notation:

- ▶ $c(a,b)$: link cost from node a to b ; $= \infty$ if not direct neighbors
- ▶ $D(f)$: current value of cost of path from source to dest. f
- ▶ $P(f)$: predecessor node along path from source to f
- ▶ N' : set of nodes whose least cost path definitively known

Dijkstra's Algorithm

1 **Initialization:**

2 $N' = \{A\}$

3 for all nodes B

4 If B adjacent to A

5 then $D(B) = c(A,B)$

6 else $D(B) = \infty$

7

8 **Loop**

9 Find C not in N' such that $D(C)$ is a minimum

10 add C to N'

11 update $D(F)$ for all F adjacent to C and not in N' :

12 $D(F) = \min(D(F), D(C) + c(C,F))$

13 /* new cost to F is either old cost to F or known

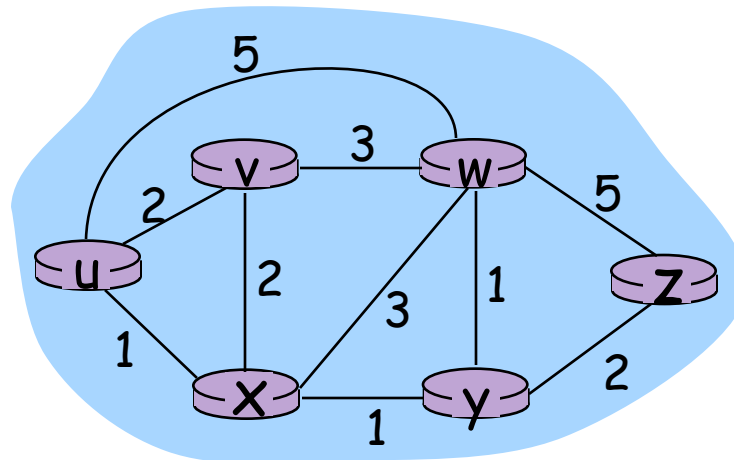
14 shortest path cost to C plus cost from C to F */

15 **until all nodes in N'**



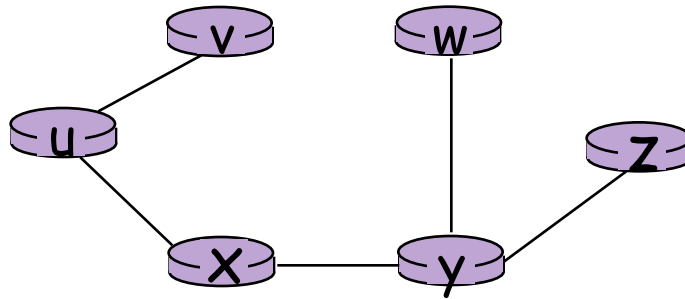
Dijkstra's algorithm: example

Step	N'	D(v),p(v)	D(w),p(w)	D(x),p(x)	D(y),p(y)	D(z),p(z)
0	u	2,u	5,u	1,u	∞	∞
1	ux	2,u	4,x		2,x	∞
2	uxy	2,u	3,y			4,y
3	uxyv		3,y			4,y
4	uxyvw					4,y
5	uxyvwz					



Dijkstra's algorithm: example (2)

Resulting shortest-path tree from u:



Resulting forwarding table in u:

destination	link
v	(u,v)
x	(u,x)
y	(u,x)
w	(u,x)
z	(u,x)

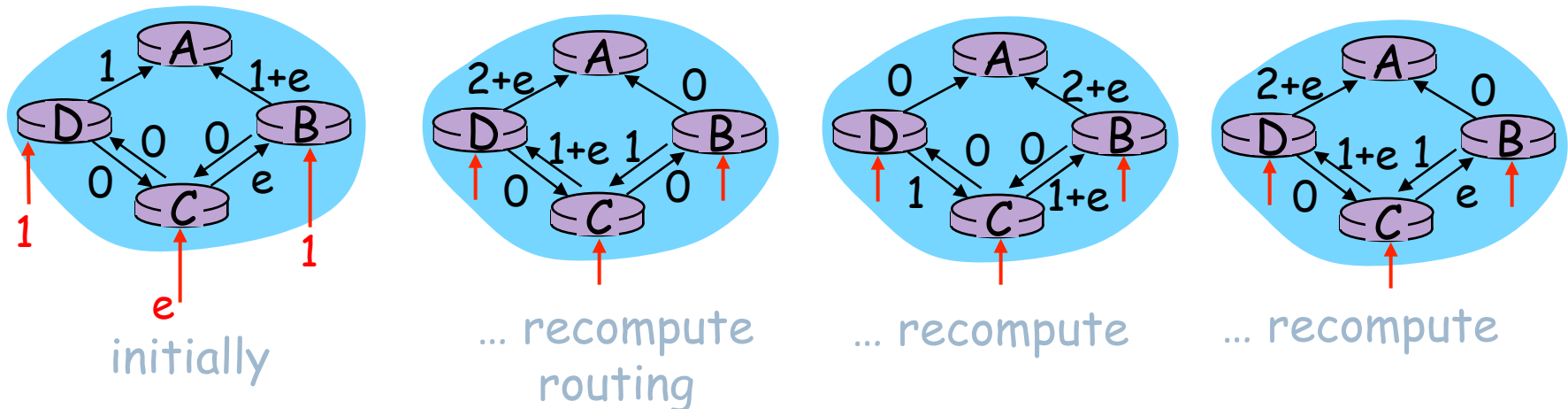
Dijkstra's algorithm, discussion

Algorithm complexity: n nodes

- ▶ each iteration: need to check all nodes, w , not in N
- ▶ $n(n+1)/2$ comparisons: $O(n^2)$
- ▶ more efficient implementations possible: $O(n \log n)$

Oscillations possible (in case of dynamic link update):

- ▶ e.g., link cost = amount of carried traffic (asymmetric)



Routing Algorithm classification

Global or decentralized information?

Global:

- ▶ all routers have complete topology, link cost info
- ▶ “link state” algorithms

Decentralized:

- ▶ router knows physically-connected neighbors, link costs to neighbors
- ▶ iterative process of computation, exchange of info with neighbors
- ▶ “distance vector” algorithms



Distance Vector Algorithm

Bellman-Ford Equation (dynamic programming)

Define

$d_a(b) :=$ cost of least-cost path from a to b

Then

$$d_a(b) = \min_e \{ c(a,e) + d_e(b) \}$$

where *min* is taken over all neighbors e of a

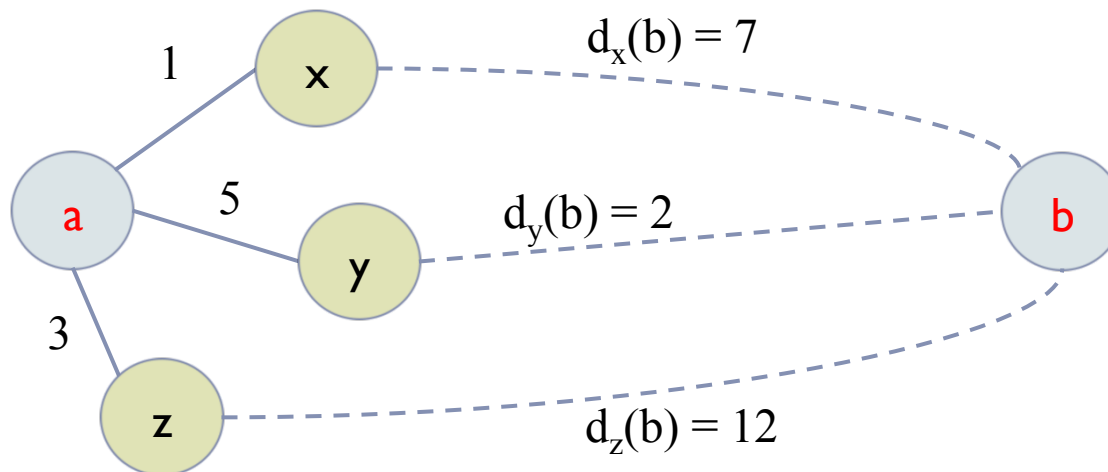
Distance Vector Algorithm

Bellman-Ford

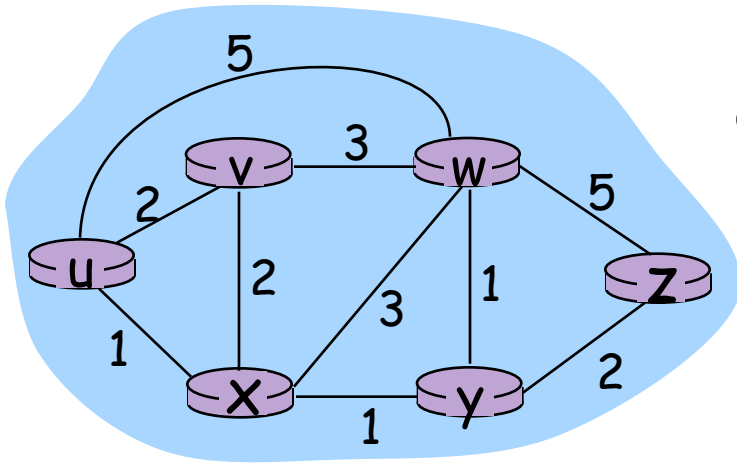
Definition:

$d_a(b) :=$ cost of least-cost path from a to b

$$d_a(b) = \min_e \{ c(a,e) + d_e(b) \}$$



Bellman-Ford example



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

B-F equation says:

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

Node that achieves minimum is next hop in shortest path → forwarding table

Distance Vector Algorithm

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

Distance vector algorithm (4)

Basic idea:

- ▶ From time-to-time, each node sends its own distance vector estimate to neighbors
- ▶ Asynchronous
- ▶ When a node 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 (5)

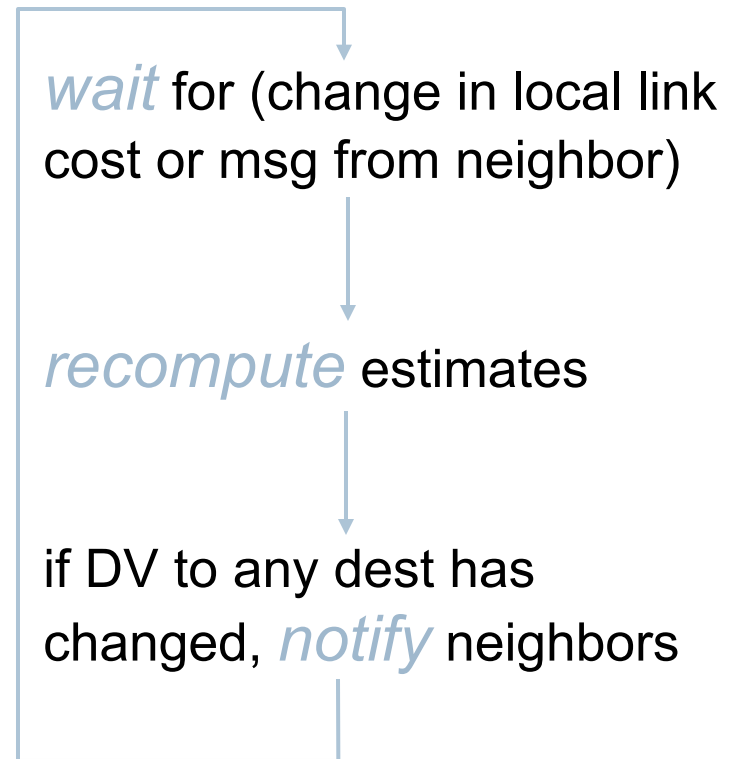
Iterative, asynchronous: each local iteration caused by:

- ▶ 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:



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

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

node x table

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

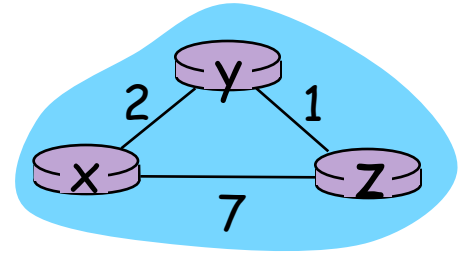
		cost to		
		x	y	z
from	x	0	2	3
	y	2	0	1
	z	7	1	0

node y table

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

node z table

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



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

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

node x table

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

node y table

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

node z table

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

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

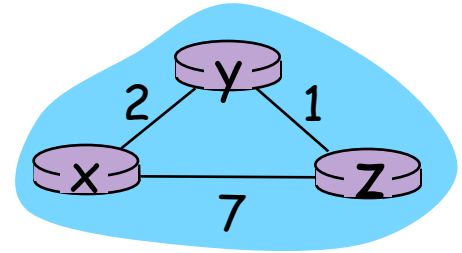
		cost to		
		x	y	z
from	x	0	2	7
	y	2	0	1
	z	7	1	0

		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

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

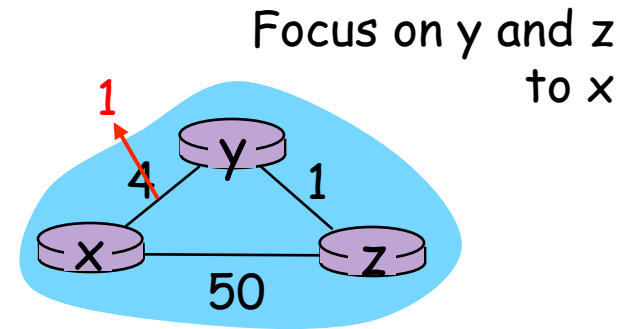
		cost to		
		x	y	z
from	x	0	2	3
	y	2	0	1
	z	3	1	0



Distance Vector: link cost changes

Link cost changes:

- ❑ node detects local link cost change
- ❑ updates routing info, recalculates distance vector
- ❑ if DV changes, notify neighbors



At time t_0 , y detects the link-cost change, updates its DV, and informs its neighbors.

At time t_1 , z receives the update from y and updates its table. It computes a new least cost to x and sends its neighbors its DV.

At time t_2 , y receives z's update and updates its distance table. y's least costs do not change and hence y does *not* send any message to z.

“good
news
travels
fast”

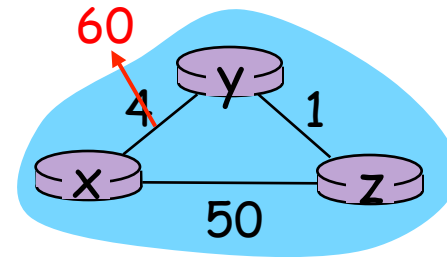
Distance Vector: link cost changes

Link cost changes:

- ❑ good news travels fast
- ❑ bad news travels slow - "count to infinity" problem!
- ❑ 44 iterations before algorithm stabilizes: see text

Poisoned 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)
- ❑ will this completely solve count to infinity problem?



What if $c(x,y)=10,000$
and $c(x,z)=9,999$?

"Count to infinity" problem

Comparison of LS and DV algorithms

Message complexity

- ▶ LS: with n nodes, E links, $O(nE)$ msgs sent
- ▶ DV: exchange between neighbors only
 - ▶ convergence time varies

Speed of Convergence

- ▶ LS: $O(n^2)$ algorithm requires $O(nE)$ msgs
 - ▶ may have oscillations
- ▶ 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
 - ▶ errors propagate thru network

Hierarchical Routing

Our routing study thus far - idealization

- ❑ all routers identical

- ❑ network "flat"

... *not* true in practice

scale: with 200 million

destinations:

- ▶ can't store all dest's in routing tables!
- ▶ routing table exchange would swamp links!

administrative autonomy

- ▶ internet = network of networks
- ▶ each network admin may want to control routing in its own network

Hierarchical Routing

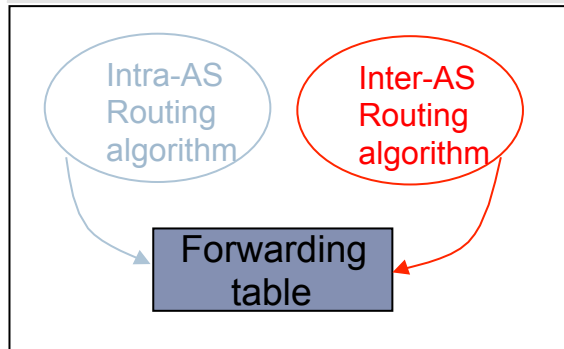
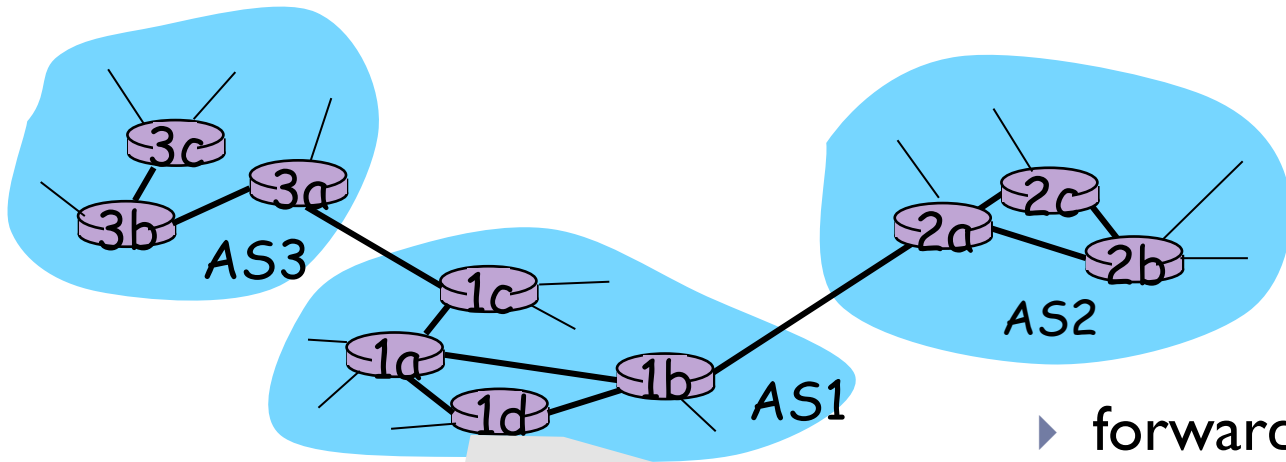
- ▶ aggregate routers into regions, “**autonomous systems**” (AS)
- ▶ routers in same AS run same routing protocol
 - ▶ “**intra-AS**” routing protocol
 - ▶ routers in different AS can run different intra-AS routing protocol

Gateway router

- ▶ Direct link to router in another AS

AS = group of routers typically under the same administrative control (e.g., same ISP)

Interconnected ASes



- ▶ forwarding table configured by both intra- and inter-AS routing algorithm
- ▶ intra-AS sets entries for internal dests
- ▶ inter-AS & intra-As sets entries for external dests

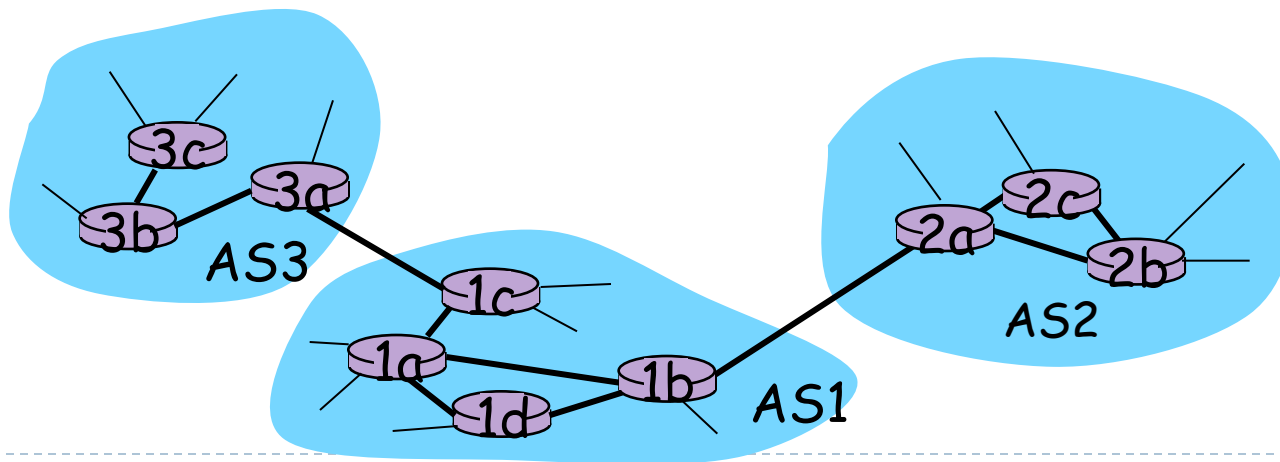
Inter-AS tasks

- ▶ suppose router in AS1 receives datagram destined outside of AS1:
 - ▶ router should forward packet to gateway router, but which one?

AS1 must:

1. learn which destds are reachable through AS2, which through AS3
2. propagate this reachability info to all routers in AS1

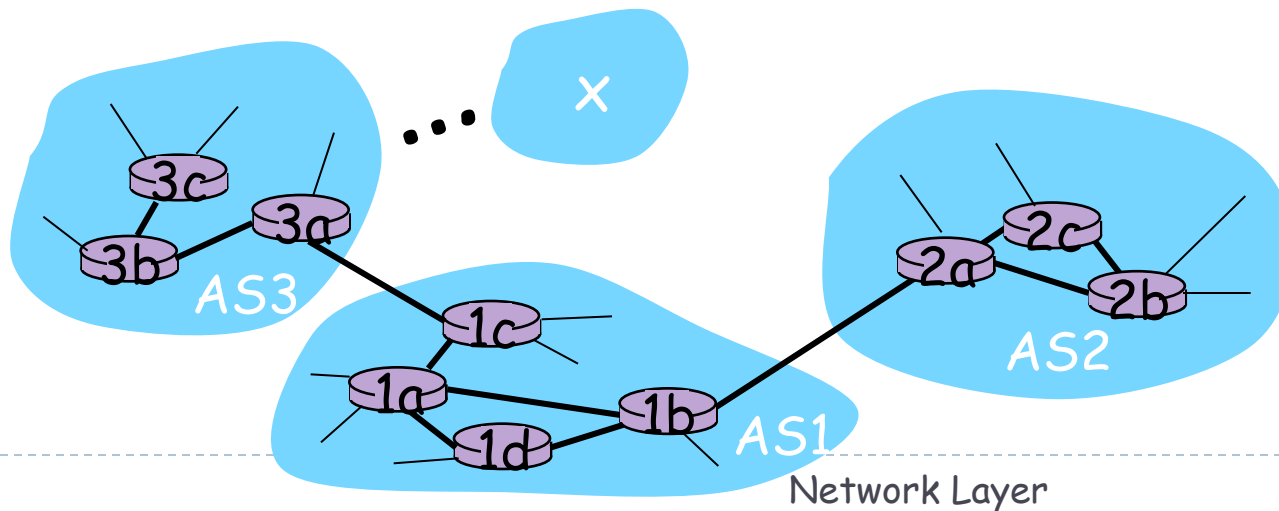
Job of inter-AS routing!



Example:

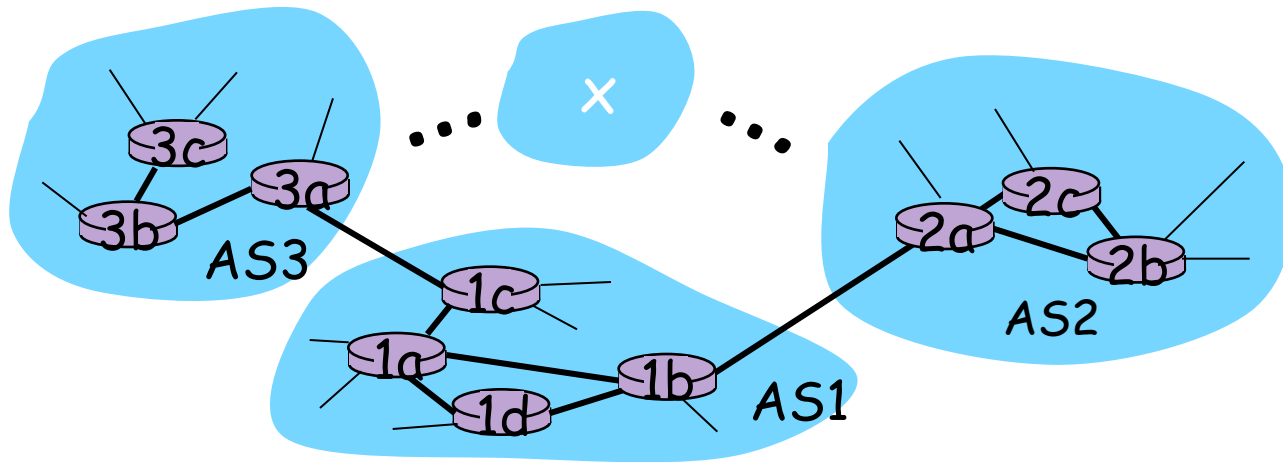
Setting forwarding table in router 1d

- ▶ suppose AS1 learns (via inter-AS protocol) that subnet **x** reachable via AS3 (gateway 1c) but not via AS2.
- ▶ inter-AS protocol propagates reachability info to all internal routers.
- ▶ router 1d determines from intra-AS routing info that its interface **l** is on the least cost path to 1c.
- ▶ installs forwarding table entry **(x,l)**



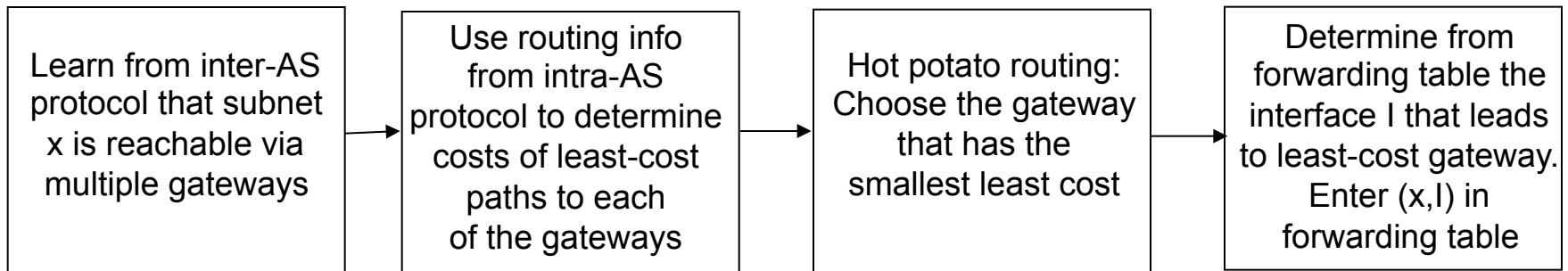
Example: Choosing among multiple ASes

- ▶ now suppose AS1 learns from inter-AS protocol that subnet **x** is reachable from AS3 *and* from AS2.
- ▶ to configure forwarding table, router 1d must determine towards which gateway it should forward packets for dest **x**.
 - ▶ this is also job of inter-AS routing protocol!



Example: Choosing among multiple ASes

- ▶ now suppose AS1 learns from inter-AS protocol that subnet **x** is reachable from AS3 *and* from AS2.
- ▶ to configure forwarding table, router Id must determine towards which gateway it should forward packets for dest **x**.
 - ▶ this is also job of inter-AS routing protocol!
- ▶ **hot potato routing**: send packet towards closest of two routers.



Internet inter-AS routing: BGP

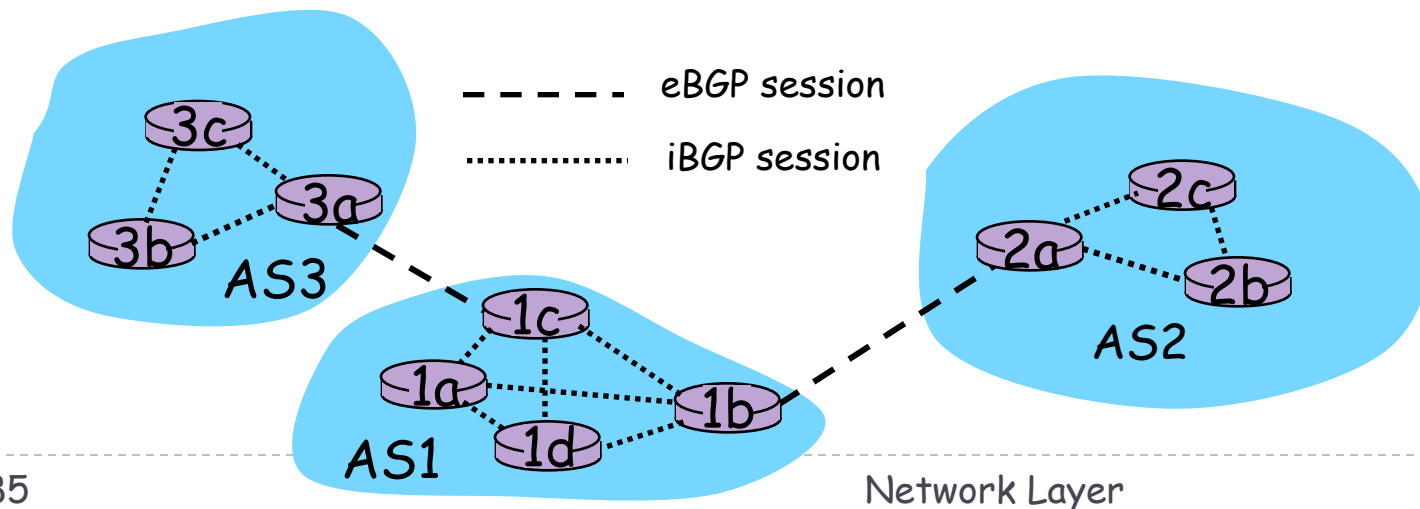
- ▶ **BGPv4 (Border Gateway Protocol):** *the de facto standard*
- ▶ BGP provides each AS a means to:
 1. Obtain subnet reachability information from neighboring ASs.
 2. Propagate reachability information to all AS-internal routers.
 3. Determine “good” routes to subnets based on reachability information and policy.
- ▶ allows subnet to advertise its existence to rest of Internet: *“I am here”*
- ▶ *BGP glues the Internet “sub-networks” together!*

- ▶ AS are assigned an AS number
 - ▶ Similar to IPs, assigned by ICANN regional registrars

BGP basics

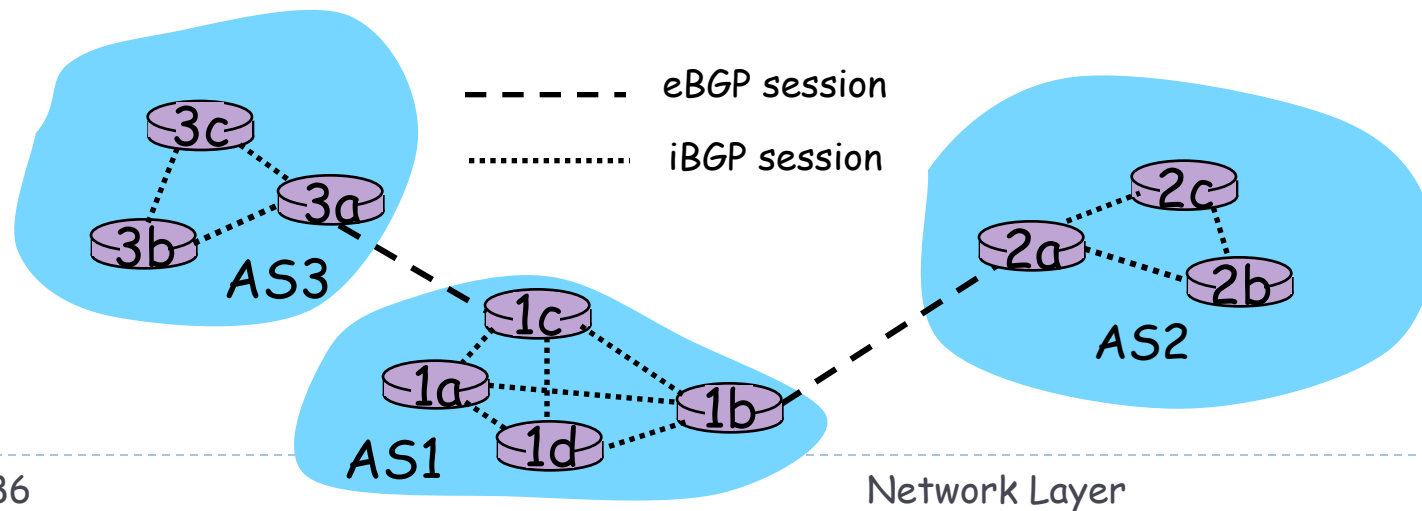
- ▶ pairs of routers (BGP peers) exchange routing info over TCP connections: **BGP sessions**
 - ▶ BGP sessions need not correspond to physical links.
- ▶ when AS2 advertises a prefix to AS1:
 - ▶ AS2 *promises* it will forward datagrams towards that prefix.
 - ▶ AS2 can aggregate prefixes in its advertisement

CIDR announcements



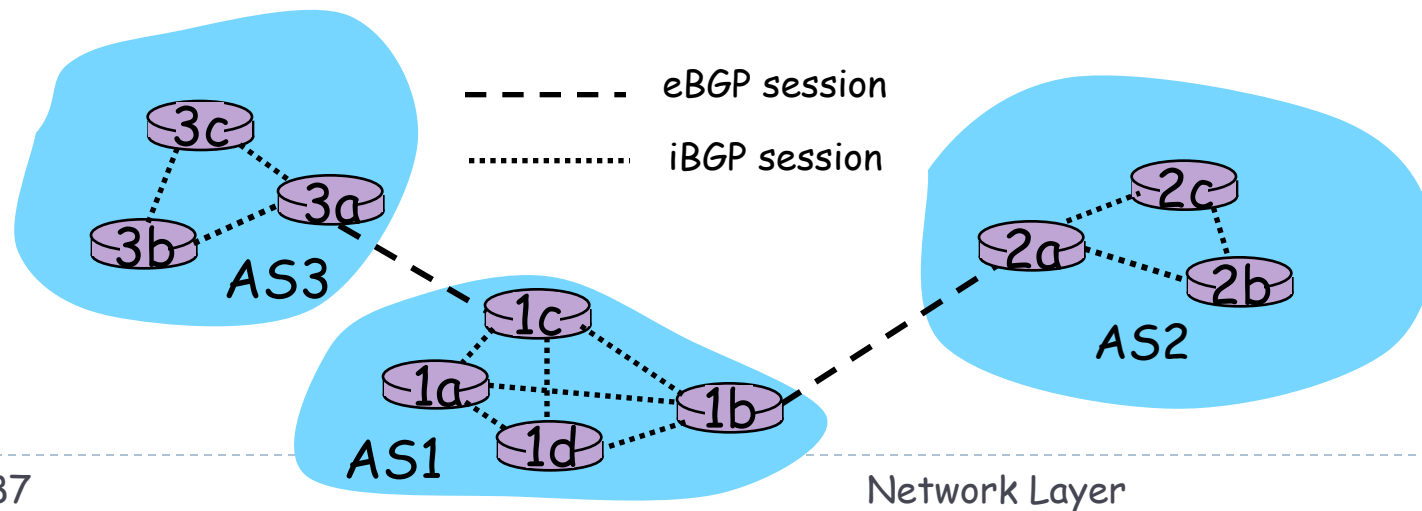
Distributing reachability info

- ▶ using eBGP session between 3a and 1c, AS3 sends prefix reachability info to AS1.
 - ▶ 1c can then use iBGP to distribute new prefix info to all routers in AS1
 - ▶ 1b can then re-advertise new reachability info to AS2 over 1b-to-2a eBGP session
- ▶ when router learns of new prefix, it creates entry for prefix in its forwarding table.



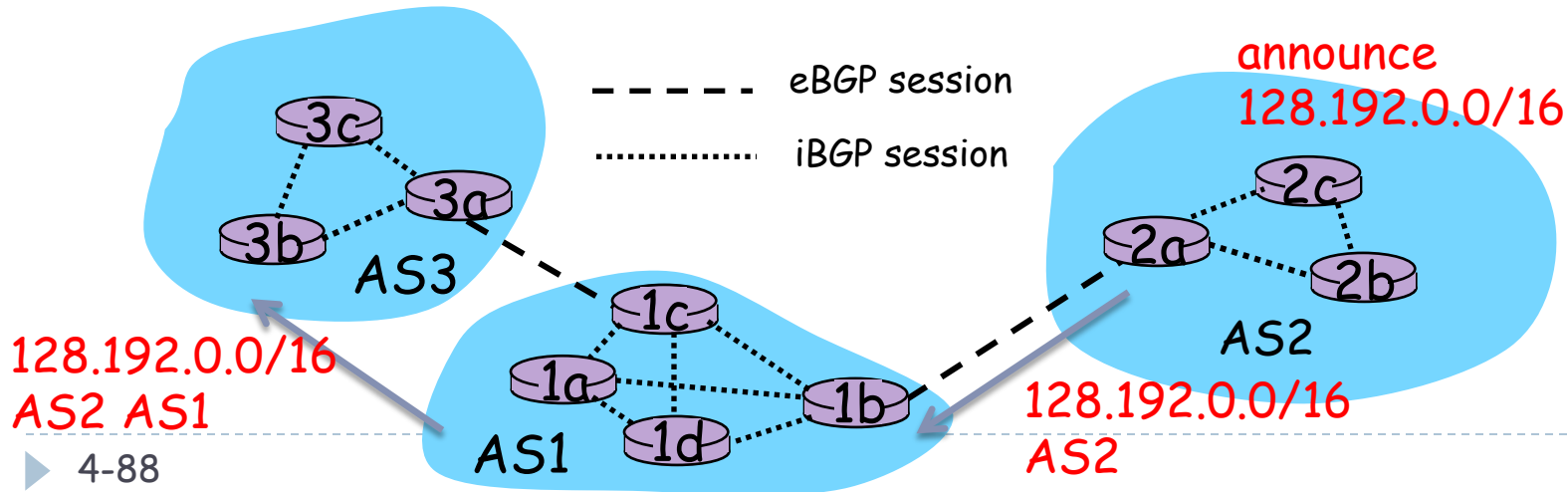
Distributing reachability info

- ▶ Assume there are 4 subnets attached to AS2
 - ▶ 138.16.64.0/24
 - ▶ 138.16.65.0/24
 - ▶ 138.16.66.0/24
 - ▶ 138.16.67.0/24
- ▶ AS2 could announce 138.16.64.0/22



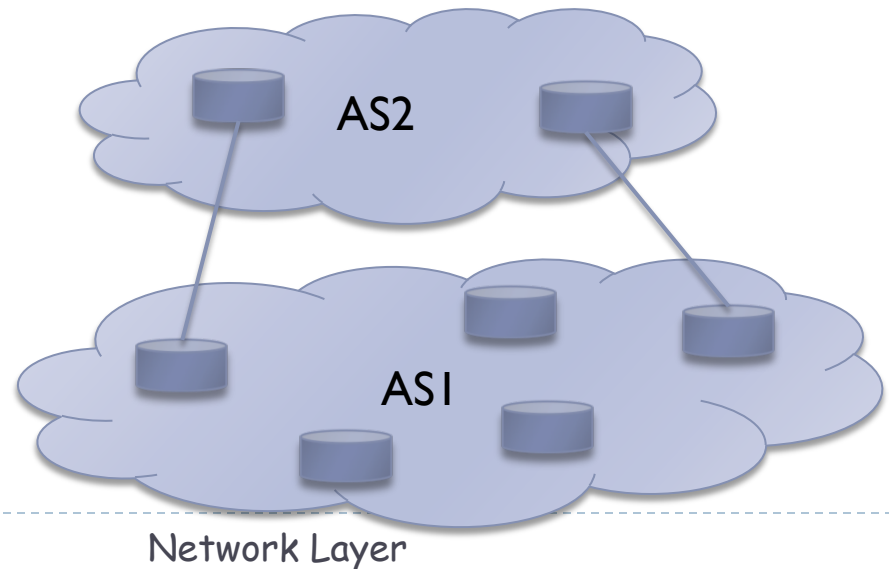
Path attributes & BGP routes

- ▶ advertised prefix includes BGP attributes.
 - ▶ prefix + attributes = “route”
- ▶ two important attributes:
 - ▶ **AS-PATH**: contains ASs through which prefix advertisement has passed: e.g, AS 67, AS 17 (avoids loops by checking AS list)
 - ▶ **NEXT-HOP**: interface that begins AS-PATH. Indicates specific router interface through which packets can be routed (e.g, 1d learns the IP of the interface of 2a and identifies its forwarding interface)
- ▶ **import policy** to accept/decline announcements



BGP route selection

- ▶ router may learn more than 1 route to same prefix.
 - ▶ must select one
- ▶ *Simplified* elimination rules (followed in order):
 1. local preference value attribute: policy decision
 2. shortest AS-PATH (similar to DV algorithm based on AS hops)
 3. closest NEXT-HOP router: hot potato routing
 4. additional criteria

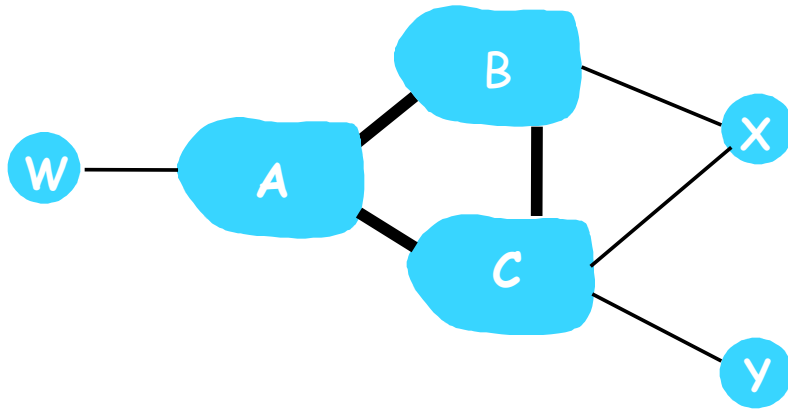




BGP messages

- ▶ BGP messages exchanged using TCP.
- ▶ BGP messages:
 - ▶ **OPEN**: opens TCP connection to peer and authenticates sender
 - ▶ **UPDATE**: advertises new path (or withdraws old)
 - ▶ **KEEPALIVE** keeps connection alive in absence of UPDATES; also ACKs OPEN request
 - ▶ **NOTIFICATION**: reports errors in previous msg; also used to close connection

BGP routing policy

Example: 6 ASs

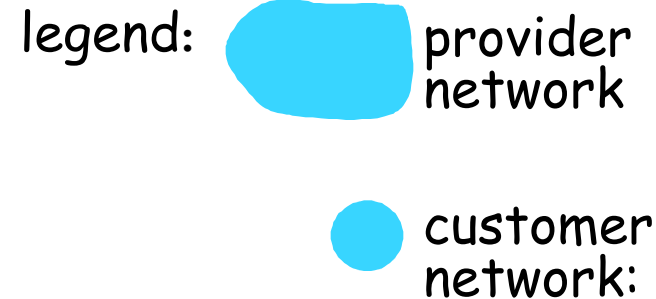
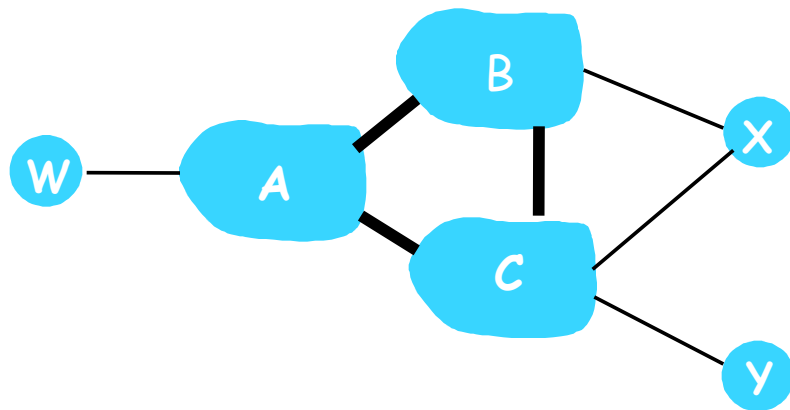


legend:  provider network
 customer network:

X will declare it knows of no paths to destinations outside itself!!!

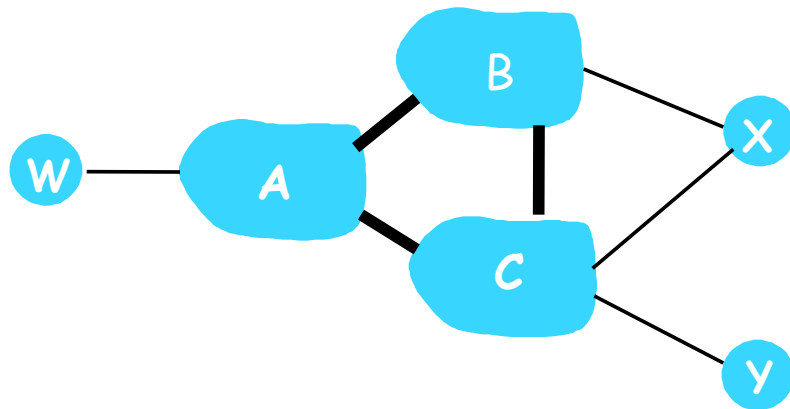
- ❑ A,B,C are **provider networks**
- ❑ X,W,Y are customer of provider networks (stub AS)
 - ❑ All traffic entering/leaving a stub AS must be destined/origin to/from that AS
- ❑ X is **dual-homed**: attached to two networks
 - X does not want to route from B via X to C
 - .. so X will not advertise to B a route to C

BGP routing policy (2)



- A advertises path AW to B
- B advertises path BAW to X
- Should B advertise path BAW to C?

BGP routing policy



legend:  provider network

 customer network:

For ISPs, traffic flowing through it should originate or be destined to a customer

- ❑ A advertises path AW to B
- ❑ B advertises path BAW to X
- ❑ Should B advertise path BAW to C?
 - No way! B gets no "revenue" for routing CBAW since neither W nor C are B's customers
 - B wants to force C to route to w via A
 - B wants to route *only* to/from its customers!

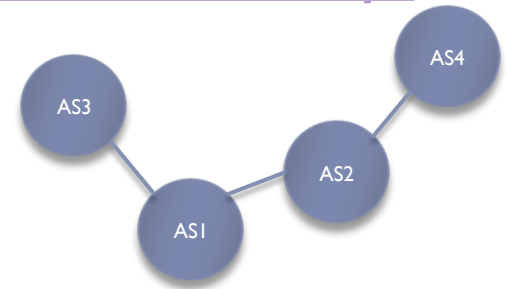
BGP Tools

- ▶ Finding the AS number
 - ▶ `$ whois -h whois.cymru.com " -v <IP_ADDRESS> "`
- ▶ BGPlay: <http://bgplay.routeviews.org/>
- ▶ BGP “Looking Glasses”
 - ▶ <http://www.bgp4.as/looking-glasses>
 - ▶ `$ telnet route-server.gblx.net`
 - ▶ `sh ip bgp 128.192.126.140`
 - ▶ `sh ip bgp`
 - (<https://learningnetwork.cisco.com/servlet/JiveServlet/showImage/2-224357-96596/show+ip+bgp+output.jpg>)

BGP Hijack Case Study

- ▶ <http://www.ripe.net/internet-coordination/news/industry-developments/youtube-hijacking-a-ripe-ncc-ris-case-study>
- ▶ <http://blog.cloudflare.com/why-google-went-offline-today-and-a-bit-about>

- ▶ **BGP gives precedence to smaller prefixes**



- ▶ E.g., consider the following case:

- ▶ AS2 advertises (128.192.10.0/24, [AS4 AS2]) to AS1
- ▶ AS3 advertises (128.192.0.0/16, [AS3]) to AS1
- ▶ A host in AS1 needs to send a packet to 128.192.10.123

- ▶ In this case, BGP will lookup the forwarding table, and match the longest common CIDR prefix → 128.192.10.0/24
 - Packets will be routed to AS2 and then to AS4