Module 4

The Network Layer

Chapter 4: outline

4.1 introduction

4.2 virtual circuit and datagram networks

4.3 what's inside a router

4.4 IP: Internet Protocol

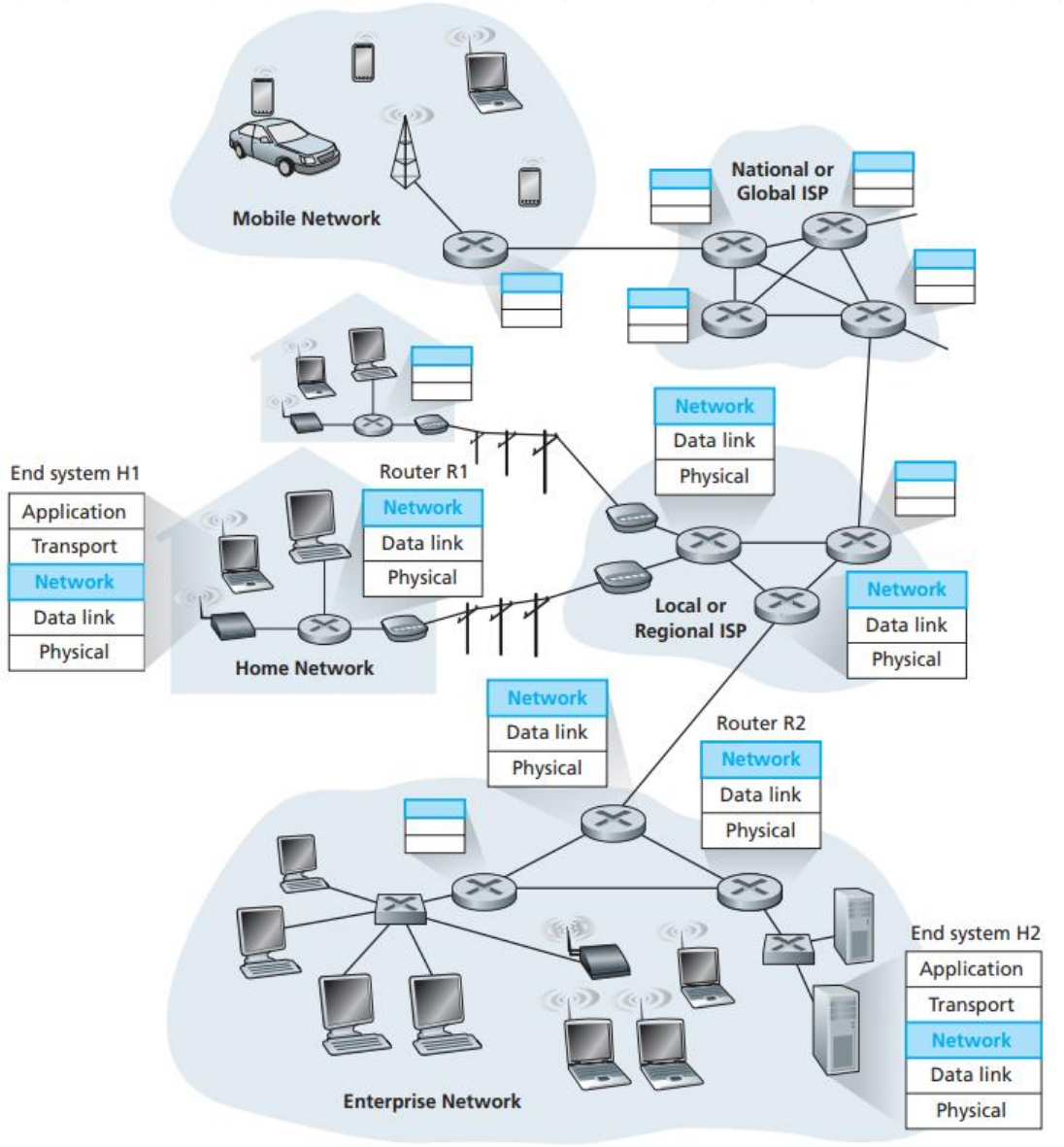
- datagram format
- IPv4 addressing
- ICMP
- IPv6

4.5 routing algorithms

- link state
- distance vector
- hierarchical routing

Network layer

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



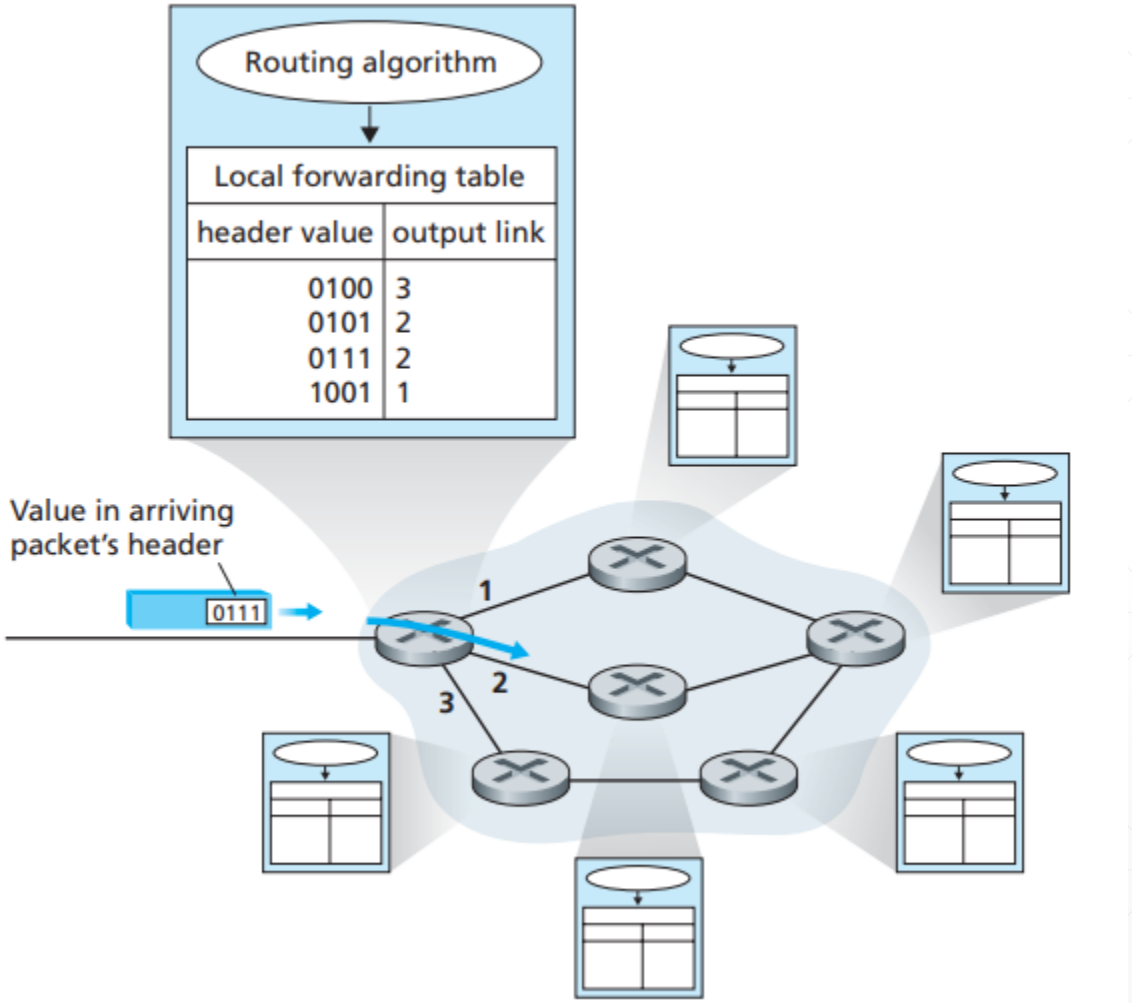
Two key network-layer functions

- *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

Routing algorithms determine values in forwarding tables



Connection setup

- 3rd important function in *some* network architectures:
  - ATM, frame relay, X.25
- before datagrams flow, two end hosts *and* intervening routers establish virtual connection
  - routers get involved
- network vs transport layer connection service:
  - *network:* between two hosts (may also involve intervening routers in case of VCs)
  - *transport:* between two processes

Network service model

Q: What *service model* for "channel" transporting datagrams from sender to receiver?

example services for individual datagrams:

- guaranteed delivery
- guaranteed delivery with less than 40 msec delay

example services for a flow of datagrams:

- in-order datagram delivery
- guaranteed minimum bandwidth to flow
- restrictions on changes in inter-packet spacing (jitter)
- Security services

Network layer service models:

Network Architecture	Service Model	Bandwidth Guarantee	No-Loss Guarantee	Ordering	Timing	Congestion Indication
Internet	Best Effort	None	None	Any order possible	Not maintained	None
ATM	CBR	Guaranteed constant rate	Yes	In order	Maintained	Congestion will not occur
ATM	ABR	Guaranteed minimum	None	In order	Not maintained	Congestion indication provided

Connection, connection-less service

- *datagram* network provides network-layer *connectionless* service
- *virtual-circuit* network provides network-layer *connection* service
- analogous to TCP/UDP connecton-oriented / connectionless transport-layer services, but:
  - *service:* host-to-host
  - *no choice:* network provides one or the other
  - *implementation:* in network core

Virtual circuits

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

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

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

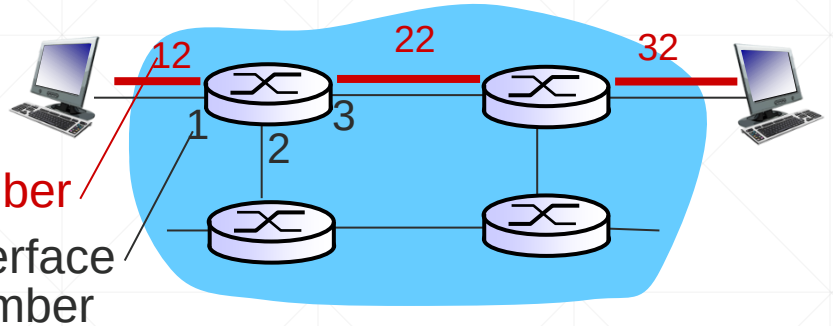
VC implementation

a VC consists of:

1. *path* from source to destination
2. *VC numbers*, one number for each link along path
3. *entries in forwarding tables* in routers along path

- packet belonging to VC carries VC number (rather than dest address)
- VC number can be changed on each link.
  - new VC number comes from forwarding table

VC forwarding table



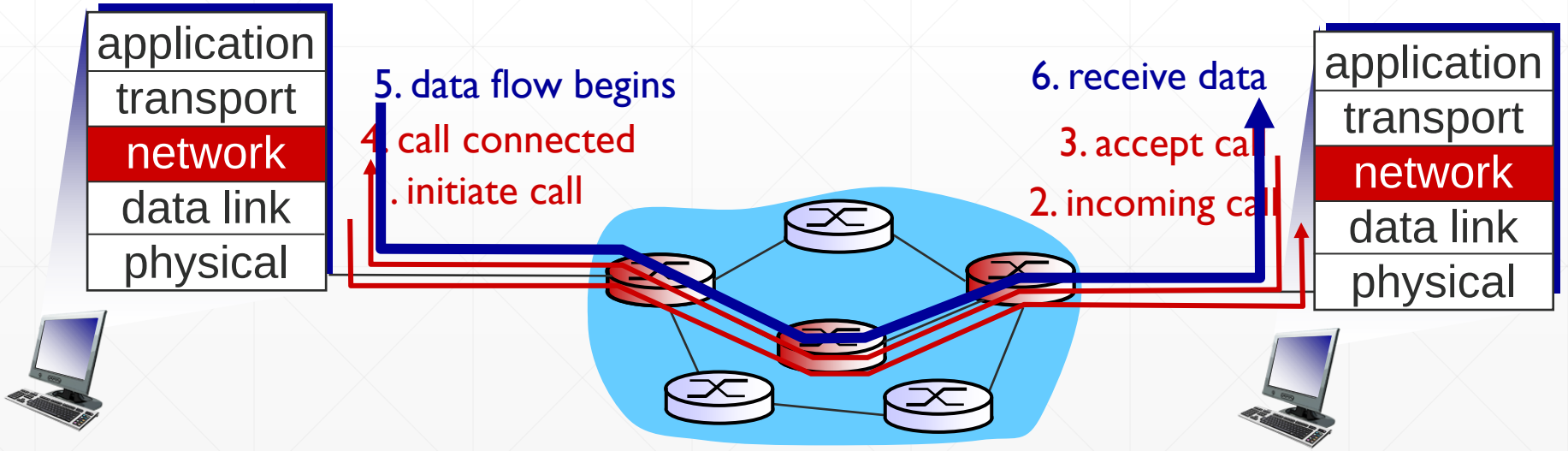
forwarding table in northwest router:

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

VC routers maintain connection state information!

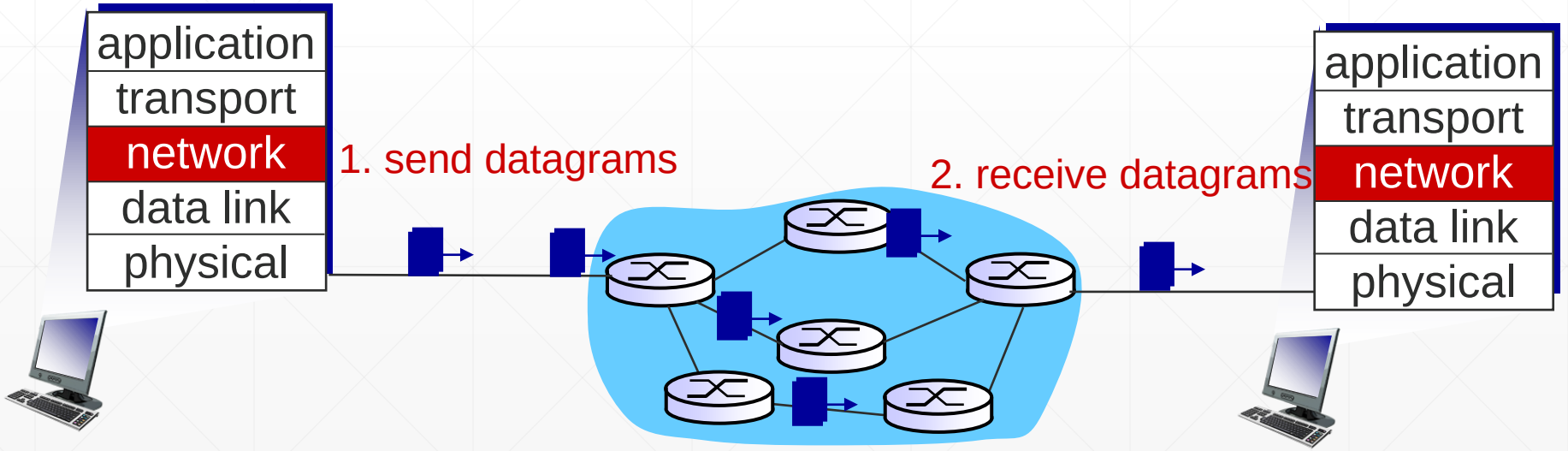
Virtual circuits: signaling protocols

- used to setup, maintain teardown VC
- used in ATM, frame-relay, X.25
- not used in today’ s Internet

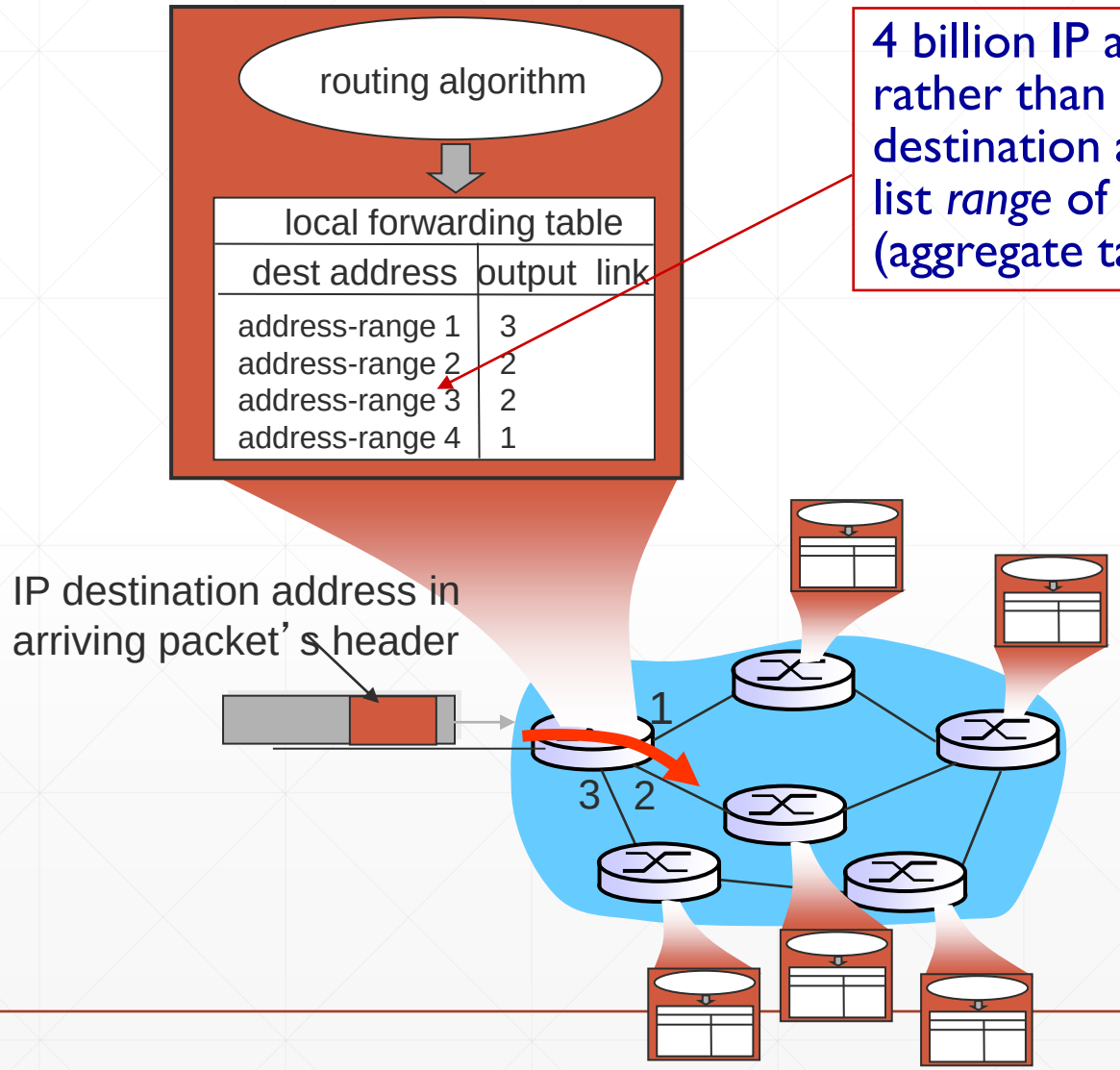


Datagram networks

- 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



Datagram forwarding table



Datagram forwarding table

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

Q: but what happens if ranges don't divide up so nicely?

Longest prefix matching

longest prefix matching

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

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

examples:

DA: 11001000 00010111 00010110 10100001 which interface?

DA: 11001000 00010111 00011000 10101010 which interface?

Datagram or VC network: why?

Internet (datagram)

- data exchange among computers
  - "elastic" service, no strict timing req.
- many link types
  - different characteristics
  - uniform service difficult
- "smart" end systems (computers)
  - can adapt, perform control, error recovery
  - ***simple inside network, complexity at "edge"***

ATM (VC)

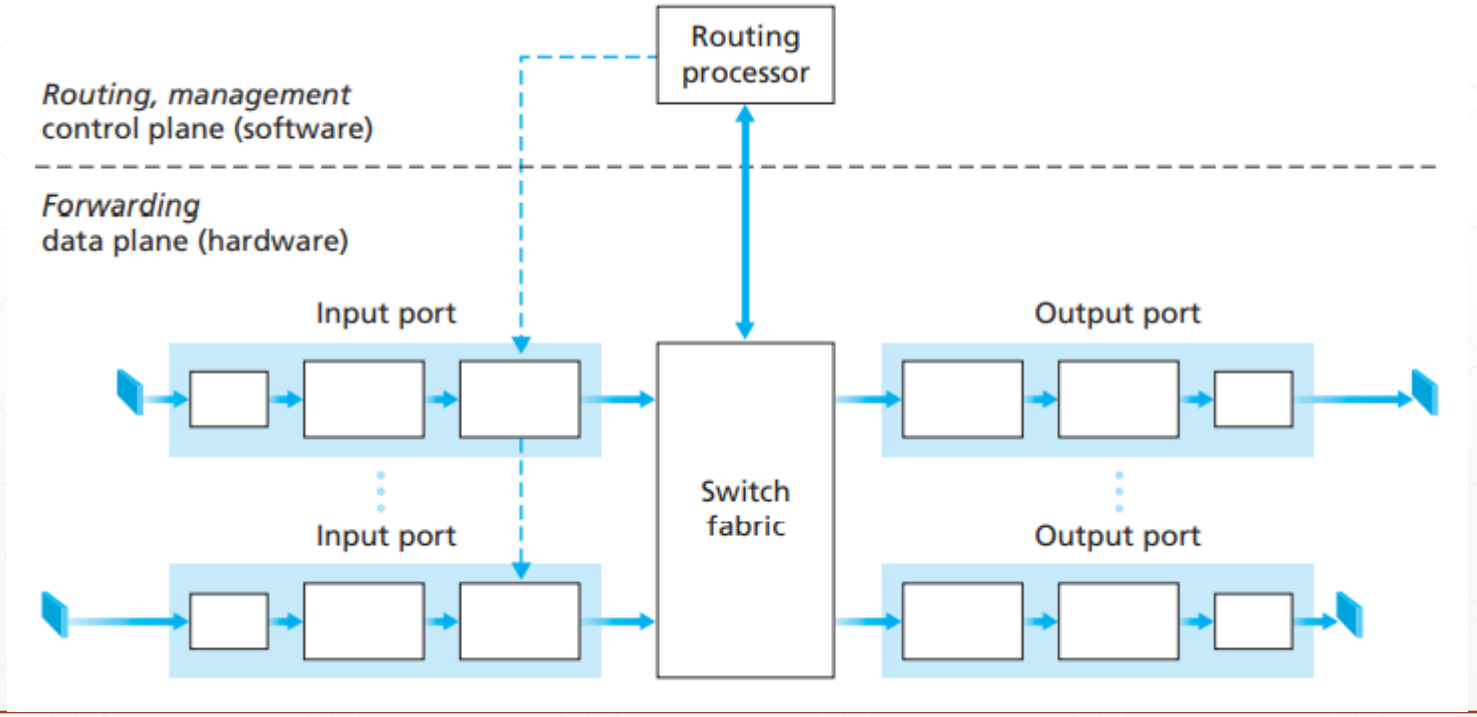
- evolved from telephony
- human conversation:
  - strict timing, reliability requirements
  - need for guaranteed service
- "dumb" end systems
  - telephones
  - ***complexity inside network***

What's inside a Router?

Router architecture overview

two key router functions:

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



Generic router architecture

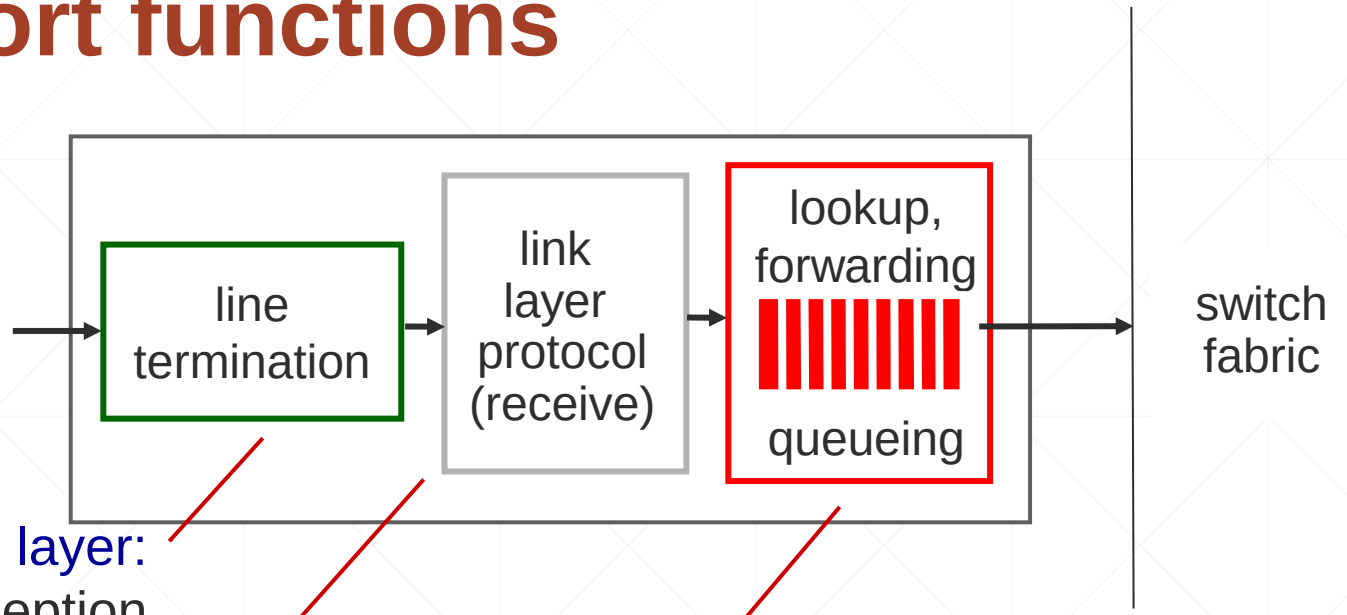
Router Architecture (conti.)

- **Four router components:**
- **(1) Input ports:**
  - Performs physical layer function like terminating an incoming physical link at a router
  - Performs link layer function
  - Lookup function performed (forwarding table is consulted for output port)
  - **Control packets** (carrying routing protocol info) is sent from input port to routing processor
  - **Port** : physical input and output router interfaces
- **(2) Switching fabric:**
  - Connects router input port to its output port

Router Architecture (conti.)

- **(3) Output port:**
  - Stores packets received from the switching fabric and transmits on the outgoing link by performing physical and link layer functions
- **(4) Routing processor:**
  - Executes the routing protocols, maintains routing tables, attaches state information and computes forwarding table for router
- **Router Forwarding Plane:**
  - Collectively the forwarding functions (input port, output port and switching fabric)
  - Implemented using hardware
- **Router Control Plane:**
  - Control functions like executing the routing protocol and responding to attached links
  - Implemented in software
  - Execute on the router processor (CPU)

Input port functions



physical layer:
bit-level reception

data link layer:
e.g., Ethernet
see chapter 5

decentralized switching:

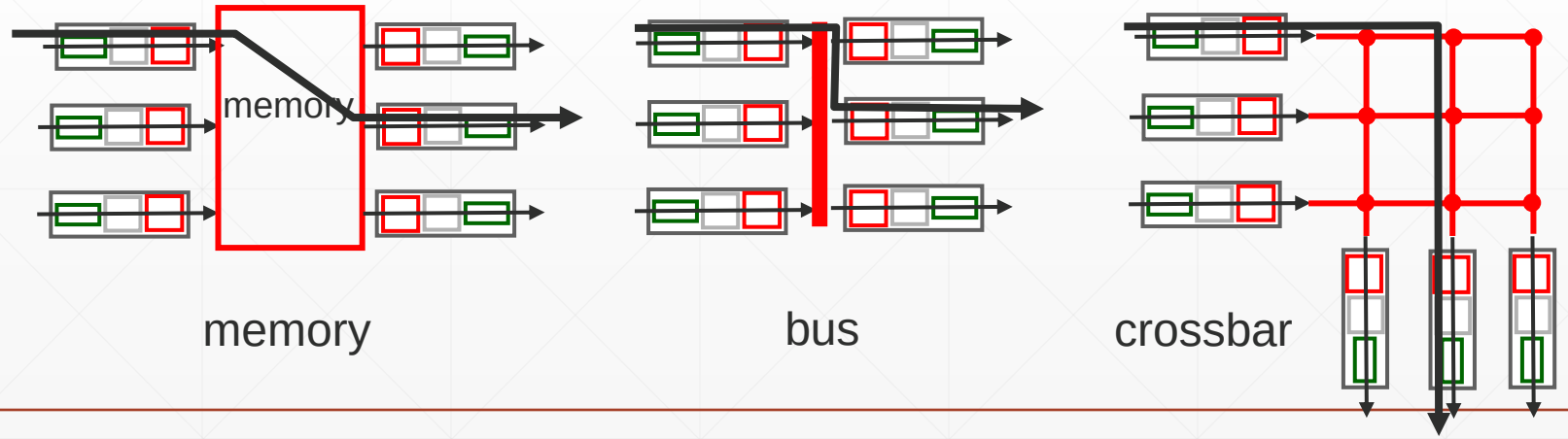
- given datagram dest., lookup output port using forwarding table in input port memory *("match plus action")*
- goal: complete input port processing at 'line speed'
- queuing: if datagrams arrive faster than forwarding rate into switch fabric

Input port functions

- forwarding table is computed and updated by the routing processor
- a shadow copy is stored at each input port
- forwarding table is copied from the routing processor to the line cards over a separate bus indicated by the dashed line
- With a shadow copy, forwarding decisions can be made locally, at each input port, without invoking the centralized routing processor

Switching fabrics

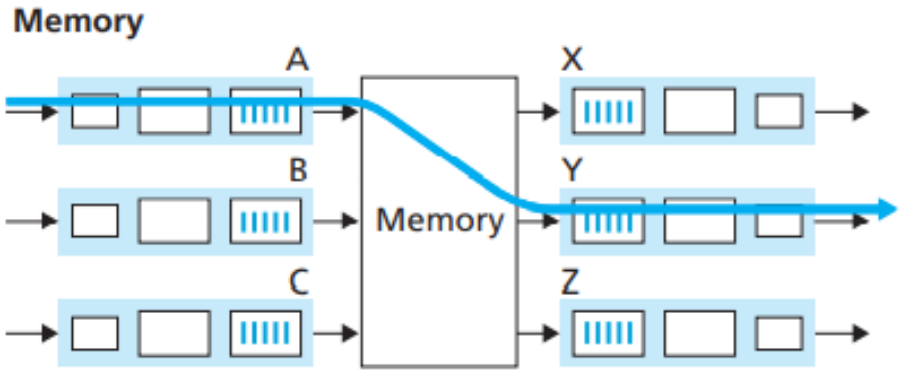
- transfer packet from input buffer to appropriate output buffer
- switching rate: rate at which packets can be transfer from inputs to outputs
  - often measured as multiple of input/output line rate
  - N inputs: switching rate N times line rate desirable
- three types of switching fabrics



Switching via memory

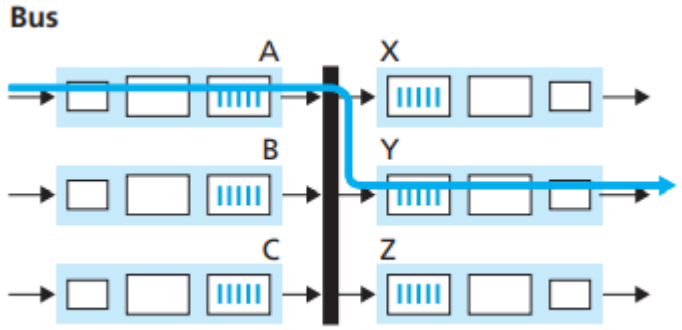
first generation routers:

- traditional computers with switching under direct control of CPU (routing processor)
- input port with an arriving packet first signaled the routing processor via an interrupt
- packet copied to processor memory
- routing processor then extracts the destination address from the header
- looked up the appropriate output port in the forwarding table, and copied the packet to the output port's buffers
- speed limited by memory bandwidth (2 bus crossings per datagram)



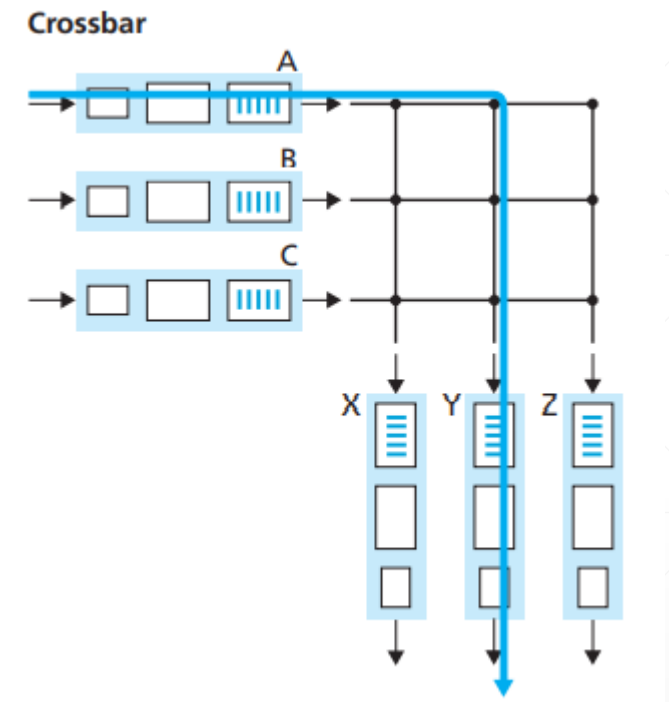
Switching via a bus

- an input port transfers a packet directly to the output port over a shared bus
  - without intervention by the routing processor
- input port pre-pend a header to the packet indicating the local output port to which this packet is being transferred
- packet is received by all output ports
- port that matches the label will keep the packet
- *bus contention:* switching speed limited by bus bandwidth
- 32 Gbps bus, Cisco 5600: sufficient speed for access and enterprise routers

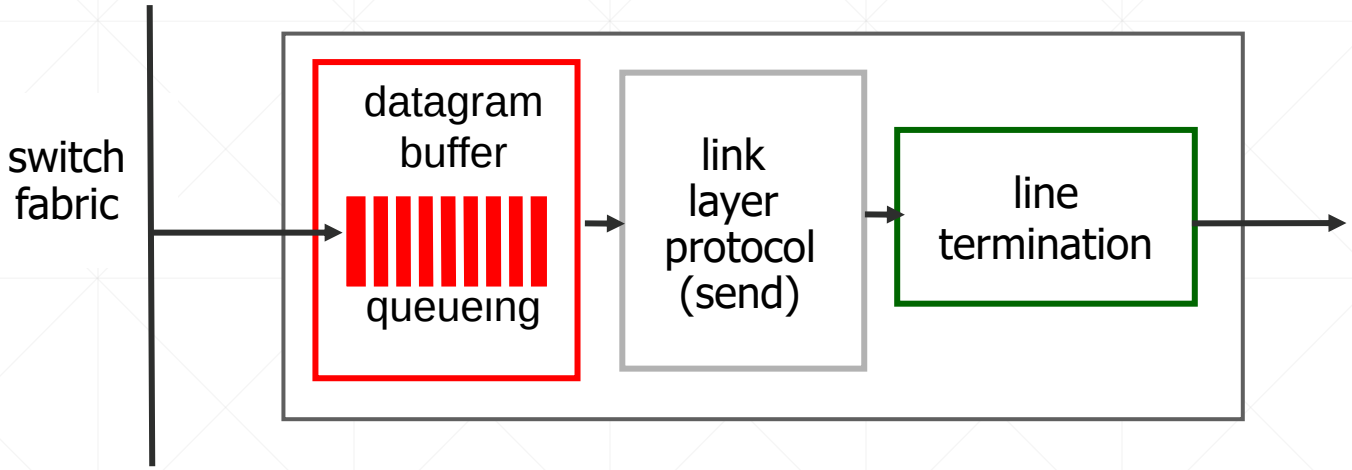


Switching via interconnection network

- A crossbar switch is an interconnection network consisting of 2N buses that connect N input ports to N output port
- overcome bus bandwidth limitations
- banyan networks, crossbar, other interconnection nets initially developed to connect processors in multiprocessor
- advanced design: fragmenting datagram into fixed length cells, switch cells through the fabric.
- Cisco 12000: switches 60 Gbps through the interconnection network



Output ports

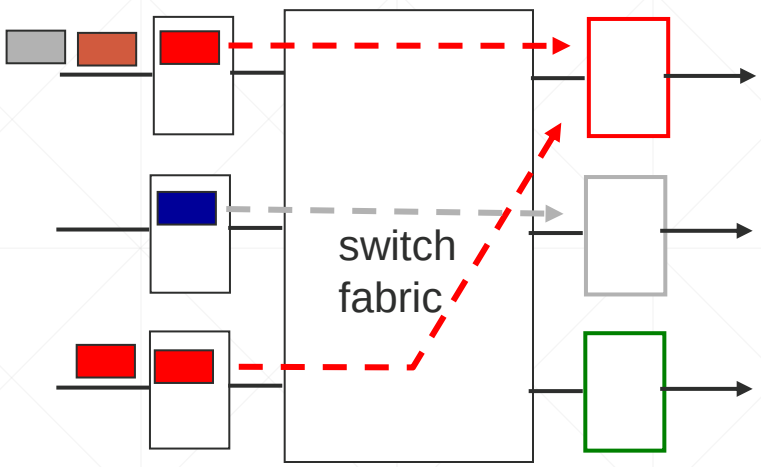


- *buffering* required when datagrams arrive from fabric faster than the transmission rate
- *scheduling discipline* chooses among queued datagrams for transmission

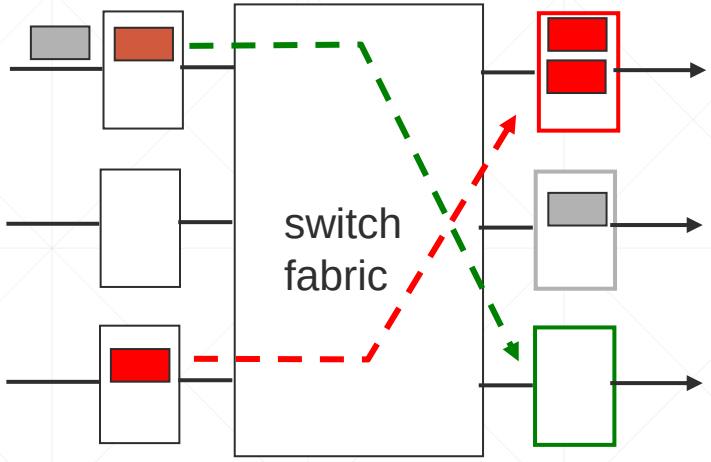
Priority scheduling – who gets best performance, network neutrality

Datagram (packets) can be lost due to congestion, lack of buffers

Output port queueing



at *t*, packets more from input to output



one packet time later

- buffering when arrival rate via switch exceeds output line speed
- *queueing (delay) and loss due to output port buffer overflow!*

How much buffering?

- RFC 3439 rule of thumb: average buffering equal to “typical” RTT (say 250 msec) times link capacity C
  - e.g., C = 10 Gpbs link: 2.5 Gbit buffer
- recent recommendation: with *N* flows, buffering equal to

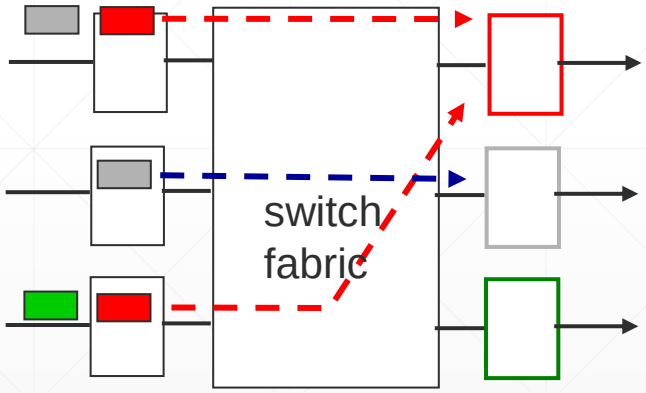
$$\frac{RTT \cdot C}{\sqrt{N}}$$

Output port

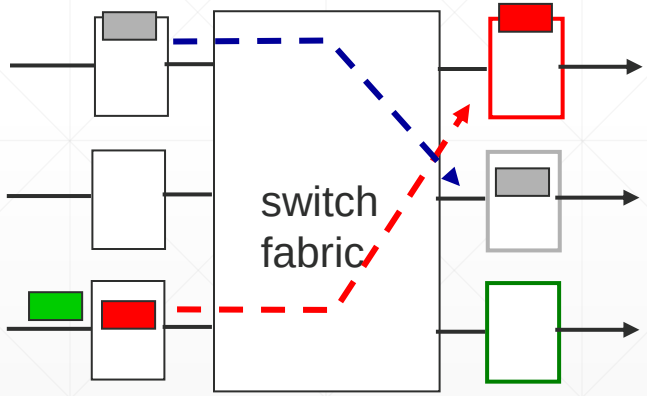
- A consequence of output port queuing is :
- **Packet scheduler** at the output port must choose one packet among those queued for transmission
- selection might be done on a simple basis, such as first-come-first-served (FCFS) scheduling, or weighted fair queuing (WFQ)
- If there is not enough memory to buffer an incoming packet, either drop the arriving or remove one or more already-queued packets
- Random Early Detection (RED) algorithm :
  - queue length is less than a minimum threshold, when a packet arrives, the packet is admitted to the queue
  - if the queue is full or the average queue length is greater than a maximum threshold, packet is dropped or marked

Input port queuing

- fabric slower than input ports combined -> queueing may occur at input queues
  - *queueing delay and loss due to input buffer overflow!*
- Head-of-the-Line (HOL) blocking: queued datagram at front of queue prevents others in queue from moving forward



output port contention:
only one red datagram can be transferred.
lower red packet is blocked

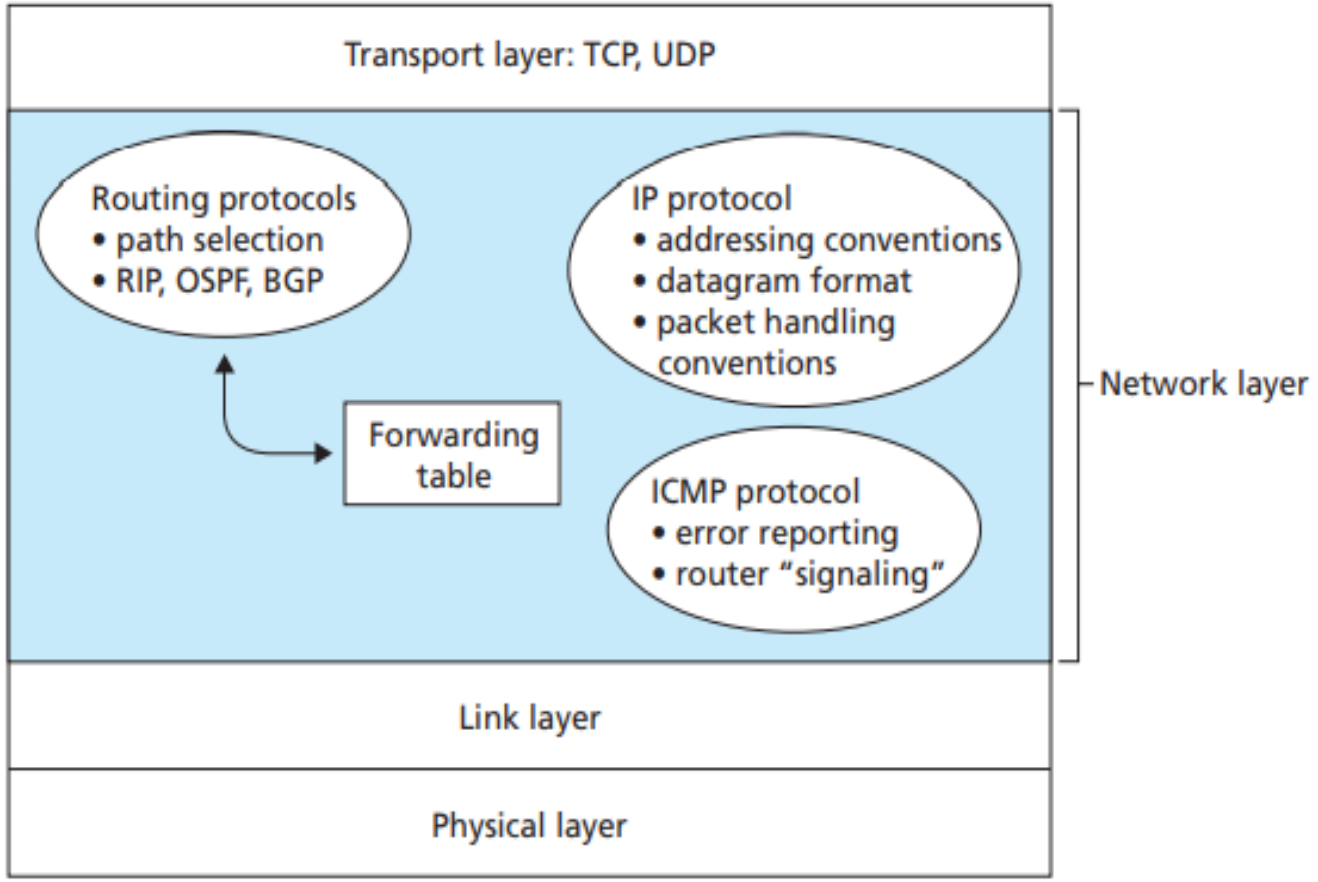


one packet time later:
green packet
experiences HOL
blocking

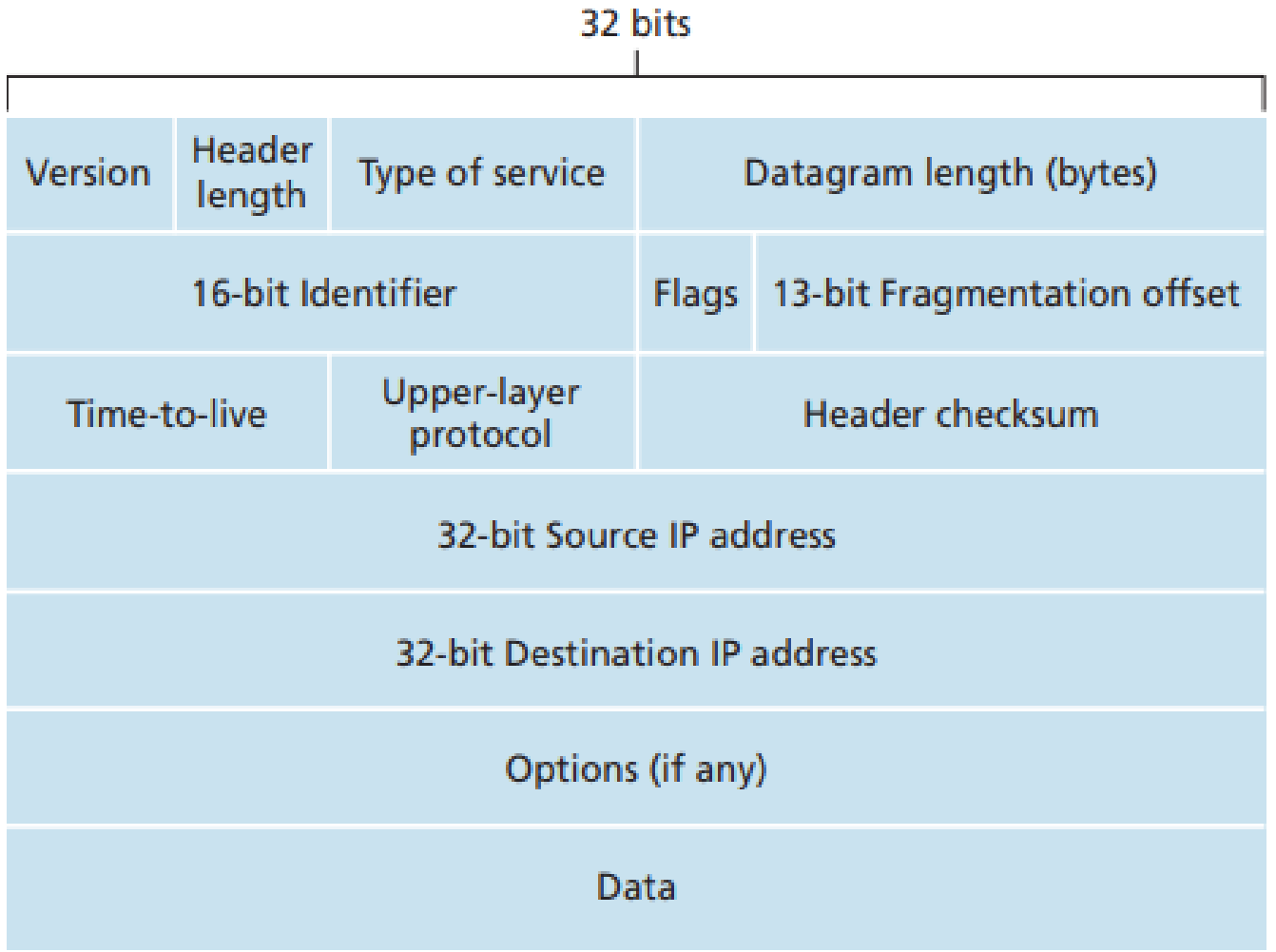
The Internet network layer

Internet's network layer major components:

- **IP Protocol**
- **Routing Component** (determines the path a datagram follows)
- **ICMP Protocol** (error and information reporting protocol)



IP Datagram Format



Datagram Format

- Network layer packet is referred as **datagram**
- Key fields in IPv4 datagram:
  - **Version Number:**
    - 4 bits specify the IP protocol version of the datagram
    - Different versions of IP use different datagram formats
  - **Header Length:**
    - IPv4 datagram can contain a variable number of options
    - 4 bits are needed to determine where in the IP datagram the data begins
    - typical IP datagram has a 20-byte header

Datagram Format

- **Type of Service:**
  - allow different types of IP datagrams
  - datagrams requiring low delay, high throughput, or reliability
- **Datagram Length:**
  - total length of the IP datagram (header plus data) in bytes
  - 16 bits field
  - minimum value 20 bytes and the maximum is 65,535 bytes
- **Identifier, flags fragmentation offset:**
  - Used for IP fragmentation

Datagram Format

- **Time-to-Live:**
  - 8 bits
  - ensure that datagrams do not circulate forever in network
  - this field is decremented by one each time the datagram is processed by a router
  - TTL field reaches 0, the datagram must be dropped
- **Protocol:**
  - used only when an IP datagram reaches its final destination
  - value of this field indicates the specific transport-layer protocol to which the data portion of this IP datagram should be passed
  - 6 indicates that the data portion is passed to TCP
  - 17 indicates that the data is passed to UDP
- **Port number** binds transport and application layer
- **Protocol number** binds network and transport layer

Datagram Format

- **Header Checksum:**
  - aids a router in detecting bit errors in a received IP datagram
  - 1s complement of the sum of bytes, known as the Internet checksum, is stored in the checksum field
  - checksum must be recomputed and stored again at each router
  - routers discard datagrams for which an error has been detected
  - only the IP header is checksummed at the IP layer
  - TCP/UDP checksum is computed over the entire TCP/UDP segment
- **Source and Destination IP addresses:**
  - source creates a datagram; it inserts the source IP address and destination IP address
  - source host determines the destination address via a DNS lookup

Datagram Format

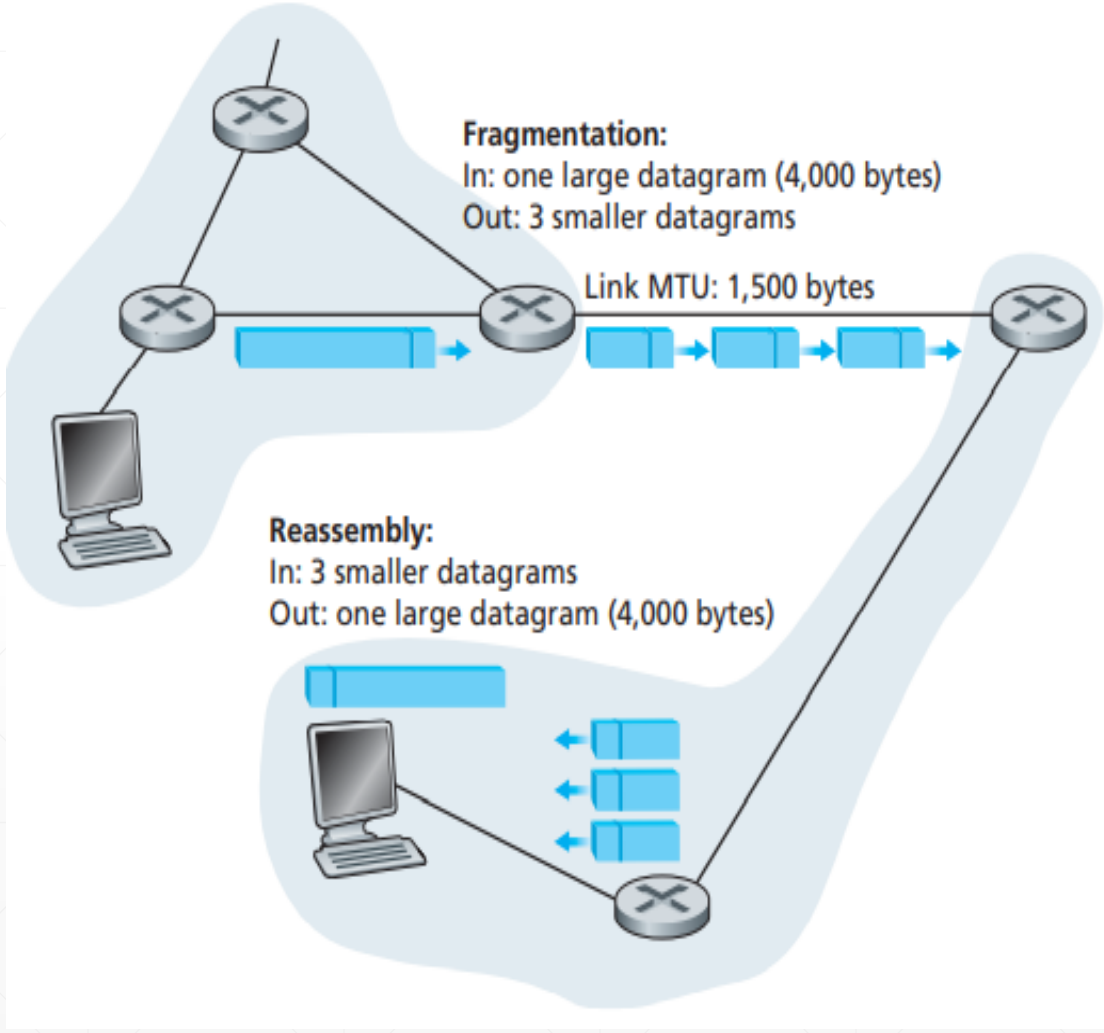
- **Options:**
  - allow an IP header to be extended
- **Data(payload):**
  - the data field of the IP datagram contains the transport-layer segment (TCP or UDP)
  - data field can carry other types of data, such as ICMP messages
- Due to presence of option field, size of header is 20 to 60 bytes

IP Datagram Fragmentation

- maximum amount of data that a link-layer frame can carry is called the **maximum transmission unit (MTU)**
- problem is
  - each link along the route between sender and destination can use different link-layer protocols, and each of these protocols can have **different MTUs**
- When outgoing link has an MTU smaller than the size of IP datagram
  - Fragment the data into two or three smaller datagrams
  - Smaller datagrams are called fragments

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, reassembly

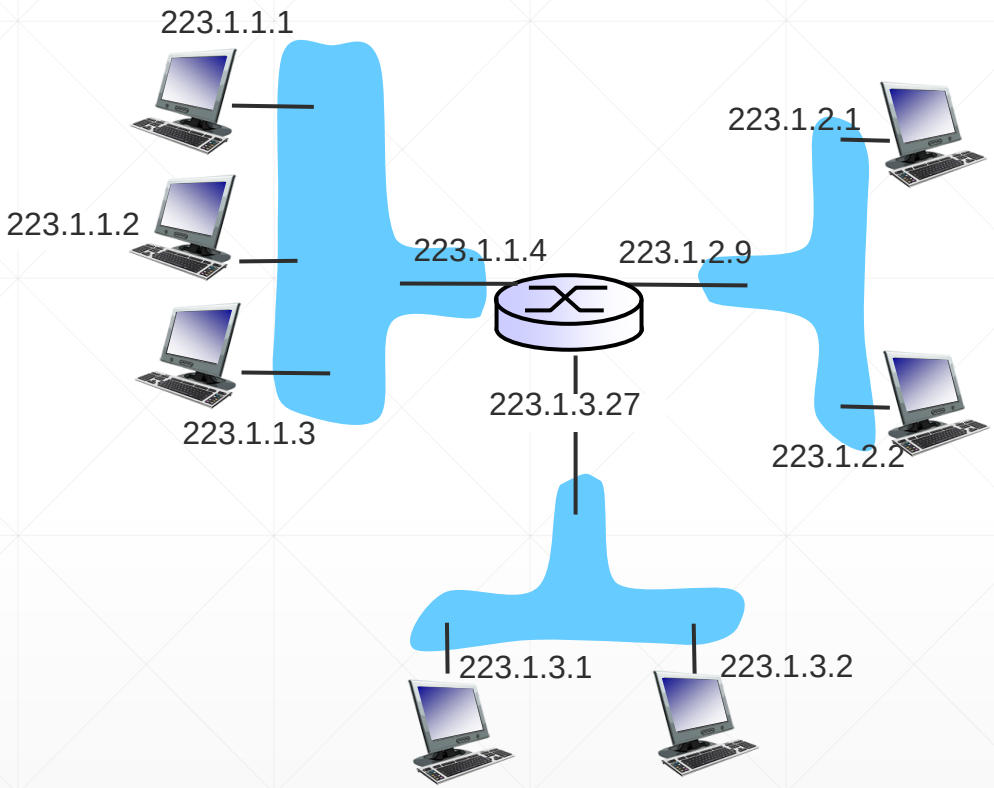
example:

- 4000 byte datagram
- 20 bytes header
- 3980 bytes of payload
- MTU = 1500 bytes

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

IP addressing: Introduction

- *IP address:* 32-bit identifier for host, router *interface*
- *interface:* connection between host/router and physical link
  - Routers typically have multiple interfaces
  - host typically has one or two interfaces (e.g., wired Ethernet, wireless 802.11)
- *IP address is associated with each interface rather than host or router containing that interface*



223.1.1.1 = 11011111 00000001 00000001 00000001

223 1 1 1

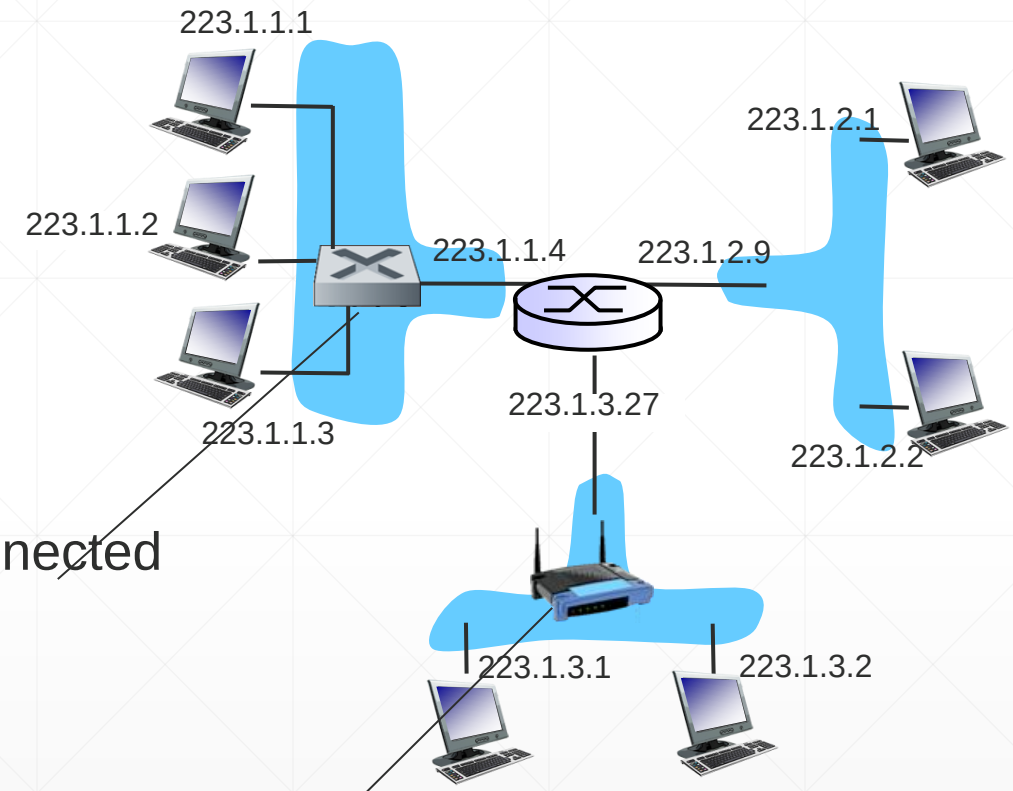
IP addressing: Introduction

- IP address is 32 bits long (4 bytes)
- Total of **2^32** possible IP addresses
- Written in **Dotted Decimal Notation**
  - each byte of the address is written in its decimal form and is separated by a period (dot) from other bytes in the address
  - the address 193.32.216.9 in binary notation is
  - 11000001 00100000 11011000 00001001

IP addressing: Introduction

Example of IP addressing and Interface

- one router (with three interfaces)
- Interconnected seven hosts



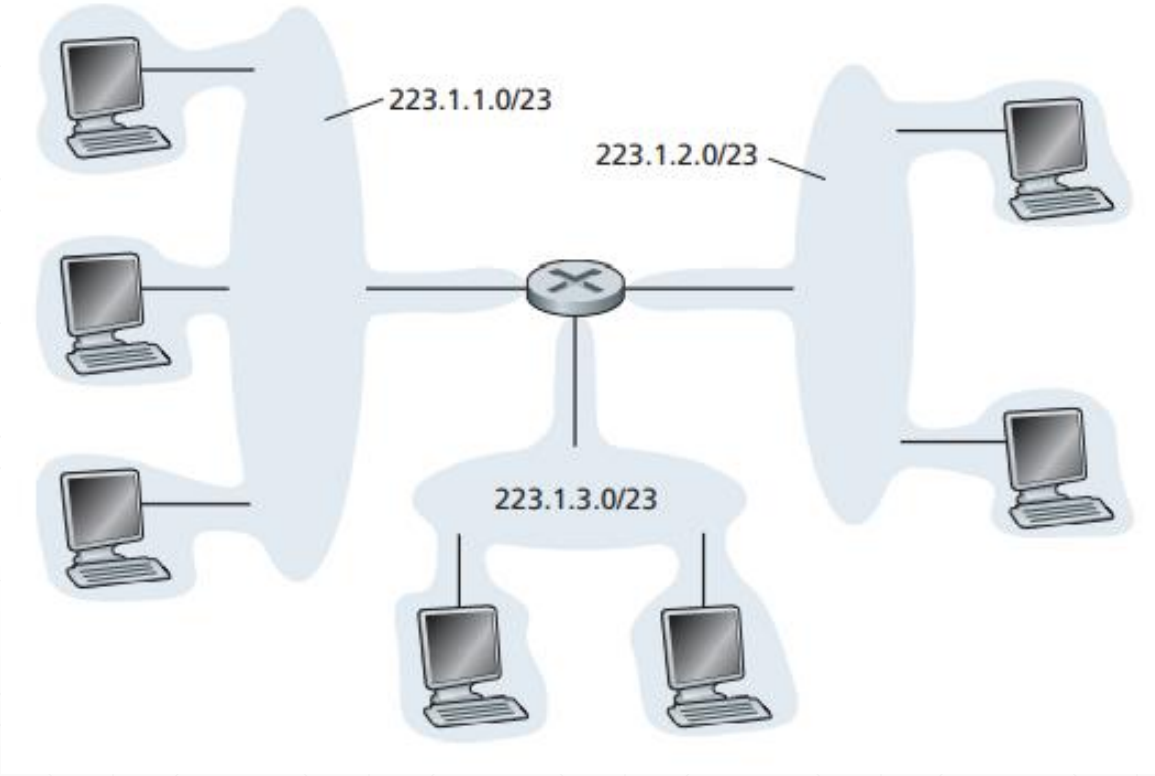
wired Ethernet interfaces connected by Ethernet switches

- The network interconnecting three host interfaces and one router interface forms a **subnet**

wireless WiFi interfaces connected by WiFi base station

Subnets

- IP address:
  - subnet part - high order bits(1s)
  - host part - low order bits(0s)
- *What's a subnet ?*
  - device interfaces with same subnet part of IP address
  - can physically reach each other *without intervening router*
  - The subnet is also called an IP network



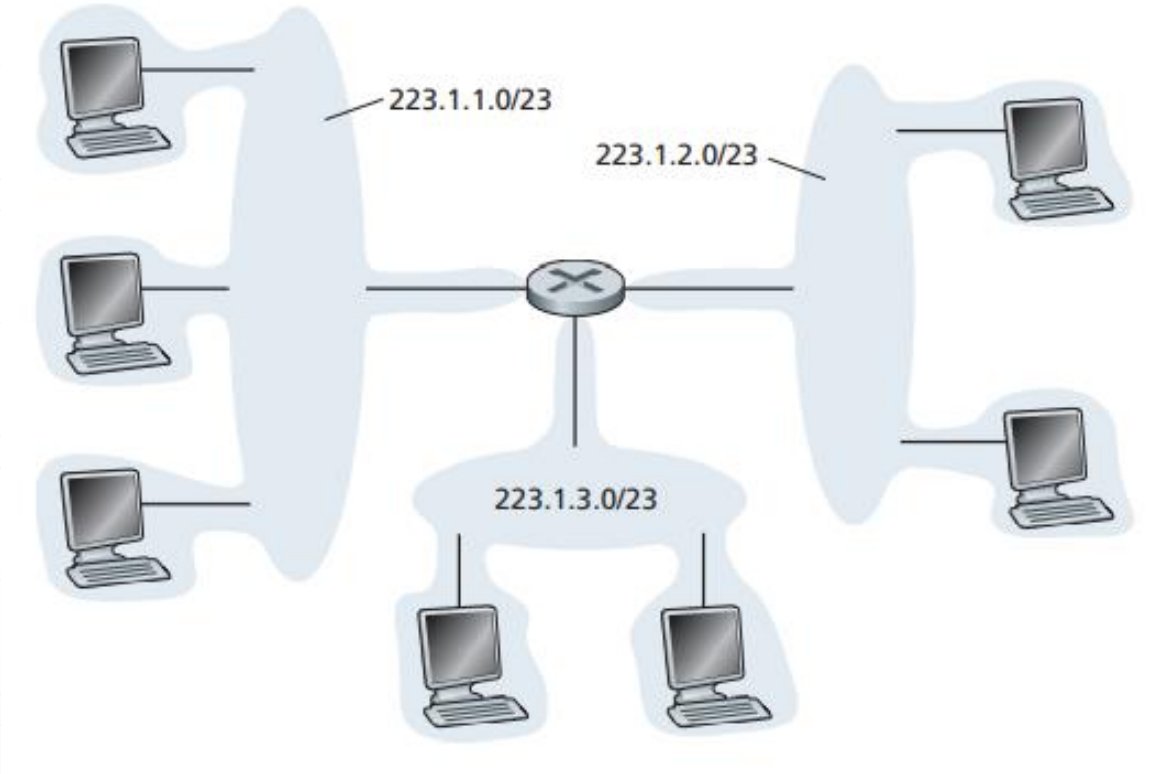
Subnet Addresses

Subnets

- IP addressing assigns an address to this subnet: 223.1.1.0/24
  - the /24 notation is known as a **subnet mask**
  - indicates that leftmost 24 bits of the 32-bit quantity define the **subnet address**
- subnet 223.1.1.0/24 thus consists of the three host interfaces (223.1.1.1, 223.1.1.2, and 223.1.1.3) and one router interface (223.1.1.4)
- Any additional hosts attached to the 223.1.1.0/24 subnet would be required to have an address of the form 223.1.1.xxx

Subnets

- 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 Addresses

Subnets

how many?

Each router has three interfaces,

one for each point-to-point link and

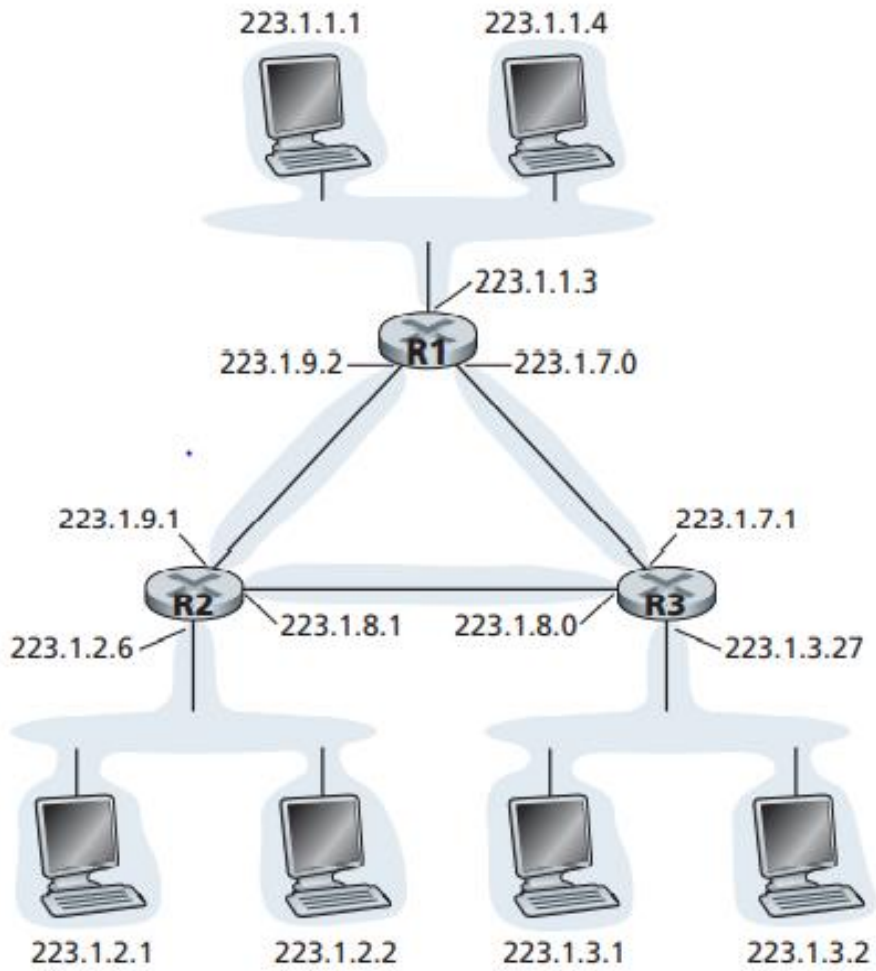
one for the broadcast link that directly connects the router to a pair of hosts

Three additional subnets:

223.1.9.0/24, for the interfaces that connect routers R1 and R2

223.1.8.0/24, for the interfaces that connect routers R2 and R3

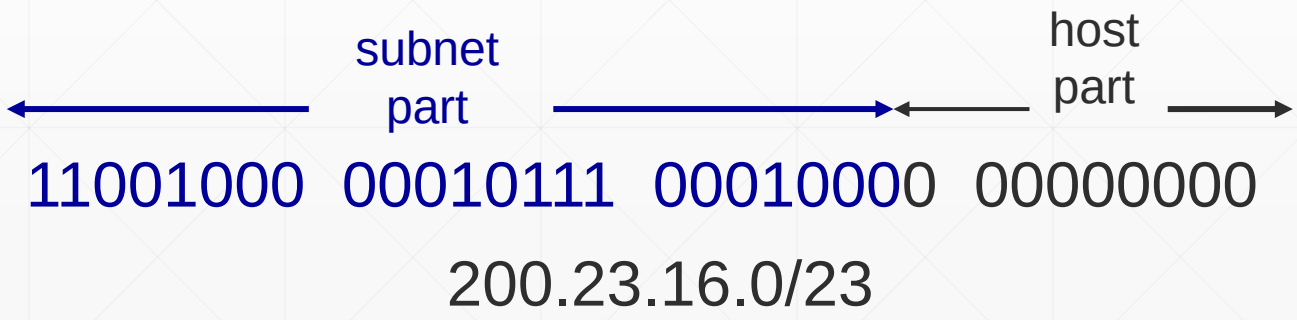
223.1.7.0/24, for the interfaces that connect routers R3 and R1



Three routers interconnecting six subnets

IP addressing: CIDR

- Internet's address assignment strategy is known as **Classless Interdomain Routing** (CIDR—pronounced cider) [RFC 4632]
- 32-bit IP address is divided into two parts
- has the **dotted-decimal** form **a.b.c.d/x**, where x indicates the **number of bits** in the first part of the address
- x most significant bits are referred as **prefix** of the address



IP addressing: CIDR

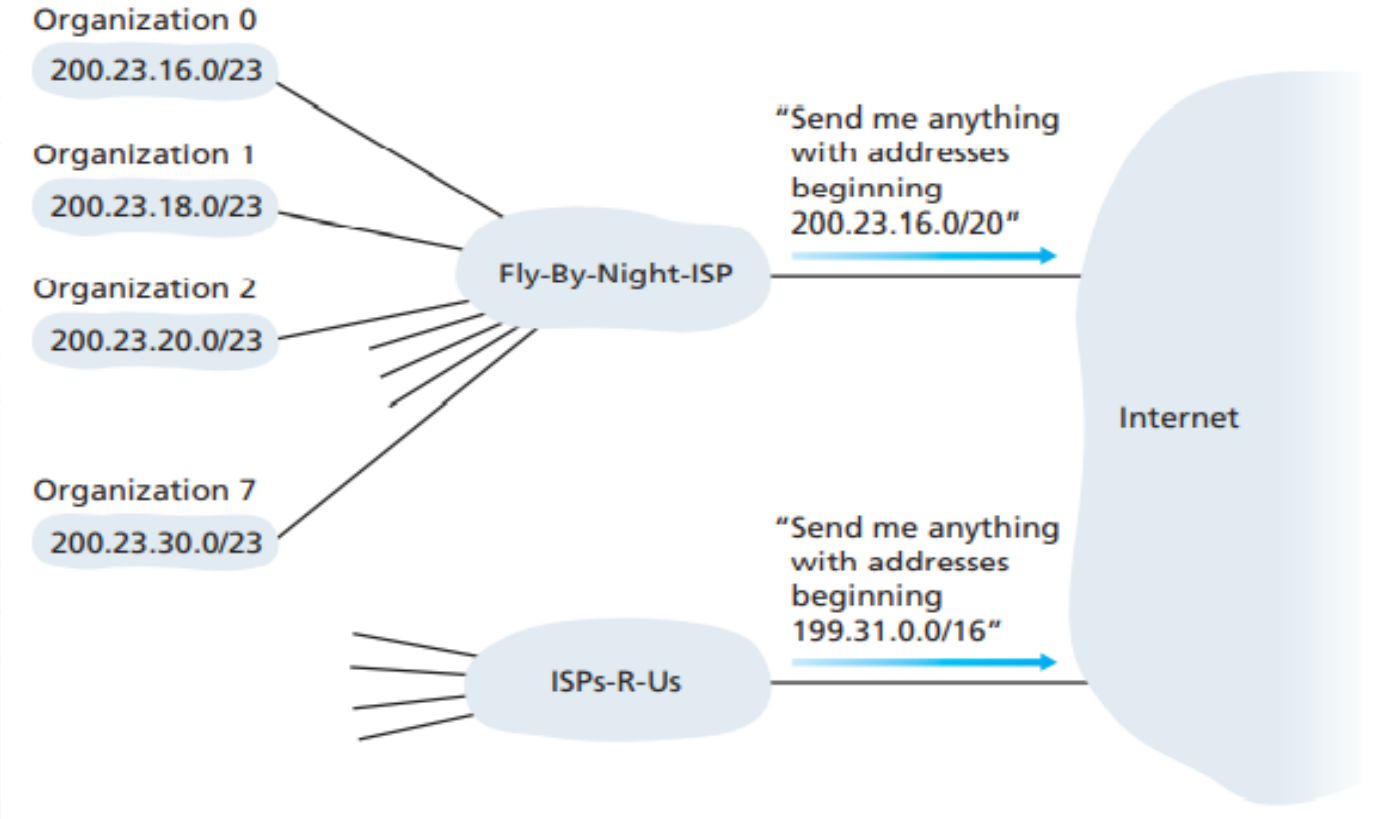
- the first 21 bits of the CIDRized address a.b.c.d/21 specify the organization's network prefix
- are common to the IP addresses of all devices in that organization
- remaining 11 bits identify the specific hosts in the organization
- These 11 rightmost bits are used for subnetting within the organization
- E.g. : a.b.c.d/24 might refer to a specific subnet within the organization

Address Aggregation or Route Aggregation

- **Address aggregation** : ability to use a single prefix to advertise multiple networks
- Example:
  - Fly-By-Night-ISP advertises to the outside world that it should be sent any datagrams whose first 20 address bits match 200.23.16.0/20
  - rest of the world need not know that within the address block 200.23.16.0/20 there are in fact eight other organizations
  - each with its own subnets

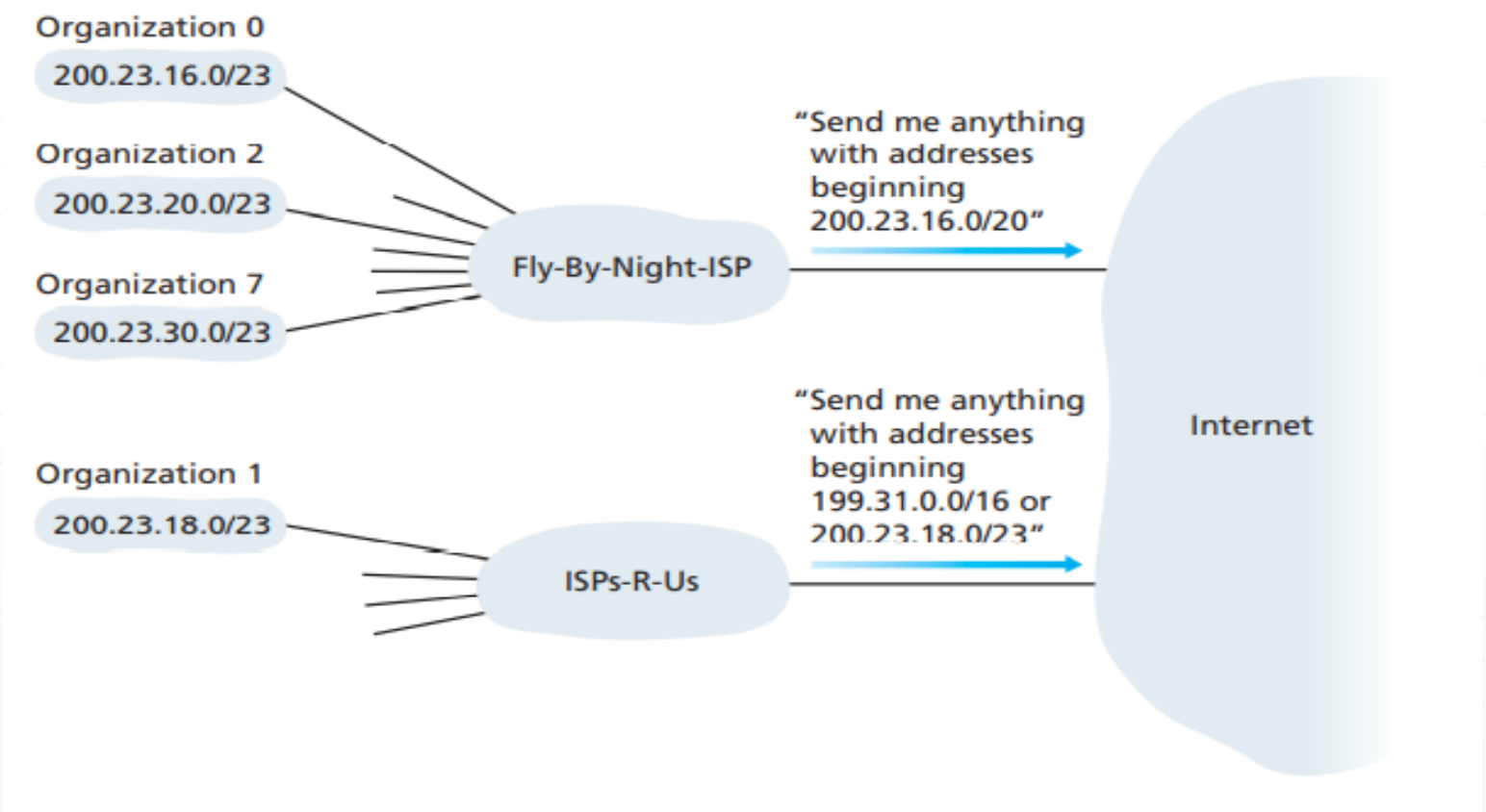
Hierarchical addressing: route aggregation

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 : Classful Addressing

- network portions of an IP address were constrained to be 8, 16, or 24 bits in length
- subnets with **8-, 16-, and 24-bit** subnet addresses were known as **class A, B, and C** networks, respectively
- class C (/24) subnet could accommodate only up to 28 – 2 = 254 hosts, too small for organizations
- class B (/16) subnet, which supports up to 65,634 hosts, was too large
- Another type of IP address
  - **Broadcast address 255.255.255.255 :** When a host sends a datagram with destination address 255.255.255.255, the message is delivered to all hosts on the same subnet

Obtaining a Block of Addresses

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

A: 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 00010000	00000000	200.23.16.0/23
Organization 1	11001000 00010111 00010010	00000000	200.23.18.0/23
Organization 2	11001000 00010111 00010100	00000000	200.23.20.0/23
...	.....	....	....
Organization 7	11001000 00010111 00011110	00000000	200.23.30.0/23

DHCP: Dynamic Host Configuration Protocol

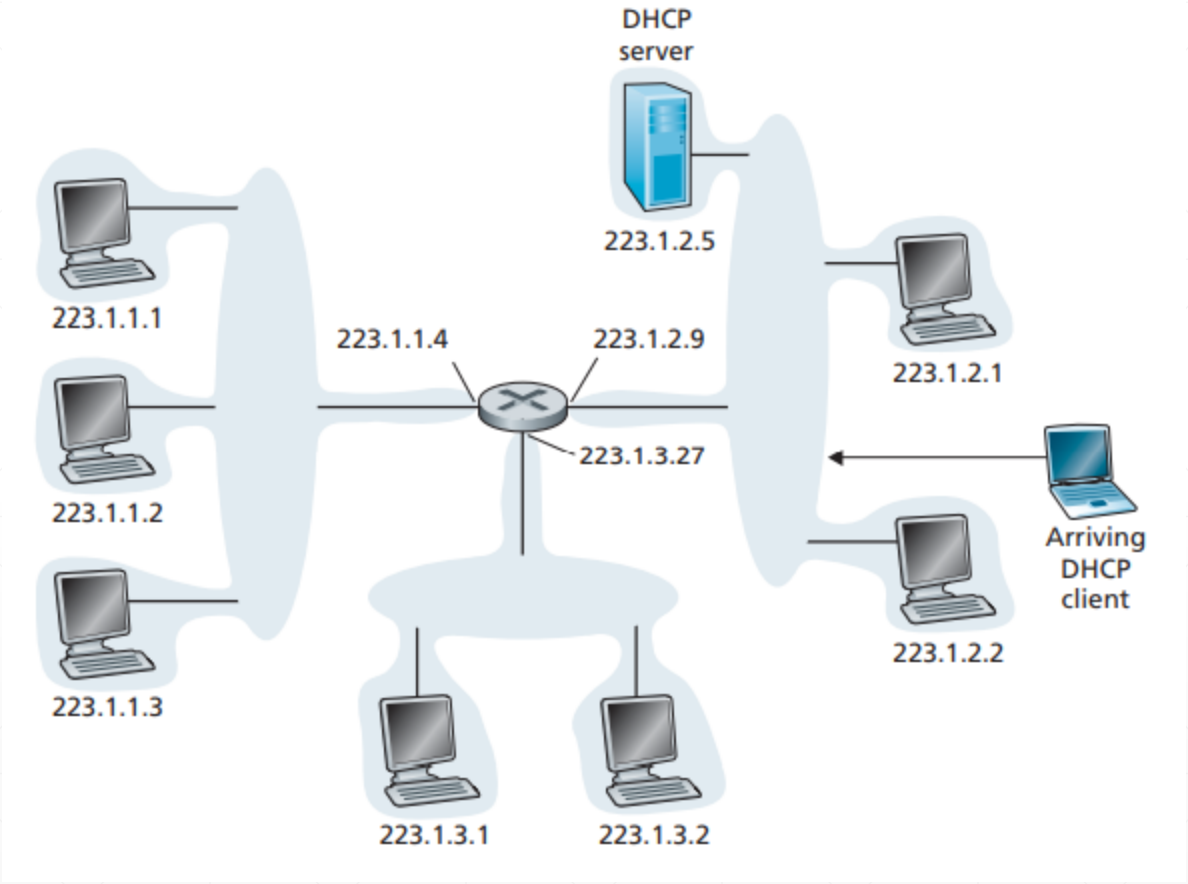
goal: allow host to *dynamically* obtain its IP address from network server when it joins network

- can renew its lease on address in use
- allows reuse of addresses (only hold address while connected/ “on”)
- support for mobile users who want to join network (more shortly)

For a newly arriving host, the DHCP protocol is a four-step process*:*

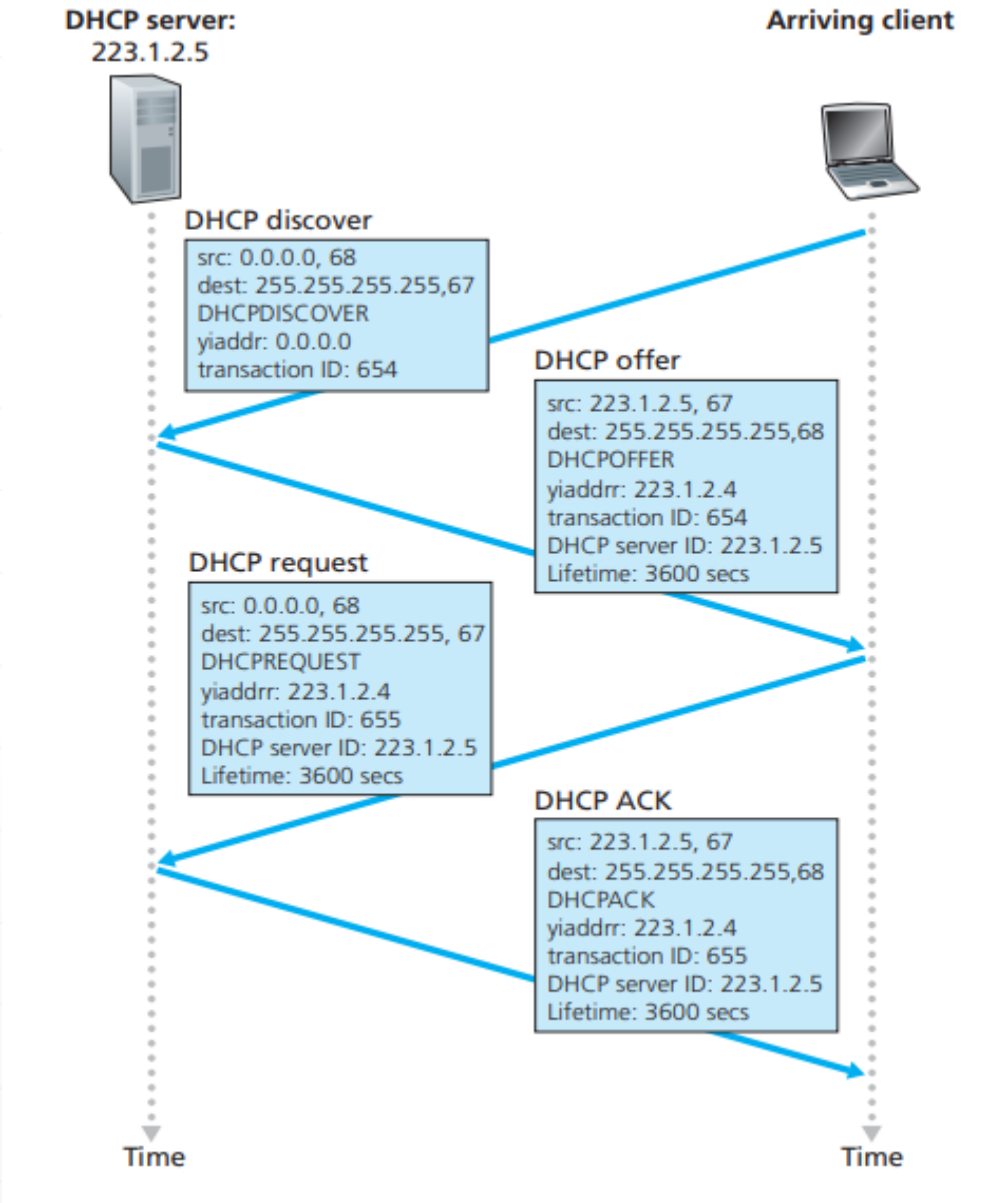
- **DHCP server discovery** : a client sends within a UDP packet to port 67
- **DHCP server offer(s)** : is broadcast to all nodes on the subnet since several DHCP servers can be present on the subnet
- **DHCP request :** newly arriving client will choose from among one or more server offers
- **DHCP ACK :** server responds to the DHCP request message with a DHCP ACK message

DHCP client-server scenario



DHCP client-server scenario

DHCP client-server interaction



DHCP: more than IP addresses

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)

IP addressing: the last word...

Q: how does an ISP get block of addresses?

A: ICANN: Internet Corporation for Assigned Names and Numbers
http://www.icann.org/

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

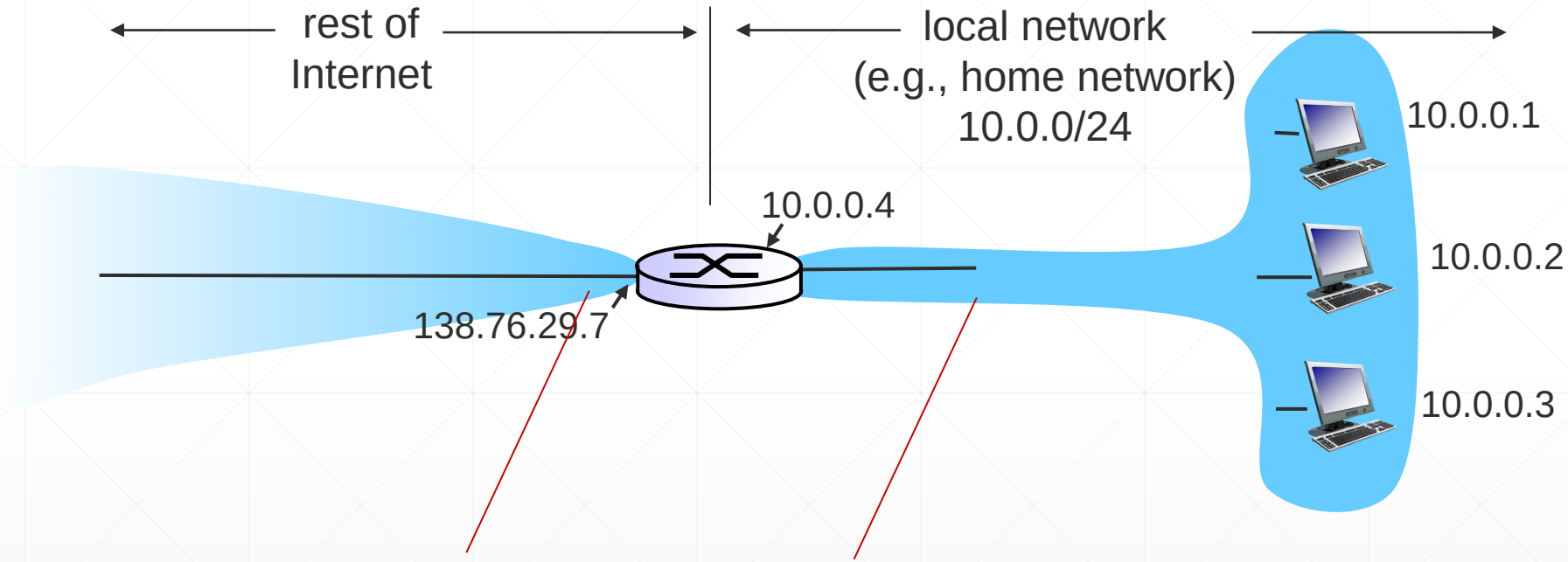
Network Address Translation (NAT)

- Used for address allocation in widespread areas
- RFC 2663, RFC 3022
- A **realm** with private addresses refers to a network whose addresses only have meaning to devices within that network
- The NAT-enabled router does not look like a router to the outside world
- NAT router behaves to the outside world as a single device with a single IP address
- NAT-enabled router is hiding the details of the home network from the outside world

Network Address Translation (NAT)

- all datagrams arriving at the NAT router from the WAN have the same destination IP address
- how does the router know the internal host to which it should forward a given datagram
  - **NAT translation table :** include port numbers as well as IP addresses in the table entries

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

implementation: NAT router must:

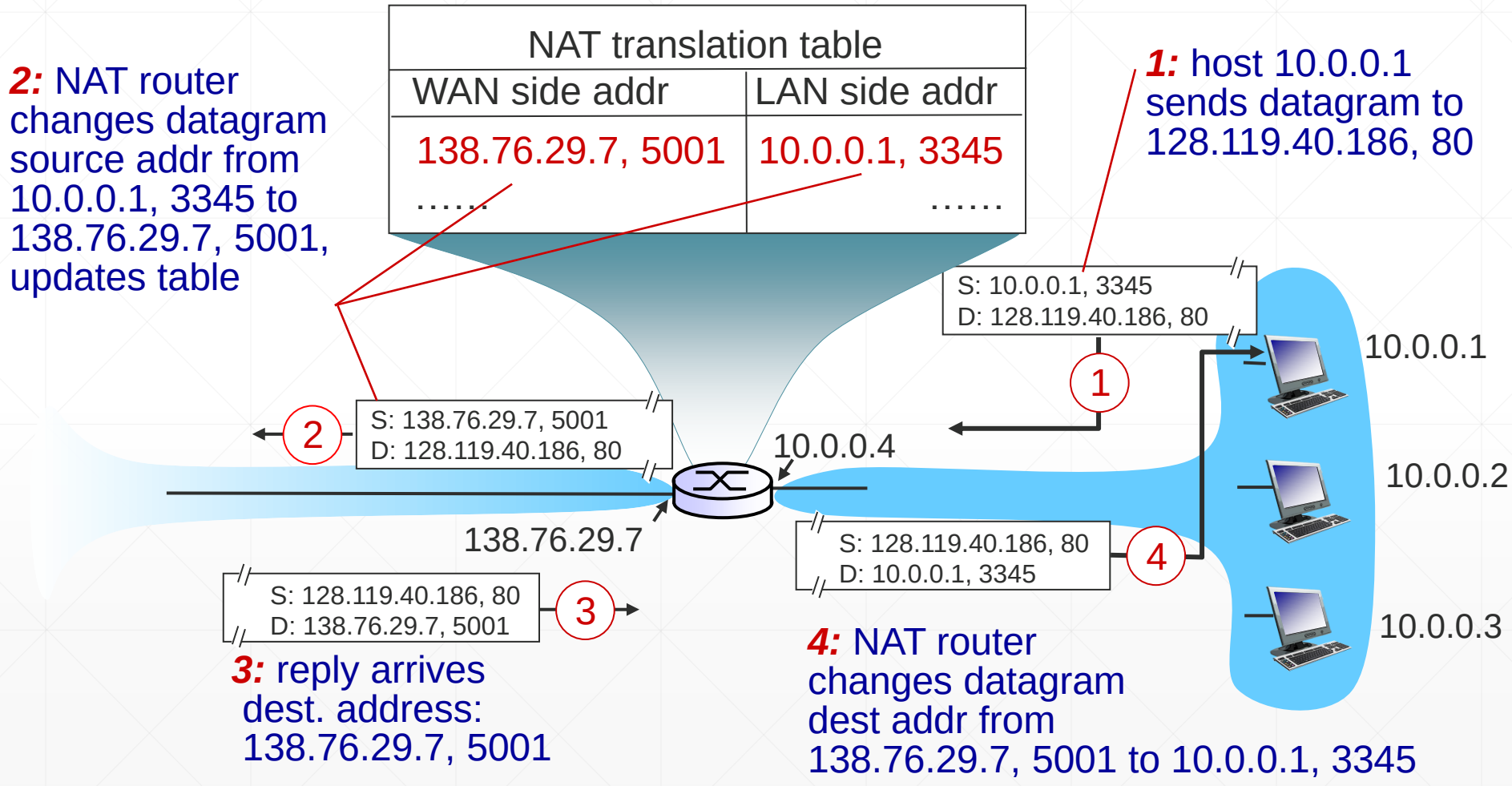
- *outgoing datagrams: replace* (source IP address, port #) of every outgoing datagram to (NAT IP address, new port #)

  . . . 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:
  - 60,000 simultaneous connections with a single LAN-side address!
- NAT is controversial:
  - Port numbers are meant to be used for addressing processes, not addressing hosts
  - Routers are supposed to process packets only upto layer 3
  - Violates end to end argument
  - Use IPv6 for shortage of IP addresses

UPnP

- Universal Plug and Play
- protocol that allows a host to discover and configure a nearby NAT
- an application running in a host can request a NAT mapping between its (private IP address, private port number) and the (public IP address, public port number) for some requested public port number
- If the NAT accepts the request and creates the mapping, then nodes from the outside can initiate TCP connections to (public IP address, public port number)
- UPnP allows external hosts to initiate communication sessions to NATed hosts, using either TCP or UDP

Internet Control Message Protocol (ICMP)

- ICMP, specified in [RFC 792]
  - used by hosts and routers to communicate network-layer information to each other
  - most typical use of ICMP is for error reporting
- ICMP is often considered part of IP but architecturally it lies just above IP
- ICMP messages are carried as IP payload, just as TCP or UDP segments are carried as IP Payload
- ICMP messages have a type and a code field
- contain the header and the first 8 bytes of the IP datagram

ICMP Message Types

ICMP Type	Code	Description
0	0	echo reply (to ping)
3	0	destination network unreachable
3	1	destination host unreachable
3	2	destination protocol unreachable
3	3	destination port unreachable
3	6	destination network unknown
3	7	destination host unknown
4	0	source quench (congestion control)
8	0	echo request
9	0	router advertisement
10	0	router discovery
11	0	TTL expired
12	0	IP header bad

IPv6 : Introduction

- In early 1990s Internet Engineering Task Force(IETF) began to develop a successor to IPv4
- Motivation:
  - Realization that 32-bit IP address space is beginning to be used up
- Solution:
  - Need a large IP address space, a new IP protocol, IPv6

IPv6

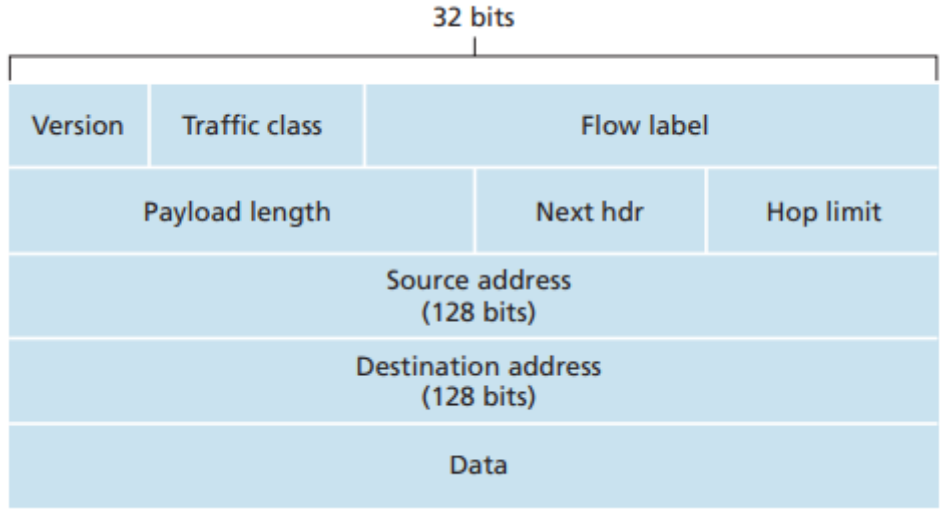
- Important changes in IPv6:
- **Expanded addressing capabilities:**
  - size of the IP address increased from 32 to 128 bits
  - Introduced a new type of address
    - **Anycast** : allows a datagram to be delivered to any one of a group of hosts
- **A streamlined 40-byte header:**
  - a few IPv4 fields have been dropped or made optional
  - 40-byte fixed-length header allows faster processing of the IP datagram

IPv6

- **Flow labelling and priority:**
- IPv6 definition of a **flow** :
  - “labeling of packets belonging to particular flows for which the sender requests special handling, such as a nondefault quality of service or real-time service”
  - For example
    - audio and video transmission might likely be treated as a flow
    - file transfer and e-mail, might not be treated as flows

IPv6 : Datagram Format

- **Version:**
  - 4-bit field identifies the IP version number
  - putting a 4 in this field does not create a valid IPv4 datagram
- **Traffic class:**
  - 8-bit field is similar to the TOS field in IPv4
- **Flow label**
  - this 20-bit field is used to identify a flow of datagrams



IPv6 : Datagram Format

- **Payload length:**
  - This 16-bit value gives the number of bytes in the IPv6 datagram
- **Next header:**
  - identifies the protocol to which the contents (data field) of this datagram will be delivered (for example, to TCP or UDP)
- **Hop limit:**
  - The contents of this field are decremented by one by each router that forwards the datagram
  - If the hop limit count reaches zero, the datagram is discarded.

IPv6 : Datagram Format

- **Source and destination addresses:**
  - The various formats of the IPv6 128-bit addresses are described in RFC 4291
- **Data:**
  - payload portion of the IPv6 datagram
  - When the datagram reaches its destination
    - the payload will be removed from the IP datagram
    - passed on to the protocol specified in the next header field

IPv6

- Few fields in IPv4 datagram are no longer present in the IPv6 datagram
- **Fragmentation/Reassembly:**
  - IPv6 does not allow for fragmentation and reassembly at intermediate routers; it can be performed only by the **source and destination**
  - IPv6 datagram received by a router is too large
    - the router simply drops the datagram
    - sends a “Packet Too Big” ICMP error message **to sender**
- **Header Checksum:**
  - Transport-layer and link-layer protocols perform checksumming, so designers of IP felt that this functionality was redundant

IPv6

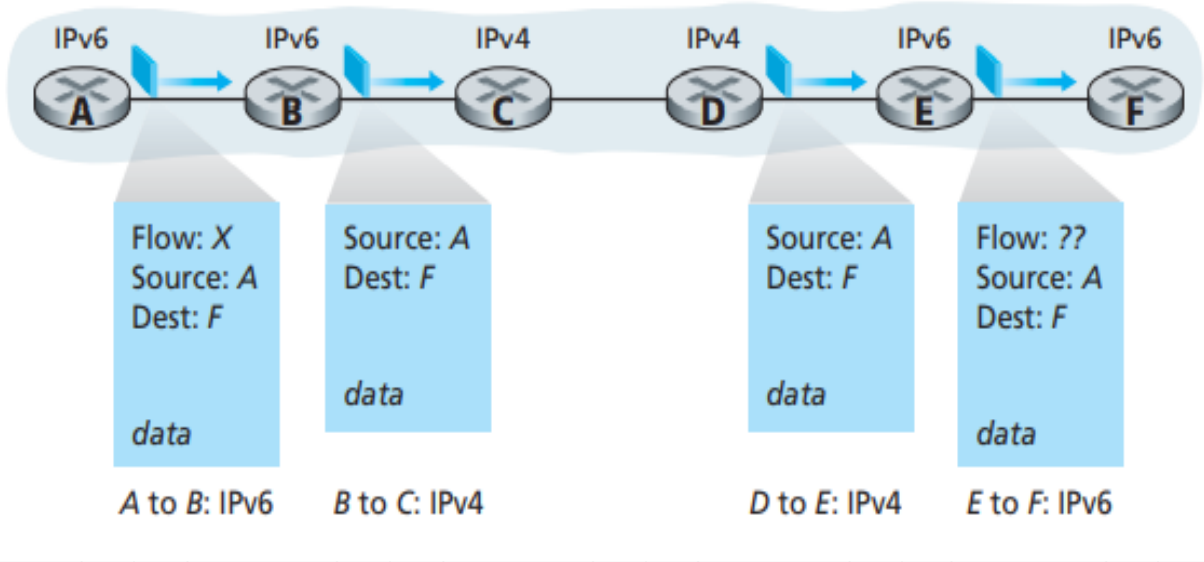
- **Options:**
  - no longer a part of the standard IP header
  - removal of the options field results in a fixed-length, 40- byte IP header
- ICMP protocol is used by IP nodes to report error conditions and provide limited information (for example, the echo reply to a ping message) to an end system
- New version of ICMP
  - ICMPv6 added new types and codes required by the new IPv6 functionality
  - include the “Packet Too Big” type, and an “unrecognized IPv6 options” error code

Transitioning from IPv4 to IPv6

- Already deployed IPv4-capable systems are not capable of handling IPv6 datagrams
- RFC 4213 describes two approaches for gradually integrating IPv6 hosts and routers into an IPv4 world
- **Dual-stack approach**
  - IPv6 nodes also have a complete IPv4 implementation
  - Such nodes are referred as IPv6/IPv4 nodes
  - has the ability to send and receive both IPv4 and IPv6 datagrams

Problem with Dual Stack Approach:

- Node A is IPv6-capable and wants to send an IP datagram to Node F, which is also IPv6-capable
- Node B must create an IPv4 datagram to send to C
- data field of the IPv6 datagram can be copied into the data field of the IPv4 datagram
- While conversion few IPv6 fields will be lost
- arriving IPv4 datagram at E from D do not contain all the fields that were in the original IPv6 datagram sent from A



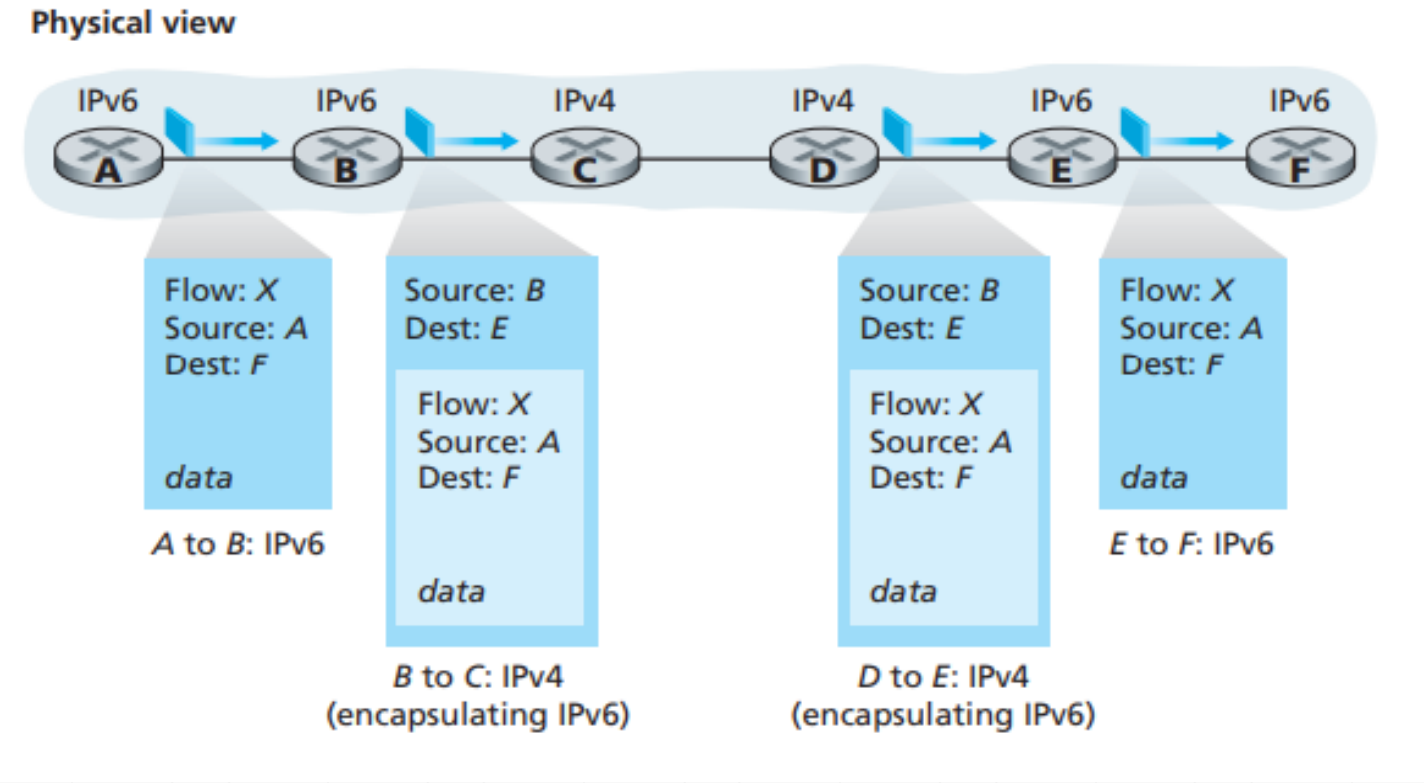
Transitioning from IPv4 to IPv6 : Tunneling

- Alternative to the dual-stack approach is **Tunneling**
- Basic idea
- Suppose two IPv6 want to interoperate using IPv6 datagrams but are connected to each other by intervening IPv4 routers
- the intervening set of IPv4 routers between two IPv6 routers is a **tunnel**
- With tunneling, the IPv6 node on the sending side of the tunnel takes the entire IPv6 datagram and puts it in the data (payload) field of an IPv4 datagram
- This IPv4 datagram is then addressed to the IPv6 node on the receiving side of the tunnel

Transitioning from IPv4 to IPv6 : Tunneling

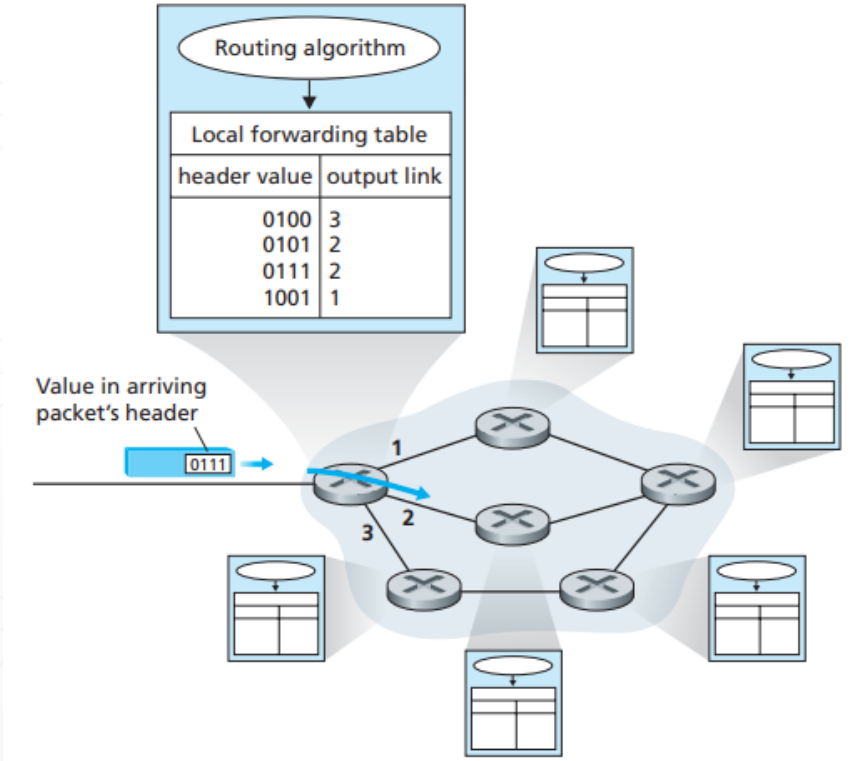
- Routers will be unaware that the IPv4 datagram itself contains a complete IPv6 datagram
- IPv6 node on the receiving side of the tunnel receives the IPv4 datagram
- determines that the IPv4 datagram contains an IPv6 datagram
- extracts the IPv6 datagram and then routes the IPv6 datagram

Transitioning from IPv4 to IPv6 : Tunneling



Routing Algorithms

- **Forwarding function:**
- when a packet arrives to a router, the router indexes a forwarding table and determines the link interface to which the packet is to be directed
- Routing algorithms, operating in network routers, exchange and compute the information that is used to configure these forwarding tables
- **Routing function:**
- determining the path, a packets takes from sender to receiver



Routing Algorithms

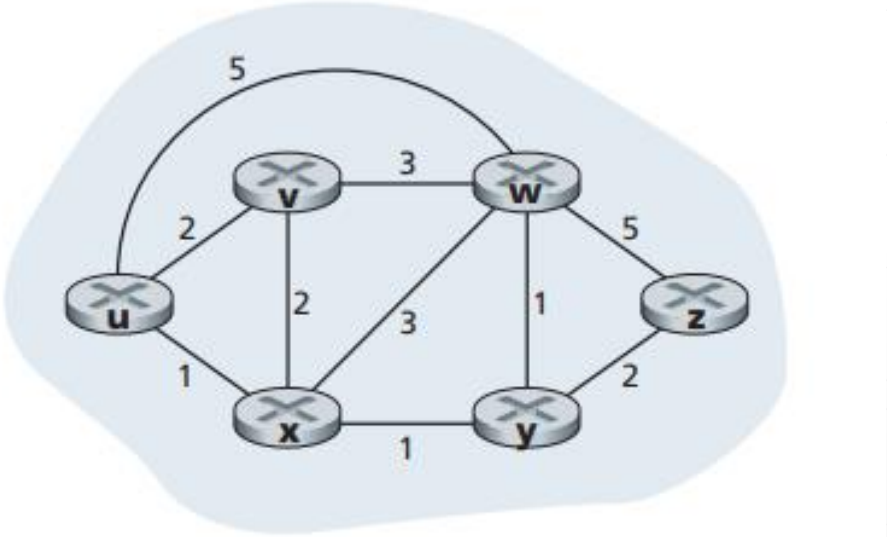
- A host is attached directly to one router, the **default router** for the host also called the **first-hop router** for the host
- the default router of the source host is the **source router**
- the default router of the destination host is the **destination router**
- **Purpose of a Routing algorithm:**
  - given a set of routers with links connecting the routers, a routing algorithm finds a “good” path from source router to destination router
  - a good path is one that has the least cost

Routing Algorithms

- A graph is used to formulate routing problems
- Graph G = (N,E) is a set N of nodes and a collection E of edges, where each edge is a pair of nodes from N
- **Nodes** in the graph represent **routers**—the points at which packet-forwarding decisions are made
- **Edges** connecting these nodes represent the **physical links** between routers

Routing Algorithms

- Edge has a value representing its **cost**
- For any edge (x,y) in E,
  - c(x,y) as the cost of the edge between nodes x and y
  - if the pair (x,y) does not belong to E, we set c(x,y) = ∞
- Consider undirected graphs whose edges do not have a direction
  - so that edge (x,y) is the same as edge (y,x) and
  - c(x,y) = c(y,x)



Routing Algorithms

- **One way** of classification is whether they are global or decentralized
- **Global routing algorithm**
  - computes the least-cost path between a source and destination using complete, global knowledge about the network
  - algorithm takes the connectivity between all nodes and all link costs as inputs
  - **Ex** : Link-state (LS) algorithm
- **Decentralized routing algorithm**
  - the calculation of the least-cost path is carried out in an iterative, distributed manner
  - no node has complete information about the costs of all network links
  - **Ex :** Distance vector (DV) algorithm

Routing Algorithms

- **Second way** of classification is whether they are static or dynamic
- **Static routing algorithms**, routes change very slowly over time, often as a result of human intervention (for example, a human manually editing a router's forwarding table)
- **Dynamic routing algorithms** change the routing paths as the network traffic loads or topology change
- **Third way** of classification is whether they are load-sensitive or load-insensitive
- In **Load-sensitive algorithm**, link costs vary dynamically to reflect the current level of congestion in the underlying link
- Today's Internet routing algorithms are **Load-insensitive**, as a link's cost does not explicitly reflect its current (or recent past) level of congestion

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 destinations

notation:

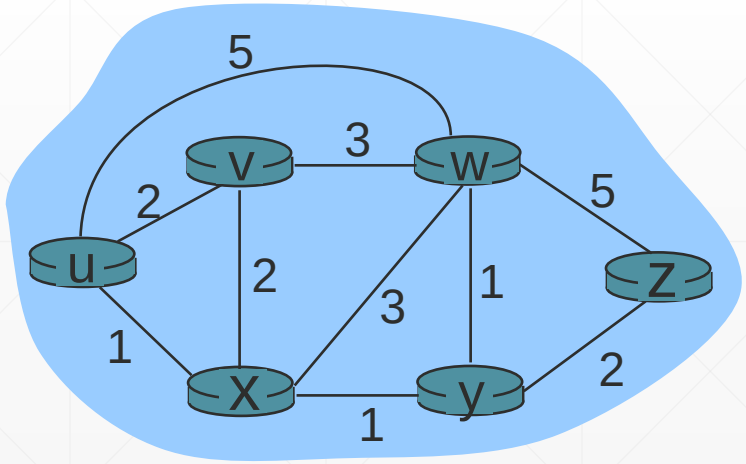
- c(x,y): link cost from node x to y; = ∞ if not direct neighbors
- D(v): current value of cost of path from source to dest. v
- p(v): predecessor node along path from source to v
- N': set of nodes whose least cost path definitively known

Dijsktra’ s Algorithm

```
1  Initialization:
2    N' = {u}
3    for all nodes v
4      if v adjacent to u
5          then D(v) = c(u,v)
6      else D(v) = ∞
7
8   Loop
9     find w not in N' such that D(w) is a minimum
10     add w to N'
11     update D(v) for all v adjacent to w and not in N' :
12        D(v) = min( D(v), D(w) + c(w,v) )
13     /* new cost to v is either old cost to v or known
14      shortest path cost to w plus cost from w to v */
15  until all nodes in N'
```

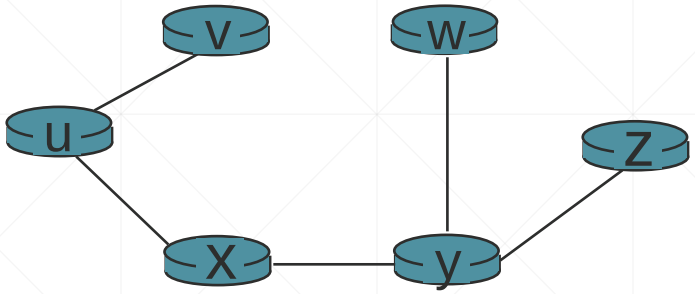
Dijkstra’s algorithm : Example

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:

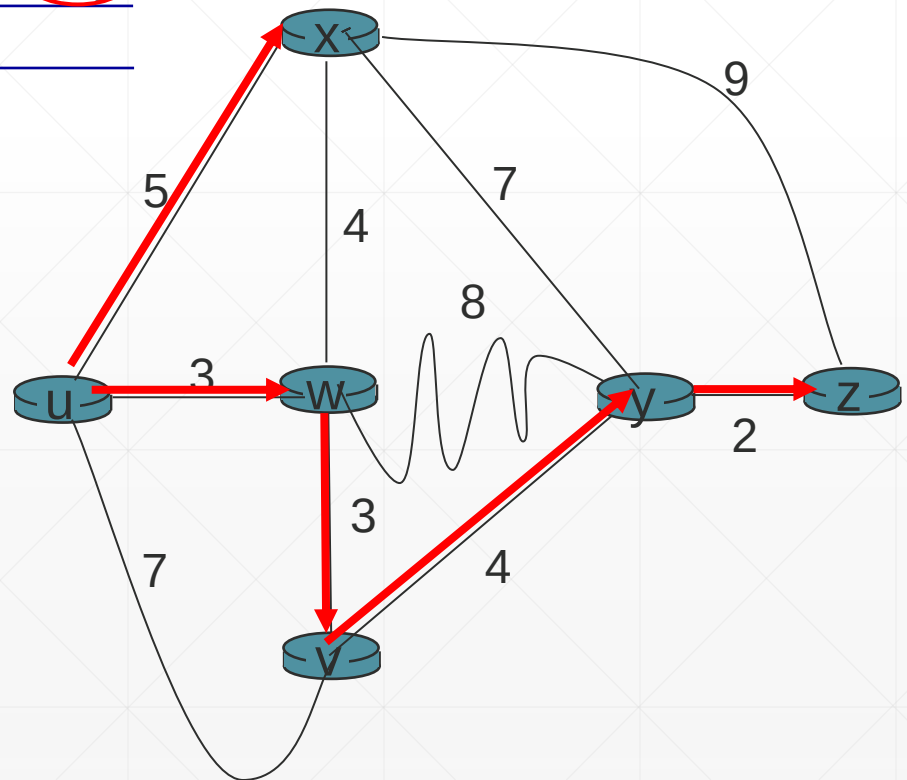
destination	link
v	(u,v)
x	(u,x)
y	(u,x)
w	(u,x)
z	(u,x)

Dijkstra's algorithm: another 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	7,u	3,u	5,u	∞	∞
1	uw	6,w		5,u	11,w	∞
2	uwx	6,w			11,w	14,x
3	uwxv				10,v	14,x
4	uwxvy					12,y
5	uwxvyz					

notes:

- construct shortest path tree by tracing predecessor nodes
- ties can exist (can be broken arbitrarily)



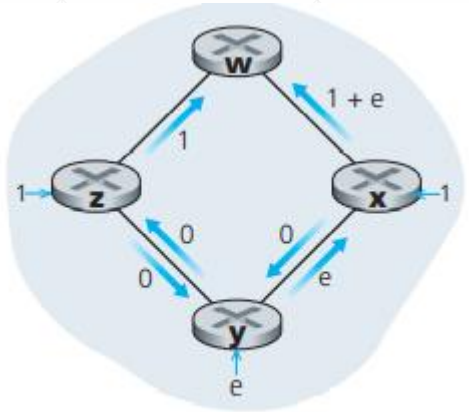
Dijkstra's algorithm, discussion

algorithm complexity: n nodes

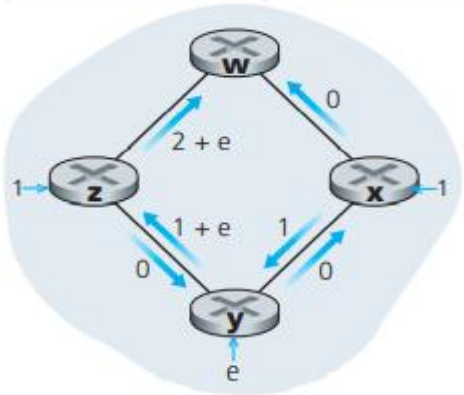
- each iteration: need to check all nodes, w, not in N
- $n(n+1)/2$ comparisons: $O(n^2)$

oscillations possible:

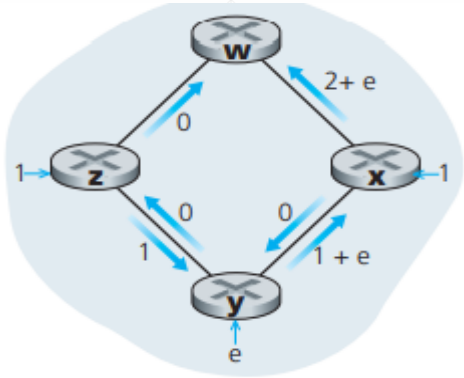
- e.g., support link cost equals amount of carried traffic:



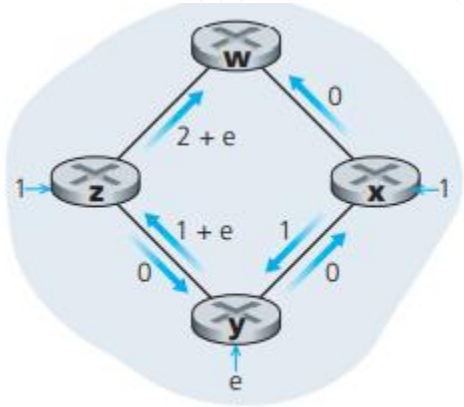
a. Initial routing



b. *x, y* detect better path to *w*, clockwise



c. *x, y, z* detect better path to *w*, counterclockwise



d. *x, y, z,* detect better path to *w*, clockwise

Distance Vector (DV) Routing Algorithm

- The distance vector (DV) algorithm is iterative, asynchronous, and distributed
- **Distributed**
  - each node receives some information from one or more of its directly attached neighbors, performs a calculation, and then distributes the results of its calculation back to its neighbors
- **Iterative**
  - this process continues until no more information is exchanged between neighbors
  - the algorithm is also self-terminating
- **Asynchronous**
  - it does not require all of the nodes to operate in lockstep with each other.

Distance vector algorithm

Bellman-Ford equation (dynamic programming)

let

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

then

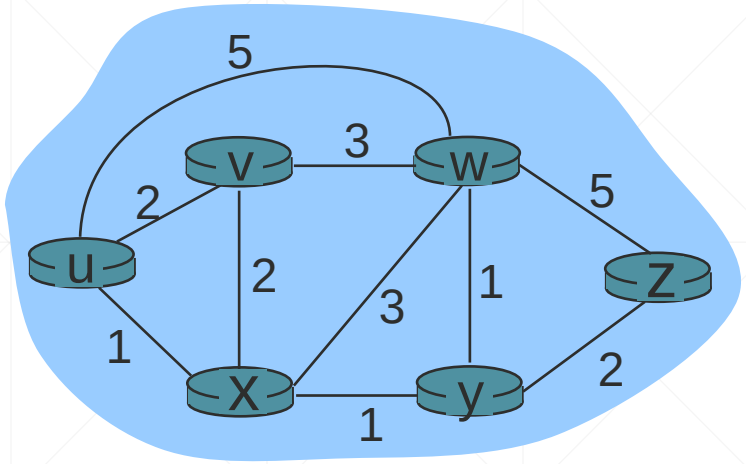
$$d_x(y) = min_v \{c(x,v) + d_v(y)\}$$

$c(x,v)$: cost to neighbor v

$d_v(y)$: cost from neighbor v to destination y

min taken over all neighbors v of x

Bellman-Ford example



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

B-F equation says:

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

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

Distance vector algorithm

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

Distance vector algorithm

key idea:

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

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

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

Distance vector algorithm

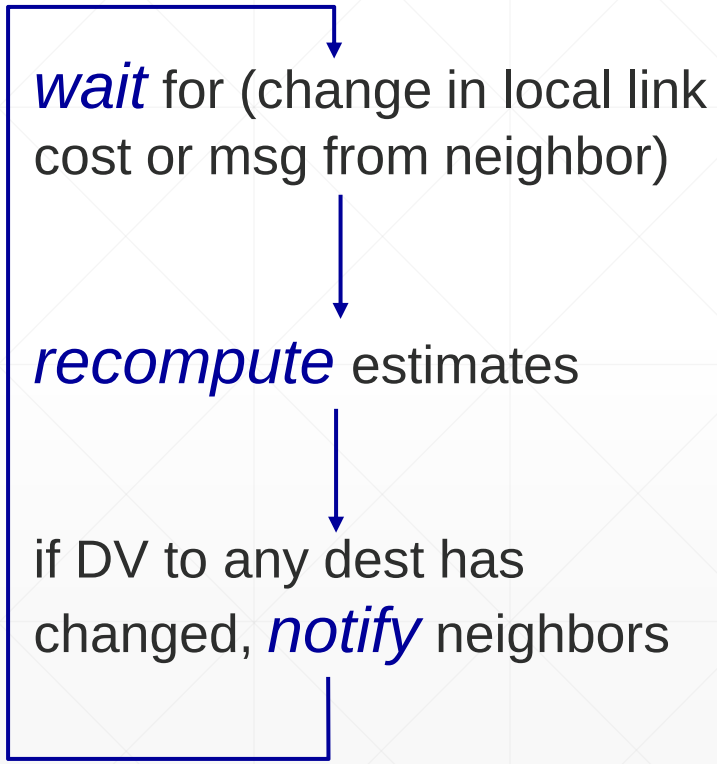
iterative, asynchronous: each local iteration caused by:

- local link cost change
- DV update message from neighbor

distributed:

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

each node:



Distance-Vector (DV) Algorithm

1. Each node waits for an update from any neighbor (Lines 10–11)
2. calculates its new distance vector when receiving an update (Line 14)
3. and distributes its new distance vector to its neighbors (Lines 16–17)

At each node, x:

```
1  Initialization:
2     for all destinations y in N:
3        D_x(y) = c(x,y)    /* if y is not a neighbor then c(x,y) = ∞ */
4     for each neighbor w
5        D_w(y) = ? for all destinations y in N
6     for each neighbor w
7        send distance vector D_x = [D_x(y): y in N] to w
8
9  loop
10    wait (until I see a link cost change to some neighbor w or
11          until I receive a distance vector from some neighbor w)
12
13    for each y in N:
14       D_x(y) = min_v{c(x,v) + D_v(y)}
15
16    if D_x(y) changed for any destination y
17       send distance vector D_x = [D_x(y): y in N] to all neighbors
18
19 forever
```

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

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

node x table

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

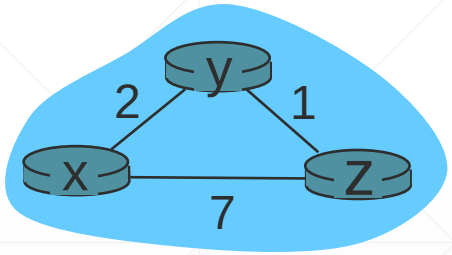
from \ cost to	x	y	z
x	0	2	3
y	2	0	1
z	7	1	0

node y table

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

node z table

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



time

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

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

node x table

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

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

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

node y table

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

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

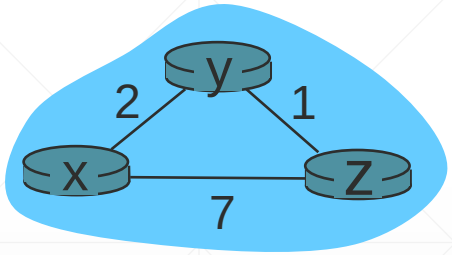
from \ cost to	x	y	z
x	0	2	3
y	2	0	1
z	3	1	0

node z table

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

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

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



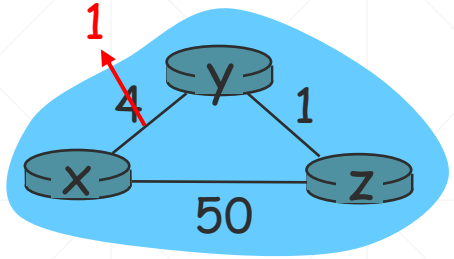
time

- In the DV algorithm
  - a node x updates its distance-vector estimate
    - when it either sees a cost change in one of its directly attached links or
    - receives a distance vector update from some neighbor
- to update its own forwarding table for a given destination y
  - x needs to know the neighboring node v*(y) that is the next-hop router along the shortest path to y
  - v*(y) can be any of the minimizing neighbors
  - Lines 13–14

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



"good news travels fast"

t_0 : *y* detects link-cost change, updates its DV, informs its neighbors.

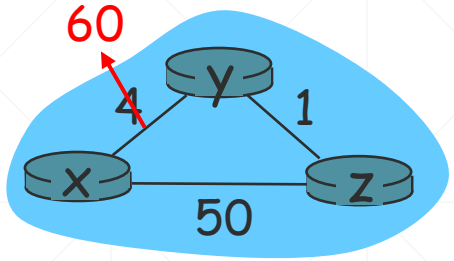
t_1 : *z* receives update from *y*, updates its table, computes new least cost to *x* , sends its neighbors its DV.

t_2 : *y* receives *z*'s update, updates its distance table. *y*'s least costs do *not* change, so *y* does *not* send a message to *z*.

Distance vector: link cost changes

link cost changes:

- node detects local link cost change
- *bad news travels slow* - "count to infinity" problem!
- 44 iterations before algorithm stabilizes



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?

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
  - error propagate thru network

Hierarchical routing

Our Study:

- Viewed network as collection of interconnected routers
- All routers execute same routing algorithm to compute routing paths
- Its simplistic for 2 reasons:

scale: more no. of routers, overhead involved in computing, sorting, comm. Info. , becomes prohibitive.

- Storing routing info requires huge amount of memory
- LS updates will leave no bandwidth for sending data packets

administrative autonomy : researchers tend to ignore issues such as company's desire to run its router or hide its network's organization.

- An organization should be able to run and administer its network as it wishes.

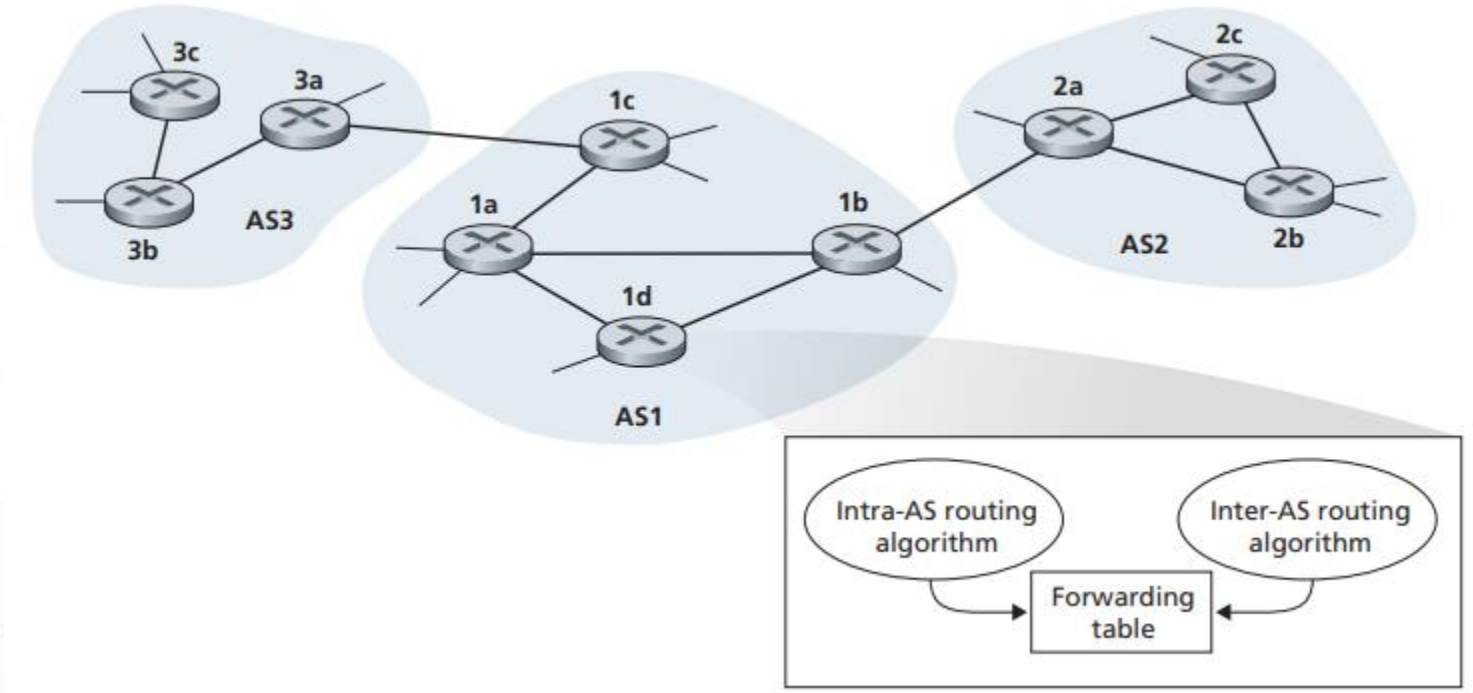
Hierarchical routing

- Problems can be solved by aggregating 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:

- It’s necessary to connect ASs to each other , thus one or more routers in AS will be responsible for forwarding packets to destinations outside the AS.
  - These routers are **Gateway router**
- at “edge” of its own AS
- has link to router in another AS

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

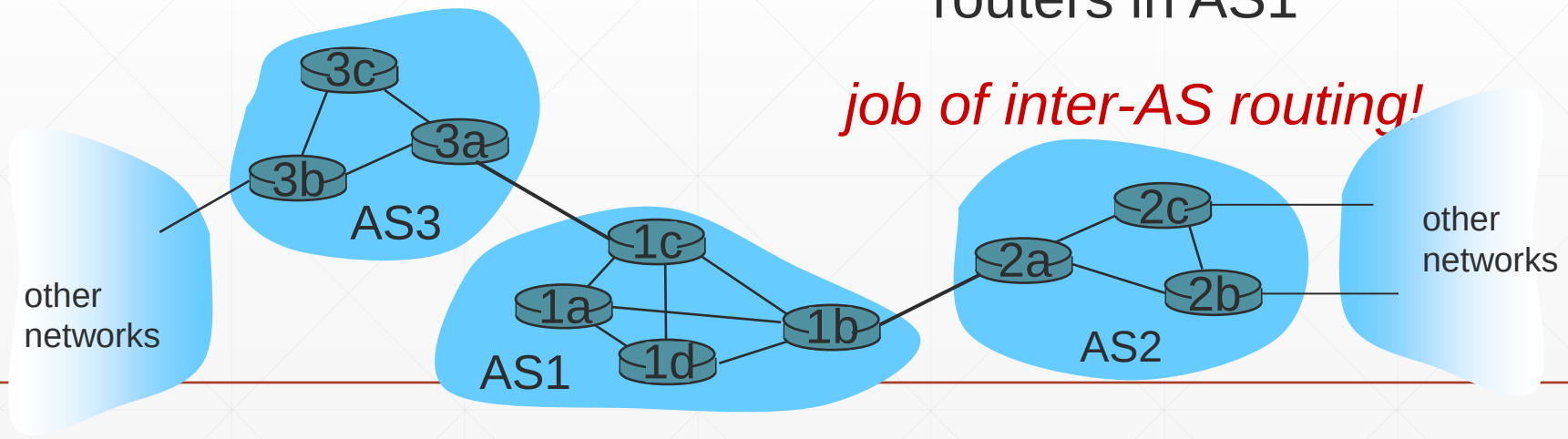
Inter-AS tasks

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

AS1 must:

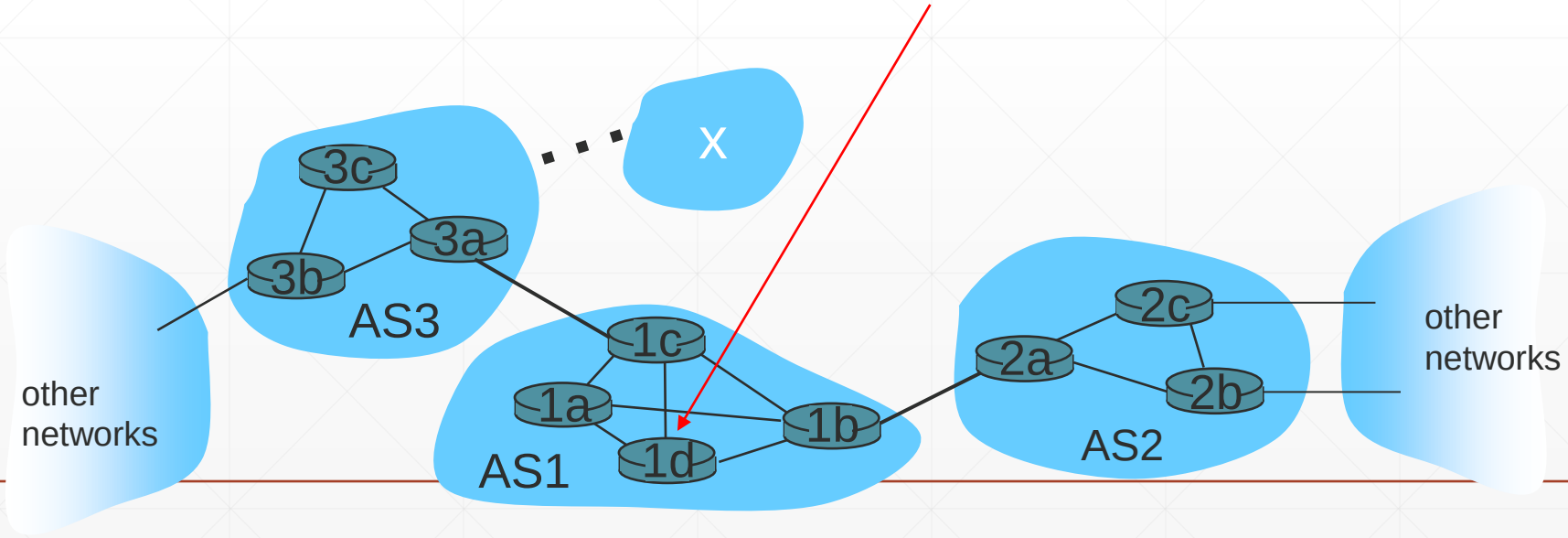
1. learn which dests are reachable through AS2, which through AS3
2. propagate this reachability info to all routers in AS1

job of inter-AS routing!



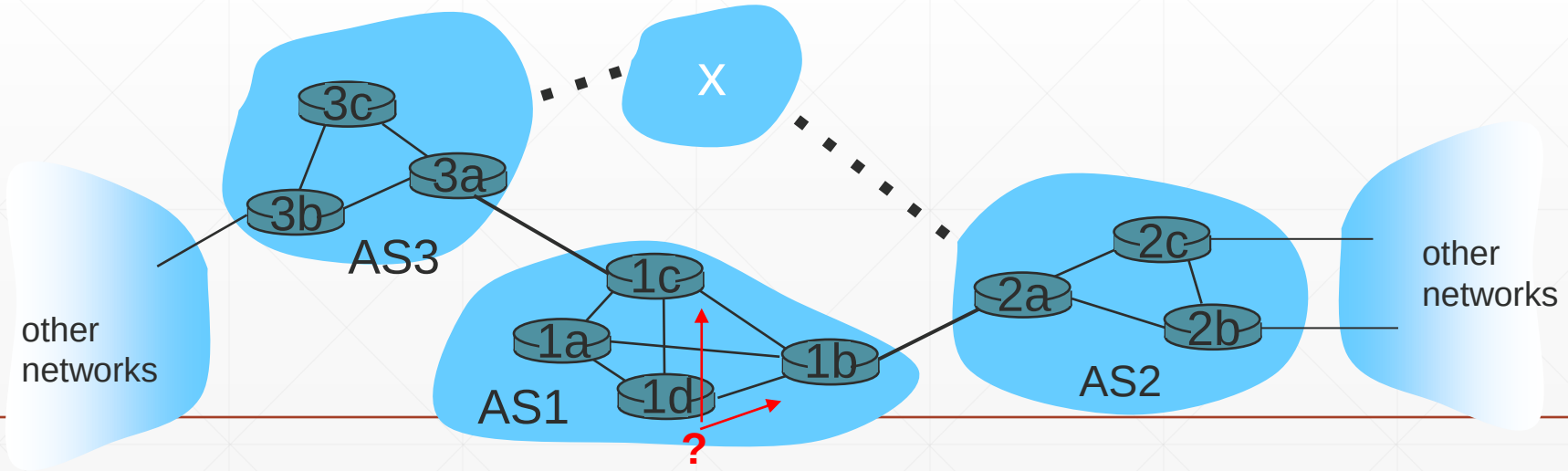
Example: setting forwarding table in router 1d

- 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 *I* is on the least cost path to 1c
  - installs forwarding table entry *(x,I)*



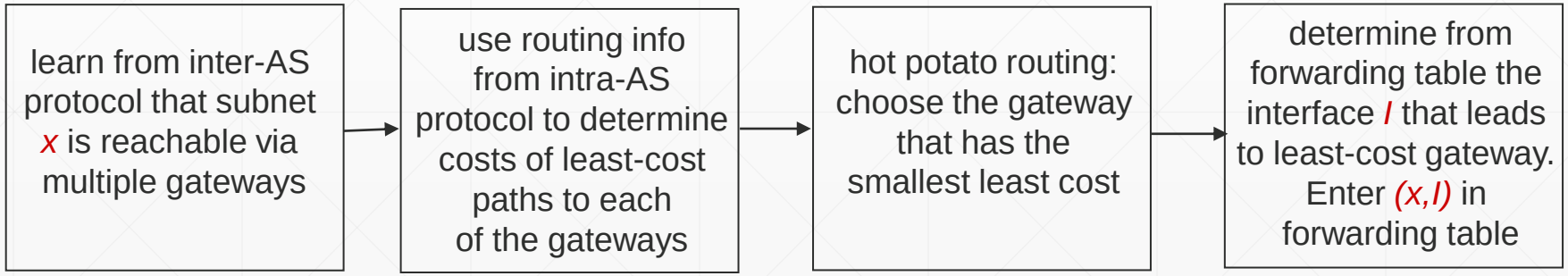
Example: choosing among multiple ASes

- 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 which gateway it should forward packets towards 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 1d 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.




Thank
you