School of Computing Science
Simon Fraser University

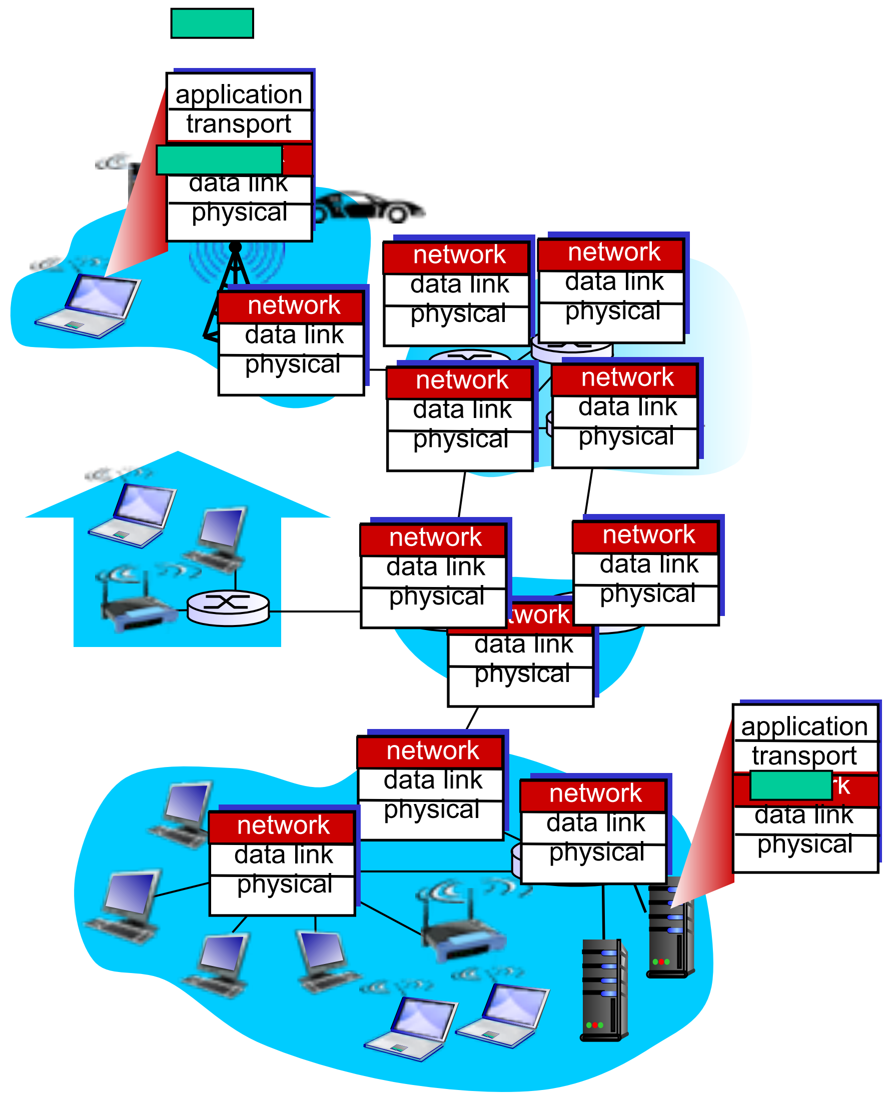
CMPT 471: Networking II

Network Layer, Multicast, Overlays and P2P Systems

Instructor: Mohamed Hefeeda

Network layer

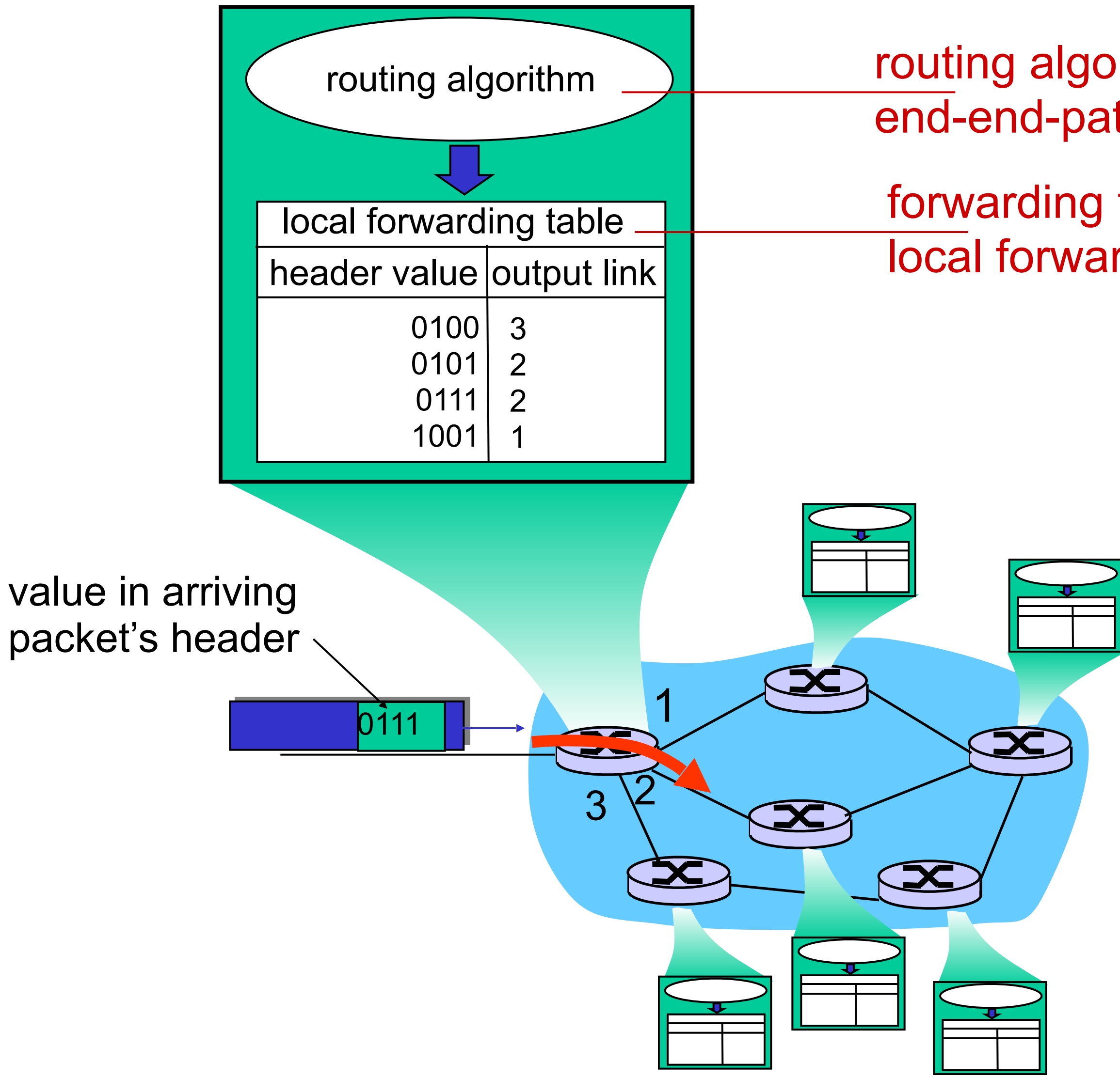
- transport segment from sending to receiving host
- on sending side encapsulates segments into datagrams
- on receiving side, delivers segments to transport layer
- network layer protocols in *every* host, router
- router examines header fields in all IP datagrams passing through it



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*

Interplay between routing and forwarding



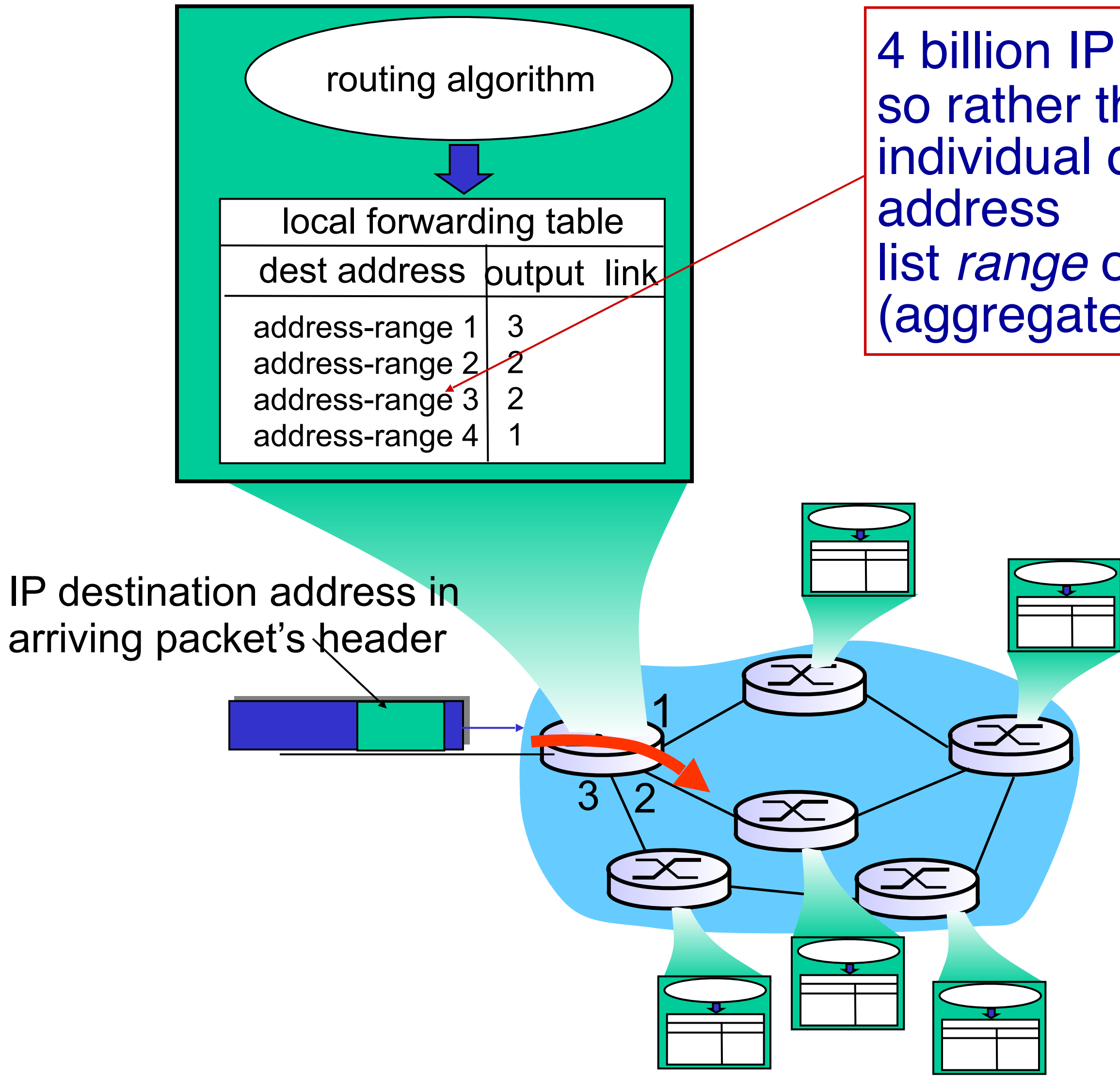
header value	output link
0100	3
0101	2
0111	2
1001	1

routing algorithm determines end-end-path through network

forwarding table determines local forwarding at this router

Datagram forwarding table

4 billion IP addresses, so rather than list individual destination address
list *range* of addresses (aggregate table entries)



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

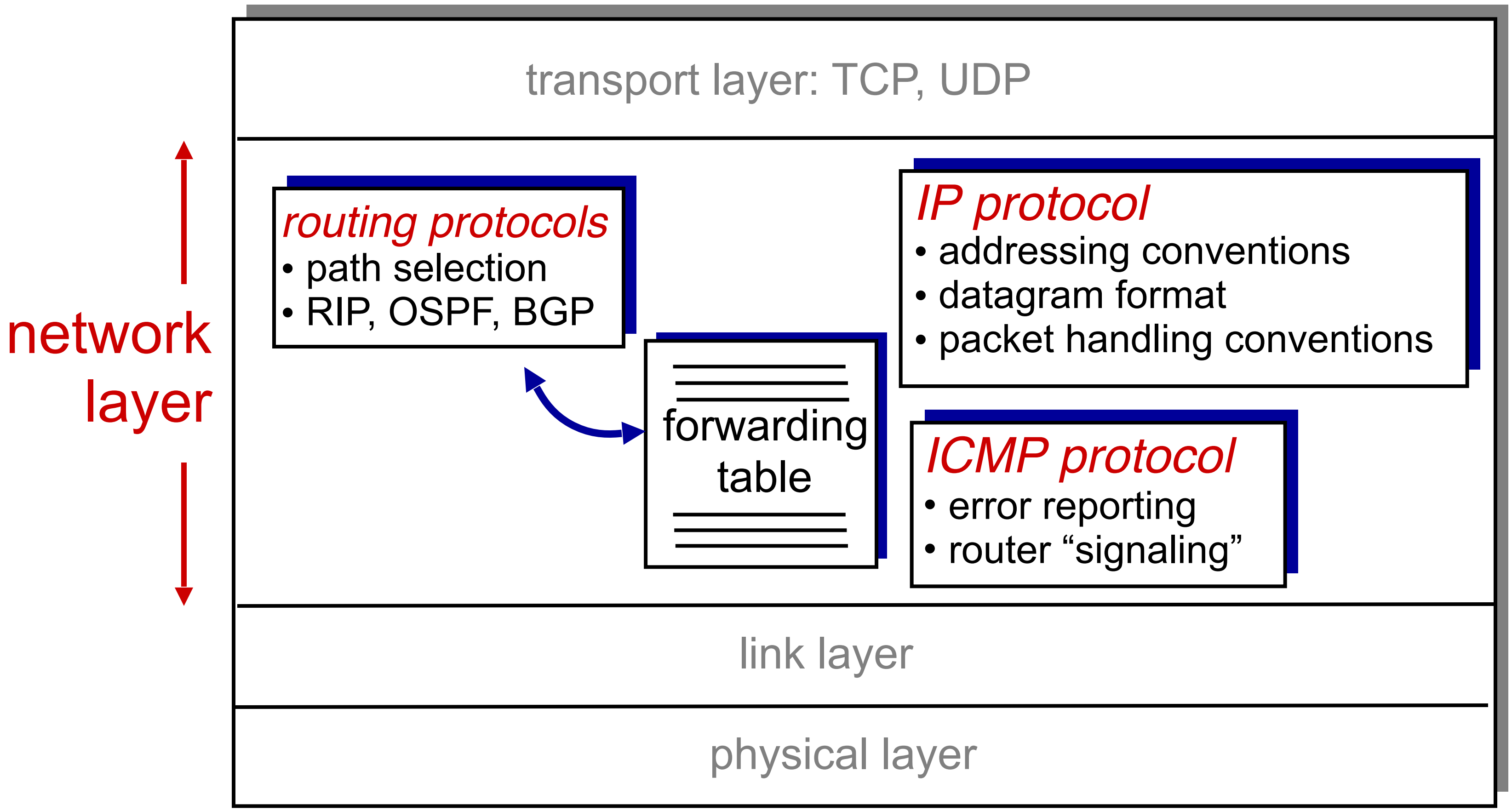
examples:

DA: 11001000 00010111 00010110 10100001 which interface?

DA: 11001000 00010111 00011000 10101010 which interface?

The Internet network layer

host, router network layer functions:



IP datagram format

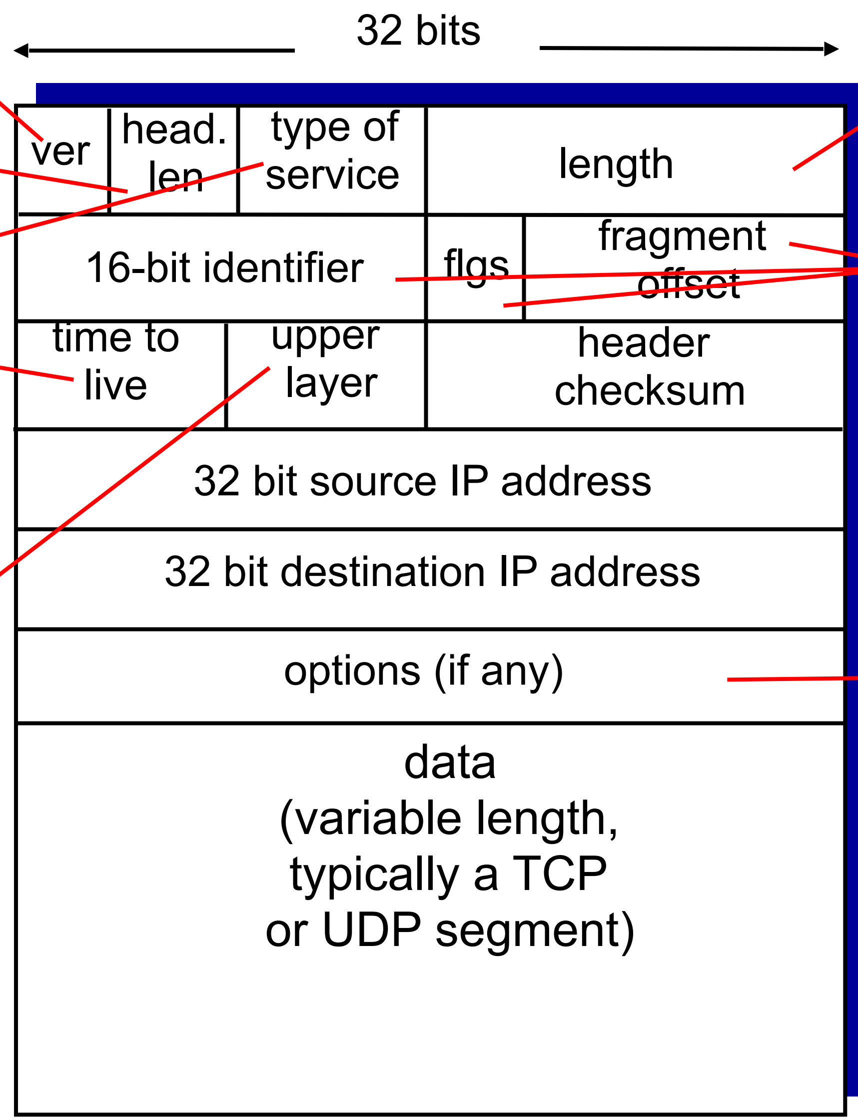
IP protocol version number

header length (bytes)

“type” of data

max number remaining hops (decremented at each router)

upper layer protocol to deliver payload to



total datagram length (bytes)

for fragmentation/ reassembly

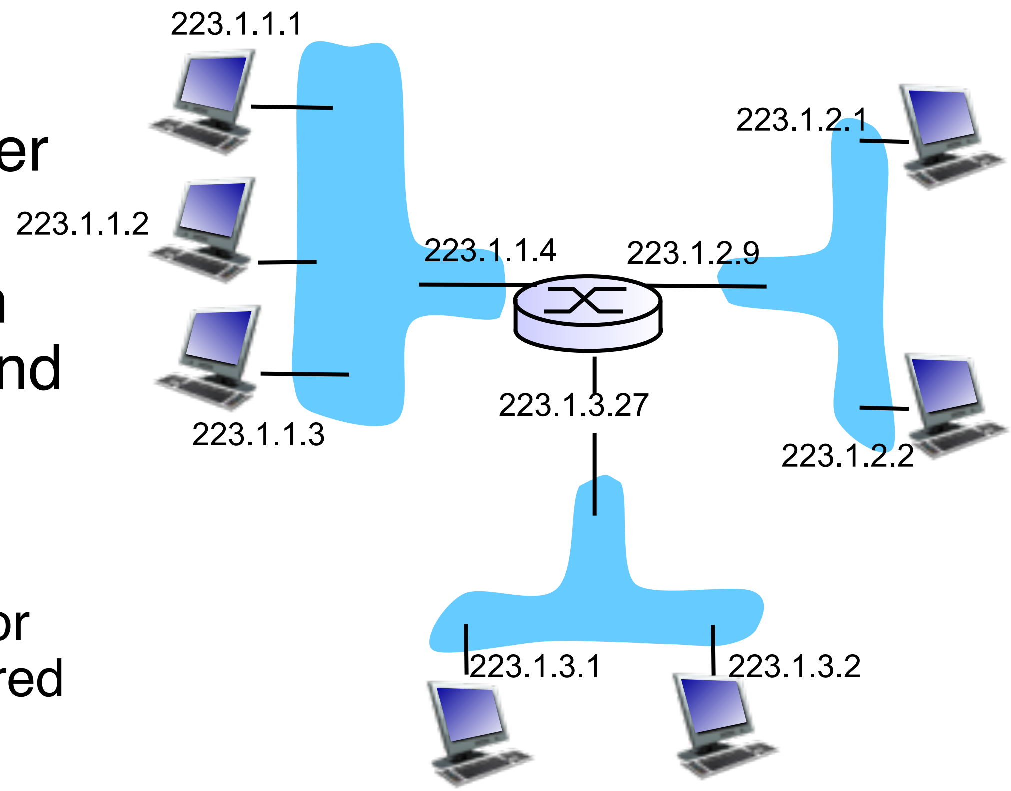
e.g. timestamp, record route taken, specify list of routers to visit.

how much overhead?

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

IP addressing: introduction

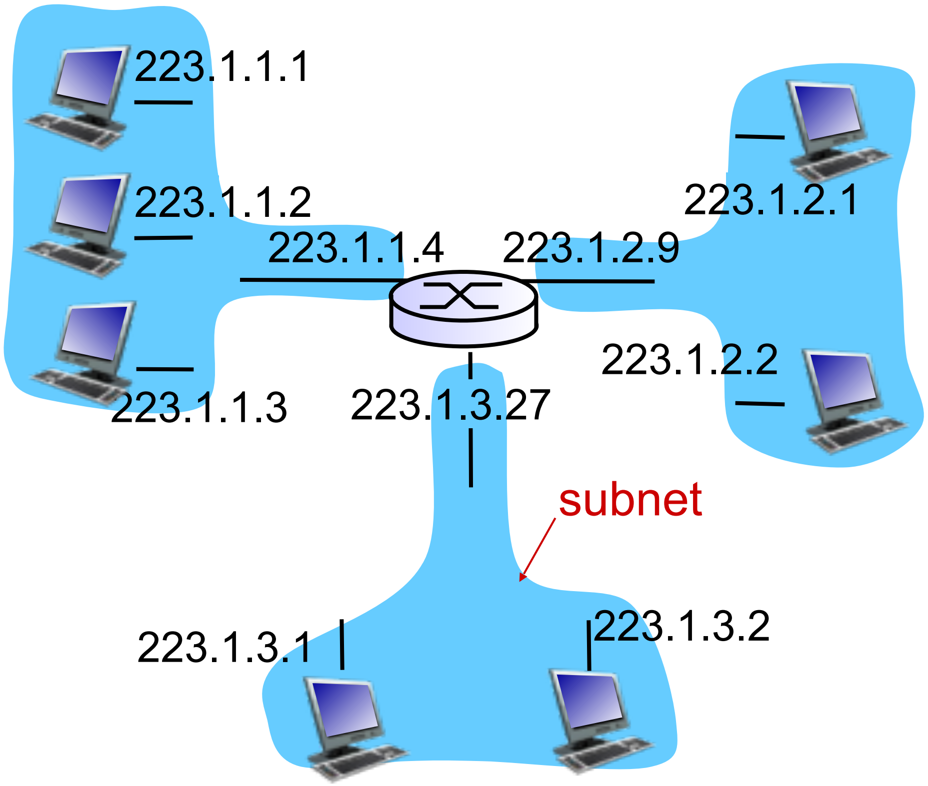
- *IP address:* 32-bit identifier for host, router *interface*
- *interface:* connection between host/router and physical link
  - router's typically have multiple interfaces
  - host typically has one or two interfaces (e.g., wired Ethernet, wireless 802.11)
- *IP addresses associated with each interface*



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

Subnets

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



network consisting of 3 subnets

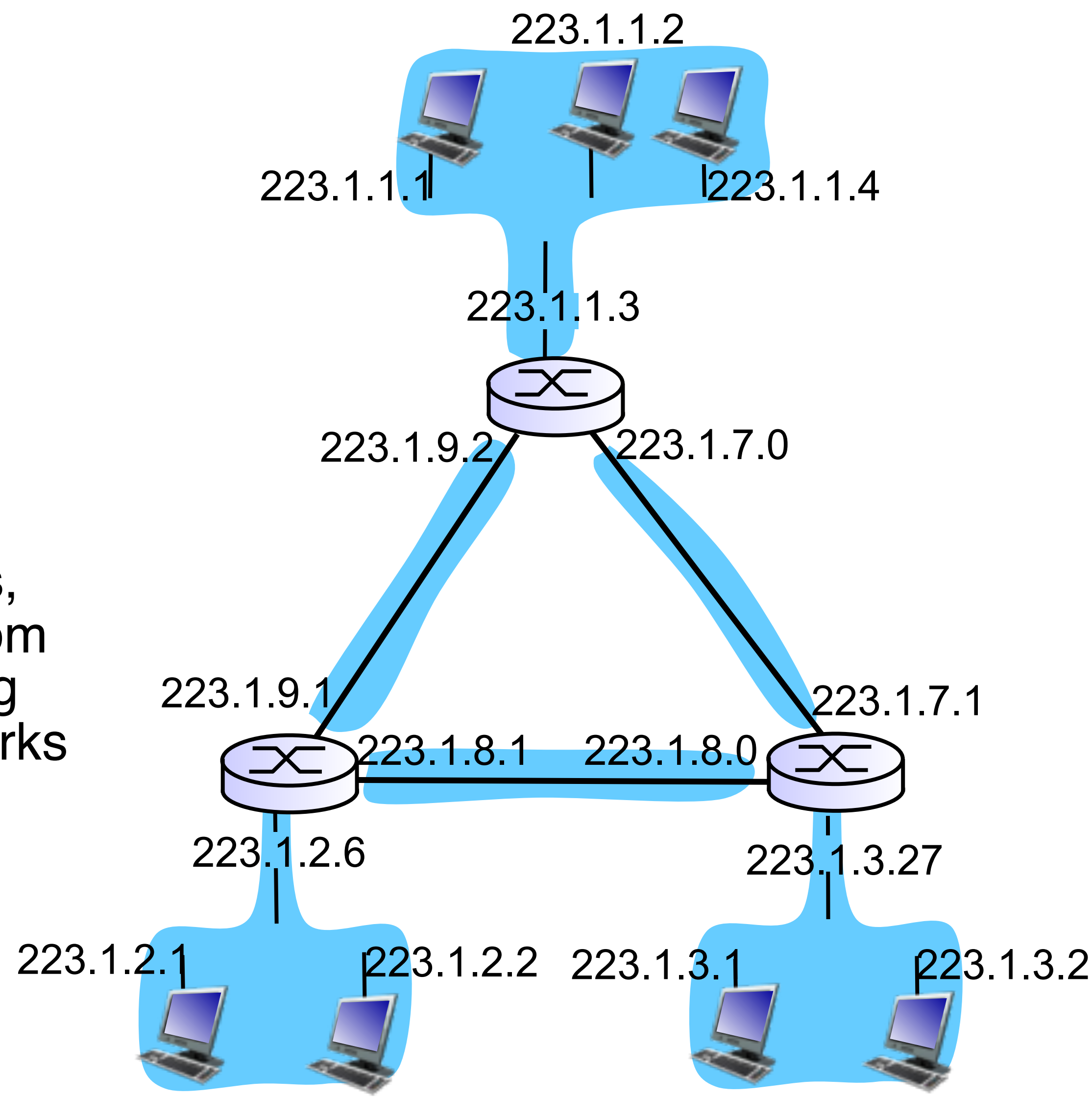
Subnets

how many?

- 6

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*



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 (called mask)

subnet part | host part

11001000 00010111 00010000 00000000

200.23.16.0/23

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

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

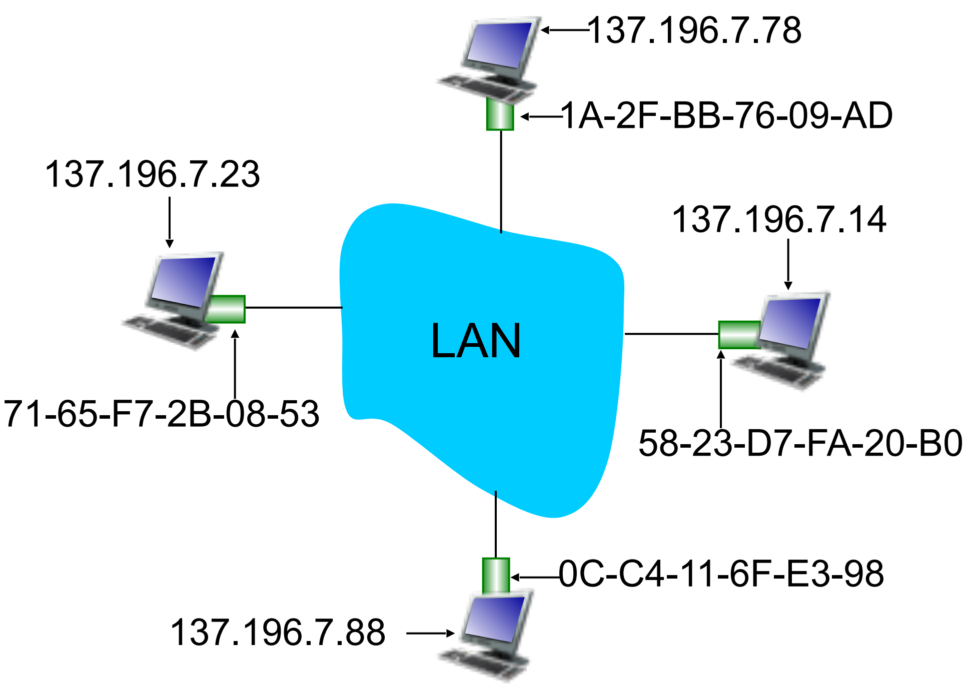
MAC addresses and ARP

- 32-bit IP address:
  - *network-layer* address for interface
  - used for layer 3 (network layer) forwarding
- MAC (or LAN or physical or Ethernet) address:
  - *used 'locally" to get frame from one interface to another physically-connected interface (same network, in IP-addressing sense)*
  - 48 bit MAC address (for most LANs) burned in NIC ROM, also sometimes software settable
  - e.g.: 1A-2F-BB-76-09-AD

hexadecimal (base 16) notation
(each "number" represents 4 bits)

ARP: address resolution protocol

ARP: Maps IP address to MAC address



ARP table: each IP node (host, router) on LAN has table

- IP/MAC address mappings for some LAN nodes:

  < IP address; MAC address; TTL>

- TTL (Time To Live): time after which address mapping will be forgotten (typically 20 min)

ARP protocol: same LAN

- A wants to send datagram to B
  - B's MAC address not in A's ARP table.
- A broadcasts ARP query packet, containing B's IP address
  - dest MAC address = FF-FF-FF-FF-FF-FF
  - all nodes on LAN receive ARP query
- B receives ARP packet, replies to A with its (B's) MAC address
  - frame sent to A's MAC address (unicast)
- A caches (saves) IP-to-MAC address pair in its ARP table until information becomes old (times out)
  - soft state: information that times out (goes away) unless refreshed
- ARP is "plug-and-play":
  - nodes create their ARP tables ***without*** *intervention from net administrator*

IPv6: motivation

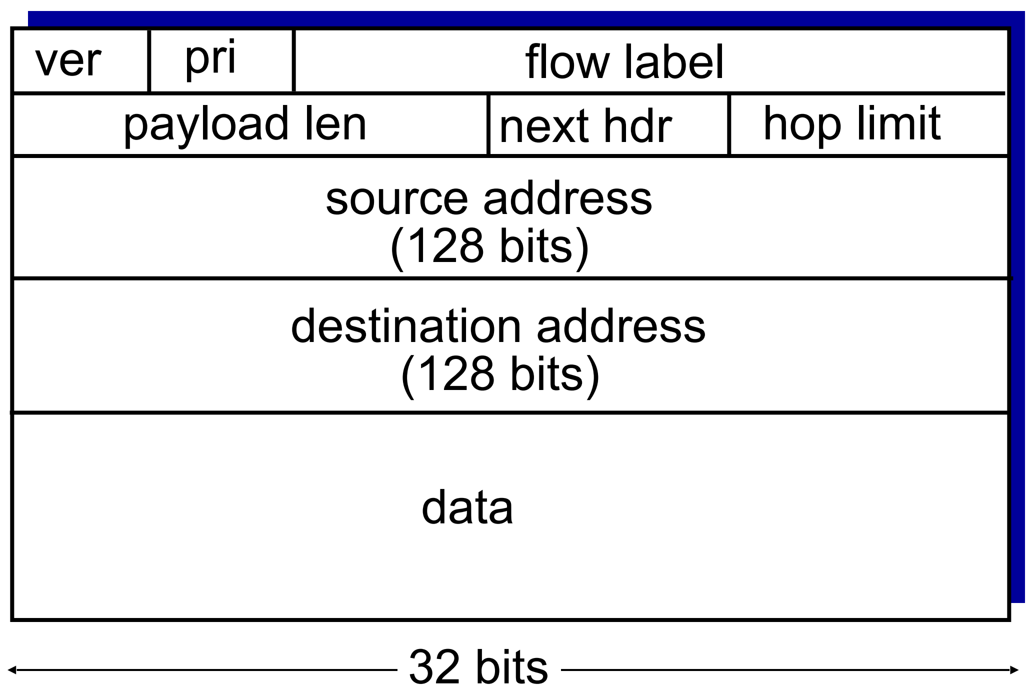
- *initial motivation:* 32-bit address space soon to be completely allocated.
- additional motivation:
  - header format helps speed processing/forwarding
  - header changes to facilitate QoS

IPv6 datagram format:

- fixed-length 40 byte header
- no fragmentation allowed

IPv6 datagram format

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

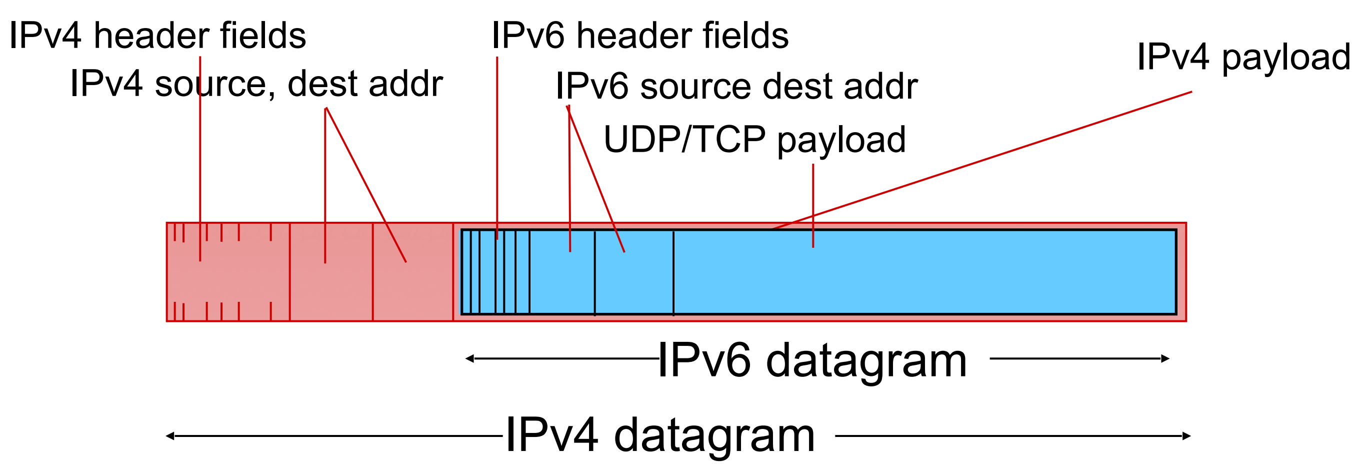


Other changes from IPv4

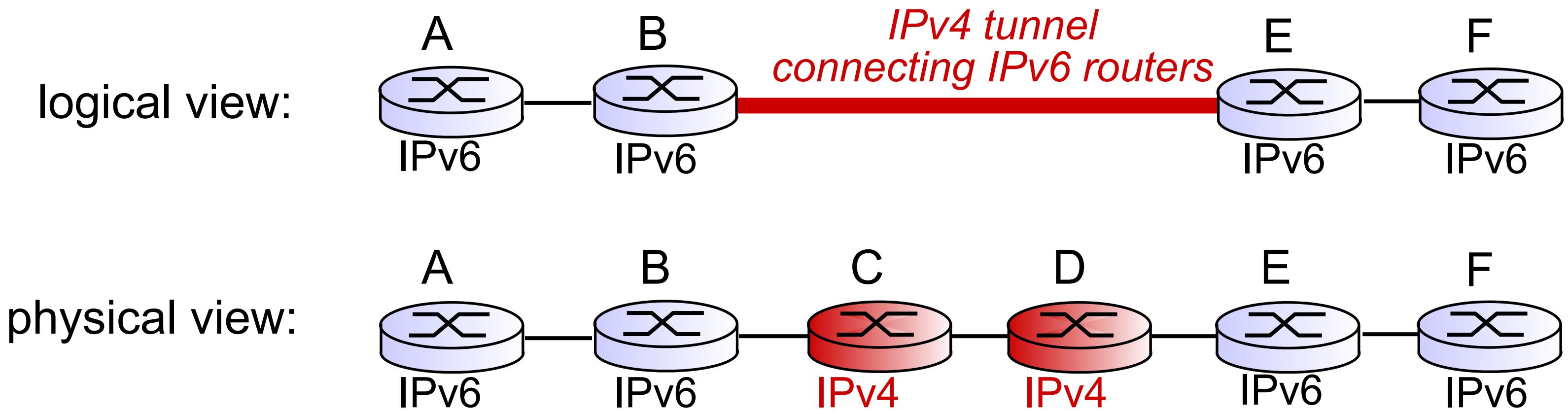
- *checksum*: removed entirely to reduce processing time at each hop
- *options:* allowed, but outside of header, indicated by “Next Header” field
- *ICMPv6:* new version of ICMP
  - additional message types, e.g. “Packet Too Big”
  - multicast group management functions

Transition from IPv4 to IPv6

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

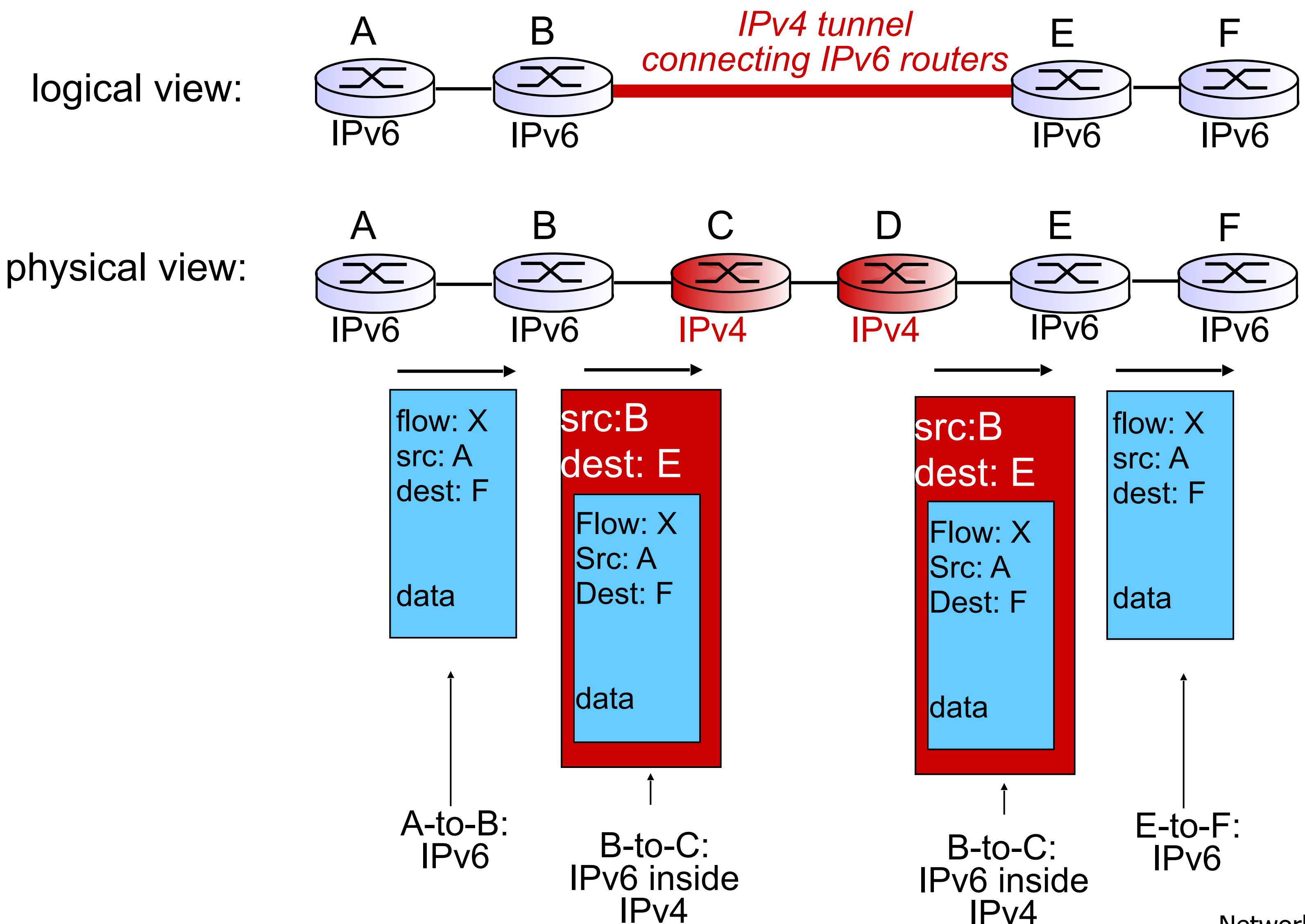


Tunneling



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

Tunneling



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

Routing Algorithms

- **Needed to populate forwarding tables**

- **They run in “control plane”**
  - Typically invoked in the order of 10s of seconds or whenever a change in network topology happens

  - They are much slower than forwarding algorithms that run in “control plane” at “wire speed” (micro/nano seconds)

Routing Algorithms

- **Problem solved by routing algorithms:**

Find optimal path between any two points in the network (graph)

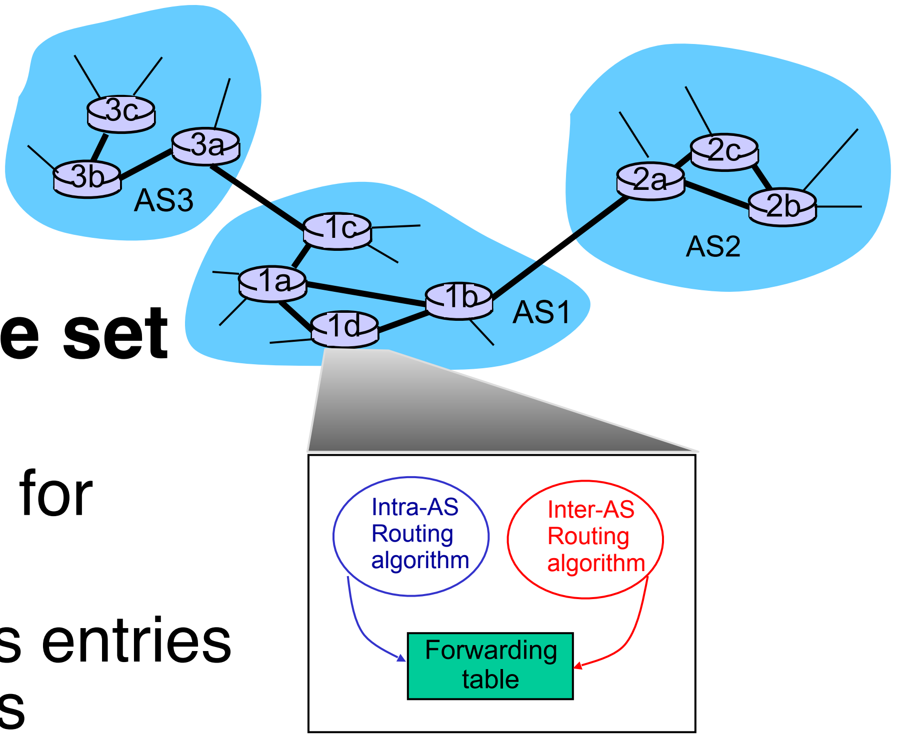
  - ➔ use graph algorithms (shortest path)

- **If Network = Internet ➔ huge graph**
  - And sub graphs (sub nets) controlled by different entities

- **How do we solve this problem?**

Hierarchical Routing

- **Solve routing problem in two levels:**
- **Intra AS (Autonomous System)**
  - **Use any algorithm, based on admin**
  - **graph algorithms**
- **Inter-ASes**
  - **Use global, standard, routing (BGP)**
- **Forwarding tables are set by both:**
  - intra-AS → sets entries for internal destinations
  - inter-AS & intra-AS sets entries for external destinations



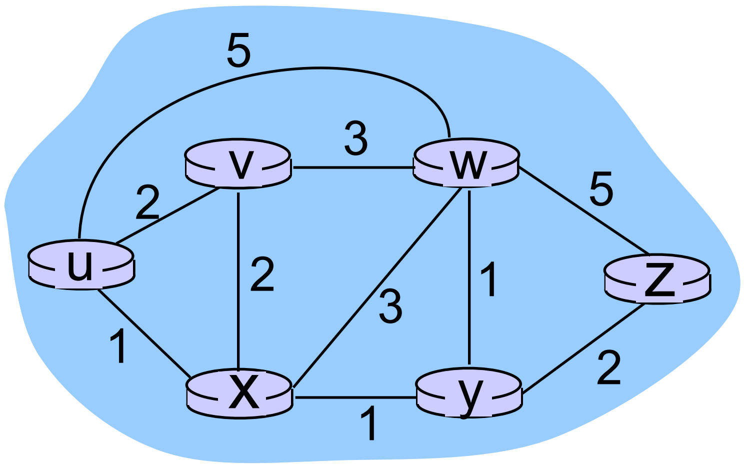
Intra-AS Routing

- also known as *interior gateway protocols (IGP)*
- most common intra-AS routing protocols:
  - **RIP:** Routing Information Protocol
    - Distance vector, Bellman-Ford algorithm, distributed
    - Old and small networks
  - **OSPF:** Open Shortest Path First
    - Link state, Dijkstra's algorithm, centralized
    - Most current networks
  - **IGRP:** Interior Gateway Routing Protocol
    - Cisco proprietary

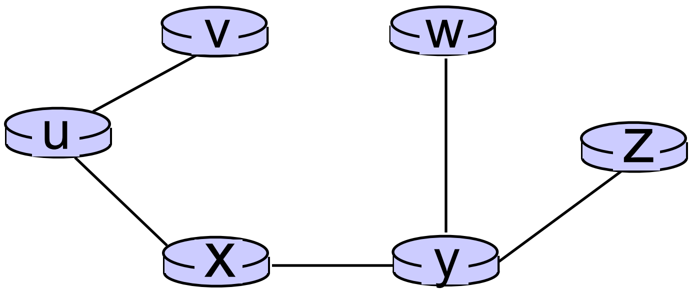
Output of Intra-domain Routing

resulting forwarding table in u:

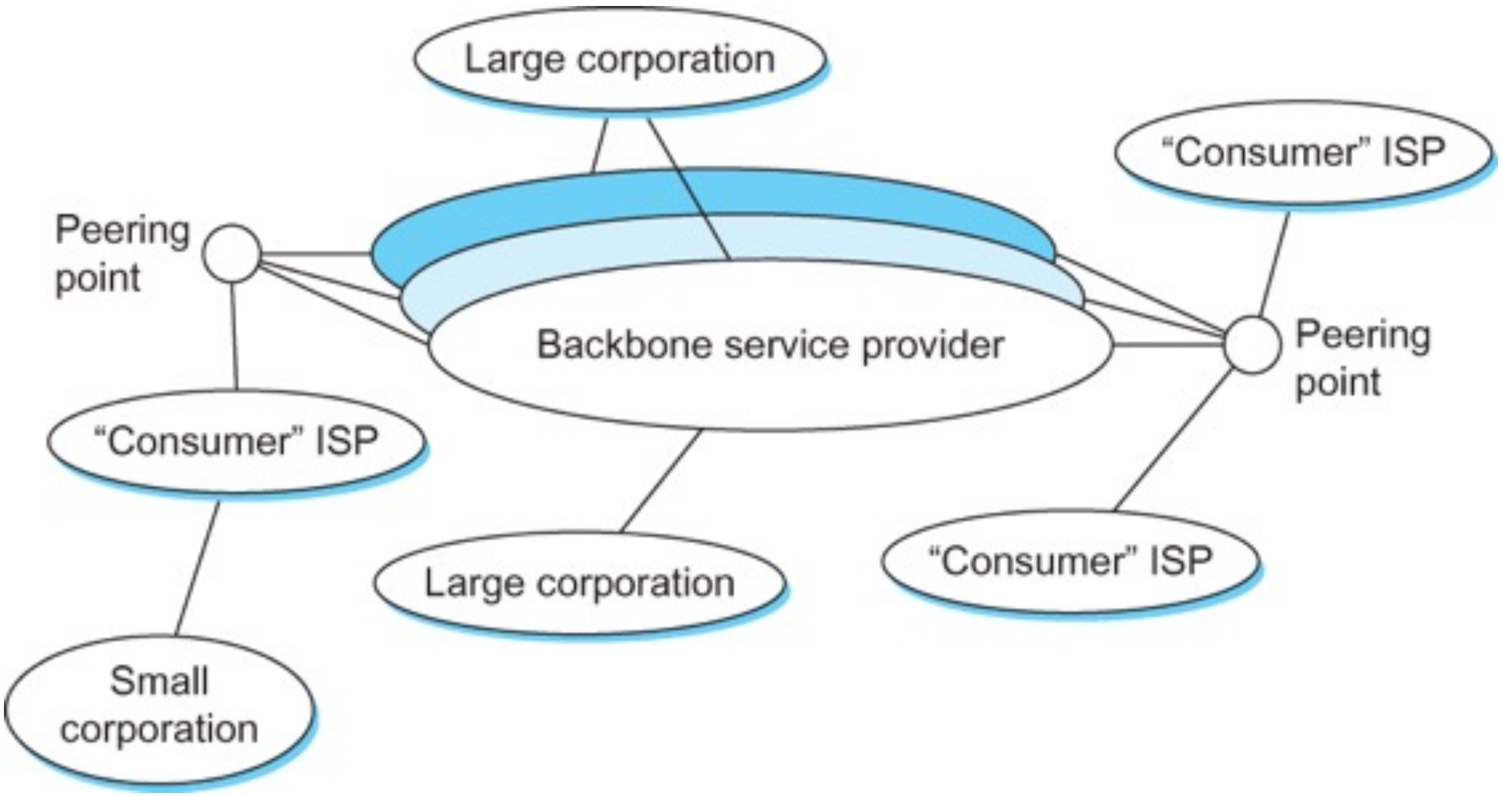
destination	link
v	(u,v)
x	(u,x)
y	(u,x)
w	(u,x)
z	(u,x)



resulting shortest-path tree from u:

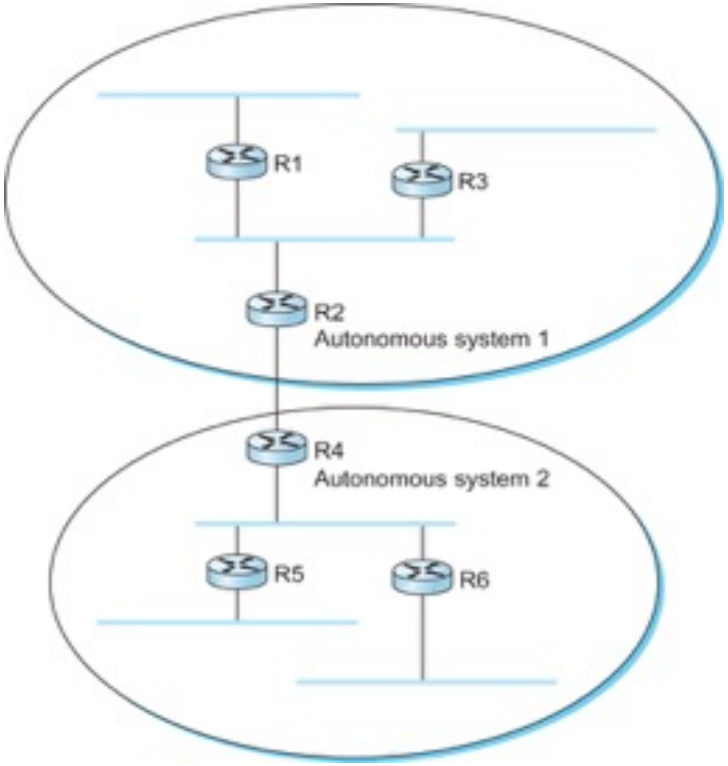


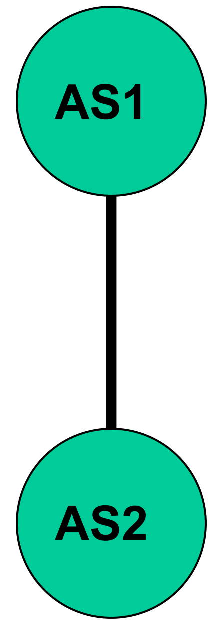
Inter-domain Routing: ASes



- Simple View of the Internet

Network of Two ASes



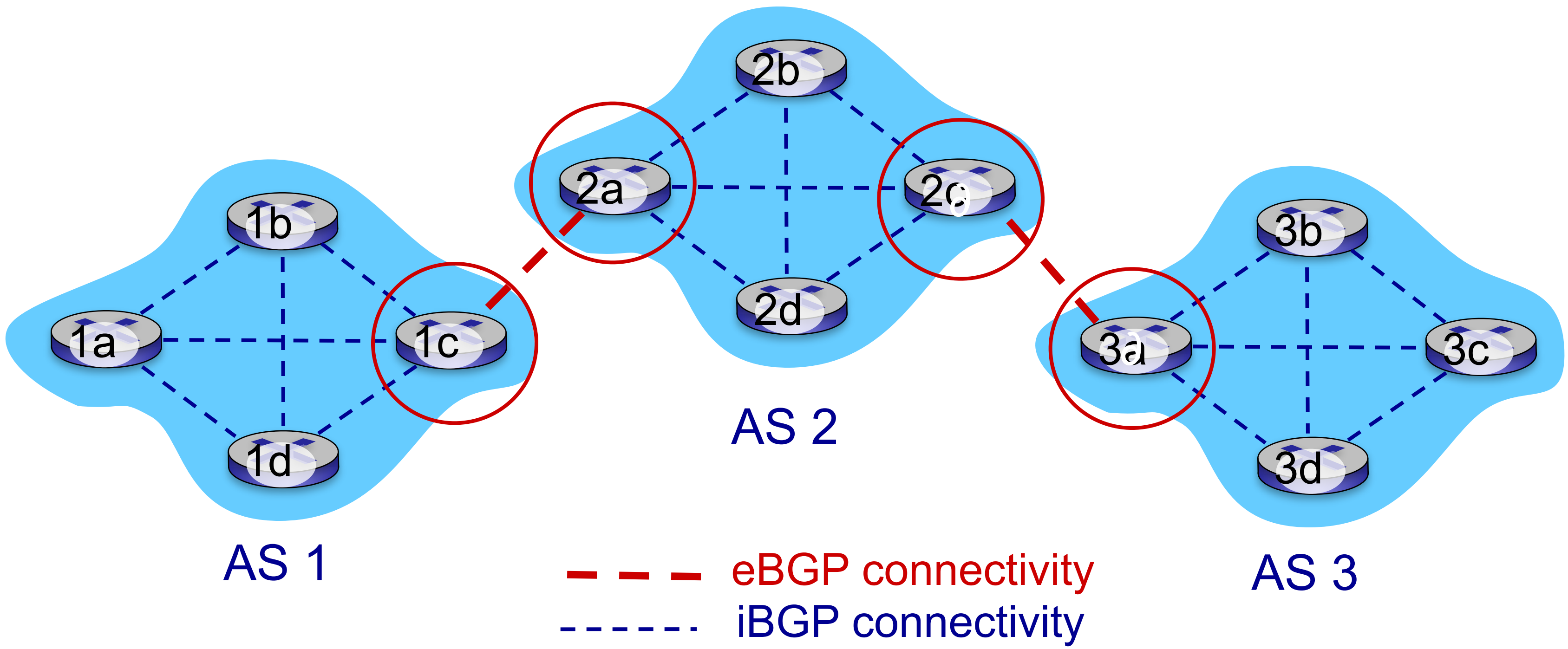


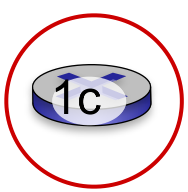
AS Graph

Internet inter-AS routing: BGP

- BGP (Border Gateway Protocol): *the* de facto inter-domain routing protocol
  - "glue that holds the Internet together"
- BGP provides each AS a means to:
  - eBGP: obtain subnet reachability information from neighboring ASes
  - iBGP: propagate reachability information to all AS-internal routers.
  - determine "*good*" routes to other networks based on reachability information and *policy*
- allows subnet to advertise its existence to rest of Internet: *"I am here"*

eBGP, iBGP connections



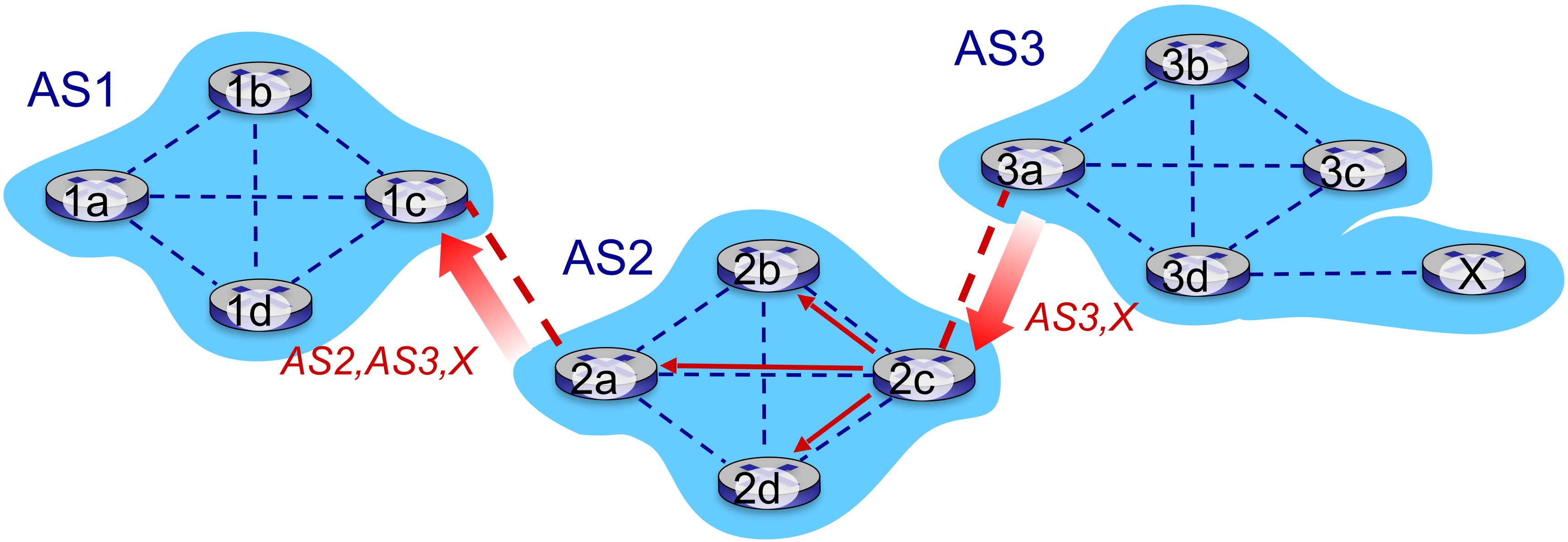


gateway routers run both eBGP and iBGP protocols

Path attributes and BGP routes

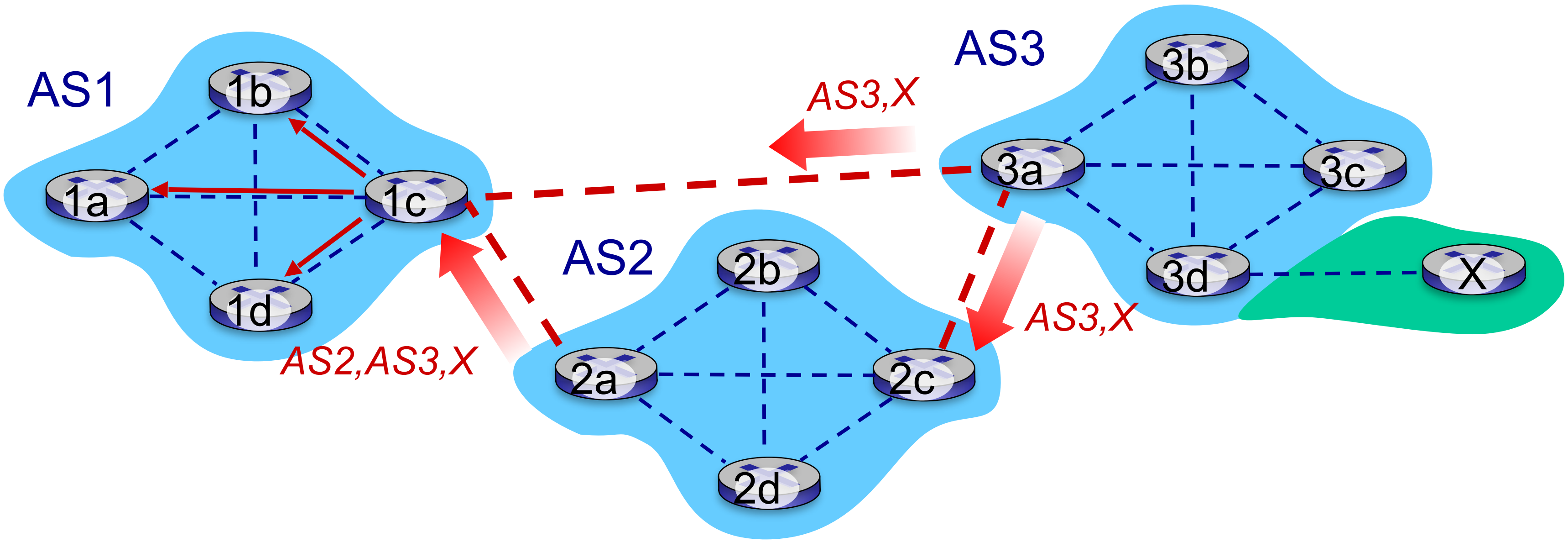
- advertised prefix includes BGP attributes
  - prefix + attributes = “route”
- two important attributes:
  - AS-PATH: list of ASes through which prefix advertisement has passed
  - NEXT-HOP: indicates specific internal-AS router to next-hop AS
- *Policy-based routing:*
  - gateway receiving route advertisement uses *import policy* to accept/decline path (e.g., never route through AS Y).
  - AS policy also determines whether to *advertise* path to other other neighboring ASes

BGP path advertisement



- AS2 router 2c receives path advertisement AS3,X (via eBGP) from AS3 router 3a
- Based on AS2 policy, AS2 router 2c accepts path AS3,X, propagates (via iBGP) to all AS2 routers
- Based on AS2 policy, AS2 router 2a advertises (via eBGP) path AS2, AS3, X to AS1 router 1c

BGP path advertisement

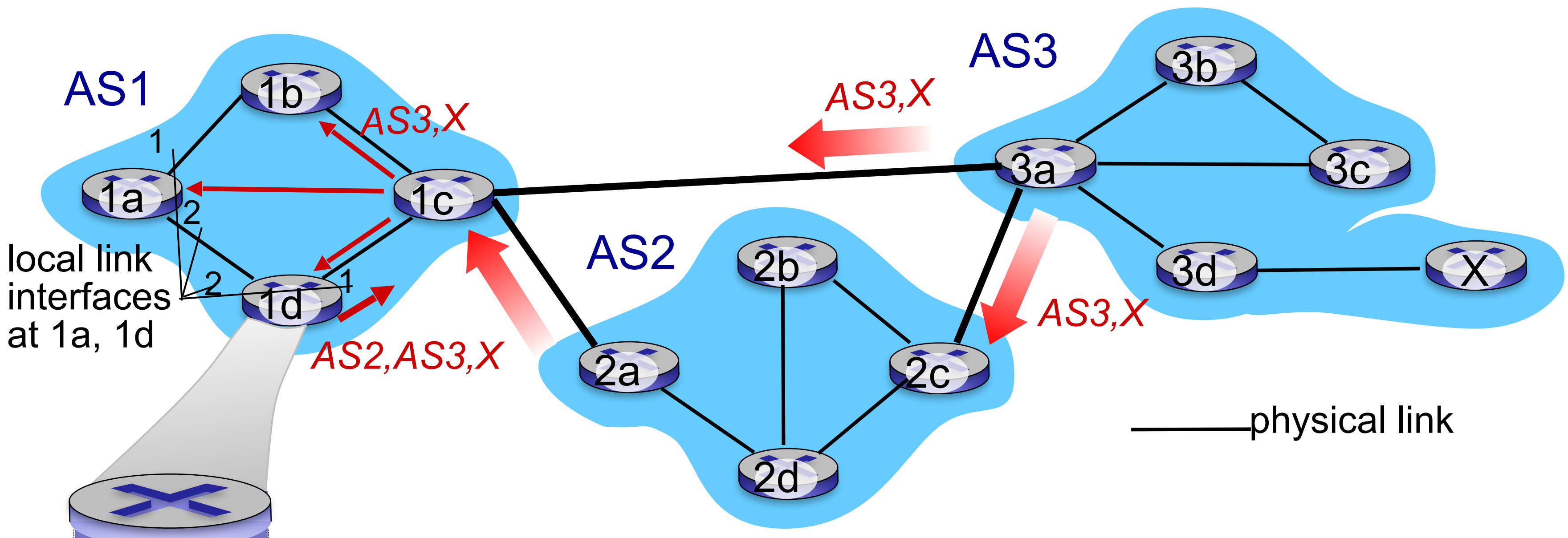


gateway router may learn about multiple paths to destination:

- AS1 gateway router 1c learns path *AS2,AS3,X* from 2a
- AS1 gateway router 1c learns path *AS3,X* from 3a
- Based on policy, AS1 gateway router 1c chooses path *AS3,X, and advertises path within AS1 via iBGP*

Setting forwarding table entries

Q: how does router set forwarding table entry to distant prefix?

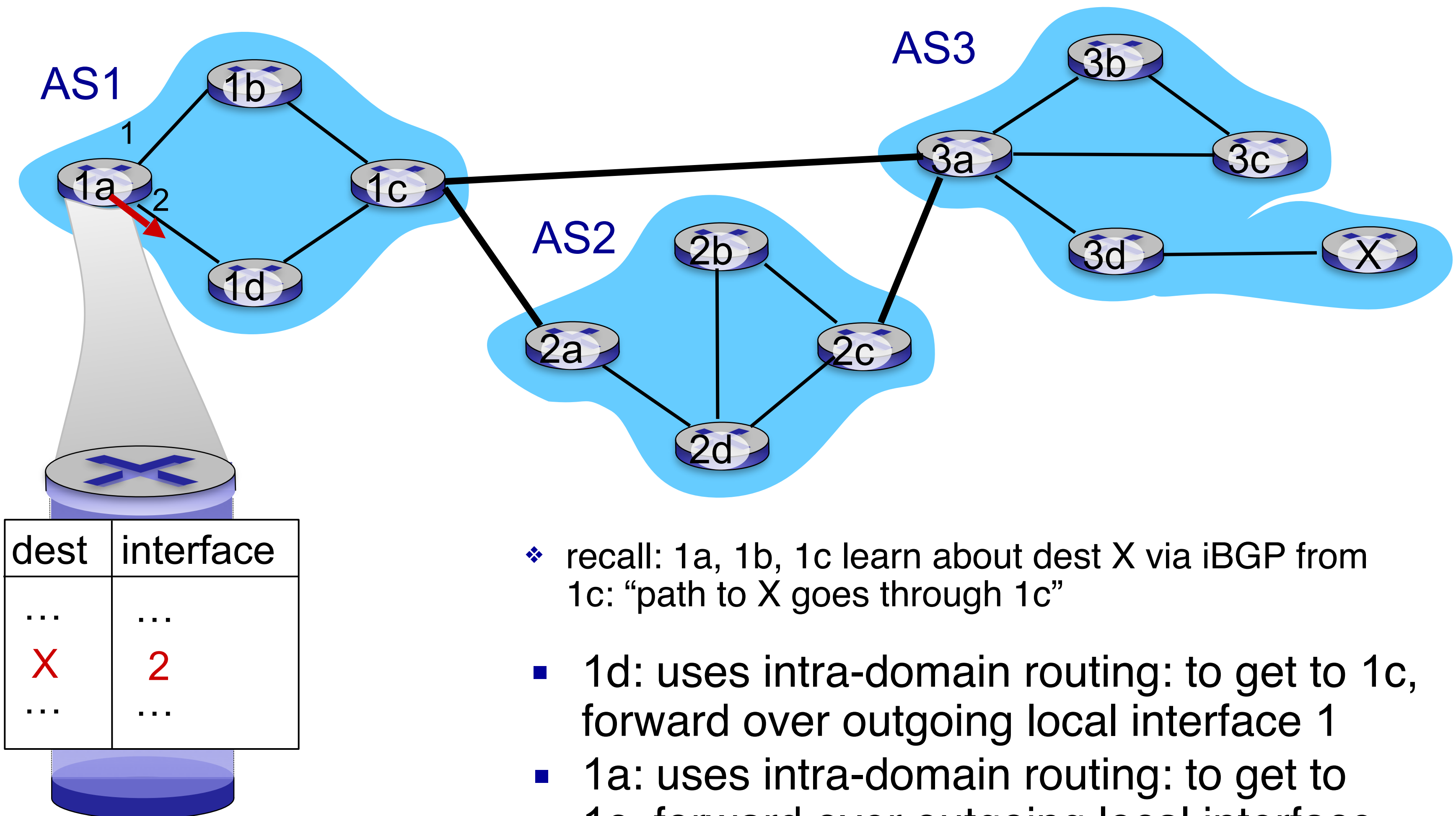


dest	interface
…	…
X	1
…	…

- recall: 1a, 1b, 1c learn about dest X via iBGP from 1c: “path to X goes through 1c”
- 1d: uses intra-domain routing: to get to 1c, forward over outgoing local interface 1

Setting forwarding table entries

Q: how does router set forwarding table entry to distant prefix?



dest	interface
…	…
X	2
…	…

- recall: 1a, 1b, 1c learn about dest X via iBGP from 1c: "path to X goes through 1c"

- 1d: uses intra-domain routing: to get to 1c, forward over outgoing local interface 1
- 1a: uses intra-domain routing: to get to 1c, forward over outgoing local interface 2

Why different Intra-, Inter-AS routing ?

policy:

- inter-AS: admin wants control over how its traffic routed, who routes through its net.
- intra-AS: single admin, so no policy decisions needed

scale:

- hierarchical routing saves table size, reduced update traffic

performance:

- intra-AS: can focus on performance
- inter-AS: policy may dominate over performance

Internet Multicast

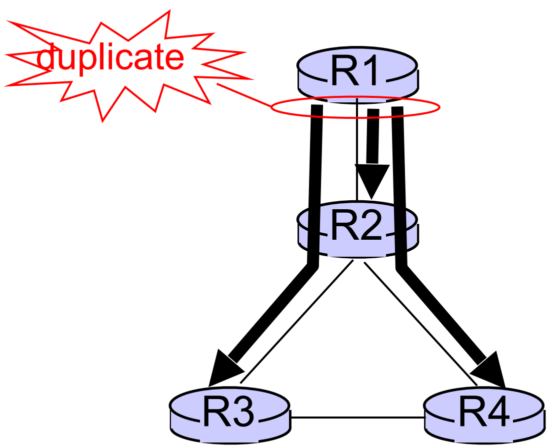
(aka IP Multicast,
Network-layer Multicast)

Multicast

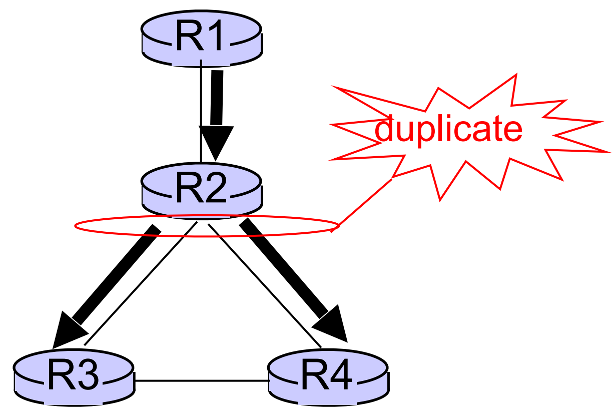
- **One-to-many**
  - One sender, many receivers, e.g.,
    - Transmitting sports games to many users
    - Transmitting news, stock-price
    - Software updates to multiple hosts
- **Many-to-many**
  - Multiple senders, multiple receivers, e.g.,
    - Multimedia teleconferencing
    - Online multi-player games

Multicast

- source duplication
  - Inefficient, many duplicates
  - Heavy load on links near source
- In-network duplication
  - Efficient
  - But routers need to support it



source duplication



in-network duplication

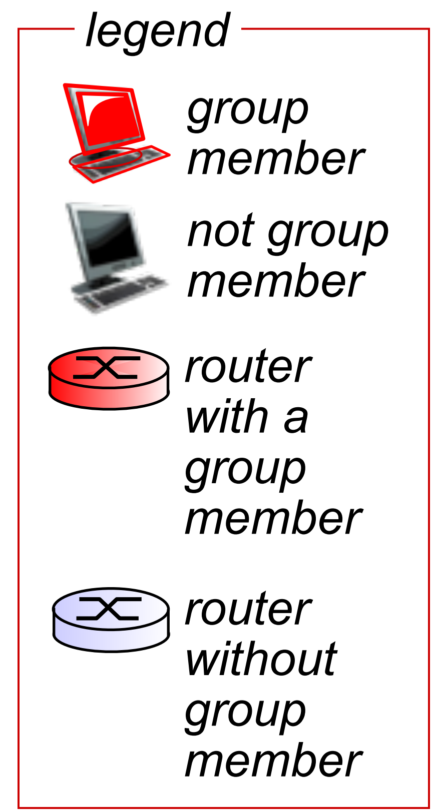
Overview

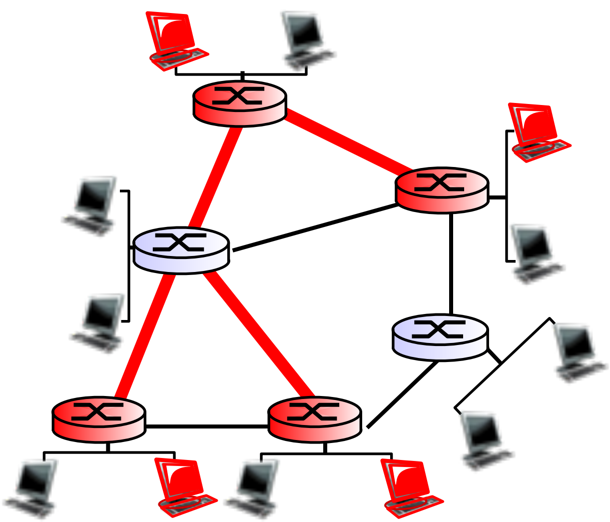
- **Create multicast group**
  - Specific IP Multicast address for the group
- **A host signals its desire to join/leave a group to its local router using:**
  - Internet Group Management Protocol (IGMP) for IPv4
  - Multicast Listener Discovery (MLD), for IPv6
- **Routers maintain multicast forwarding tables**
  - (in addition to unicast forwarding tables)
  - Specify which links to send packets on to reach hosts in the multicast group
  - ➔ form "**Multicast Distribution Trees**"

Multicast routing: problem statement

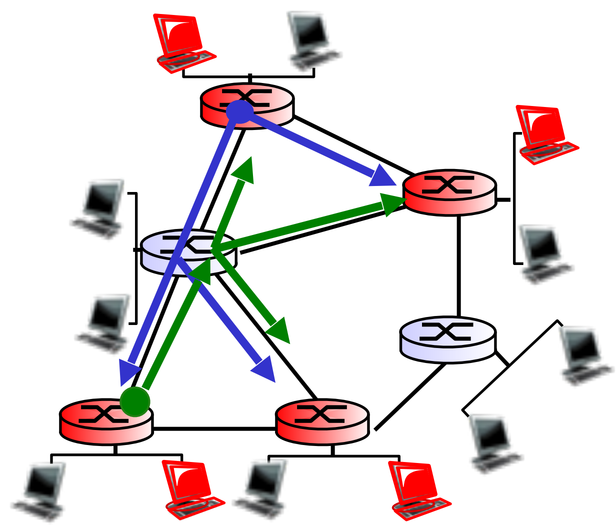
goal: find a tree (or trees) connecting routers having local mcast group members

- ***Two approaches:***
- *shared-tree:* same tree used by all group members
- *source-based:* different tree from each sender to rcvrs





shared tree



source-based trees

Approaches for building mcast trees

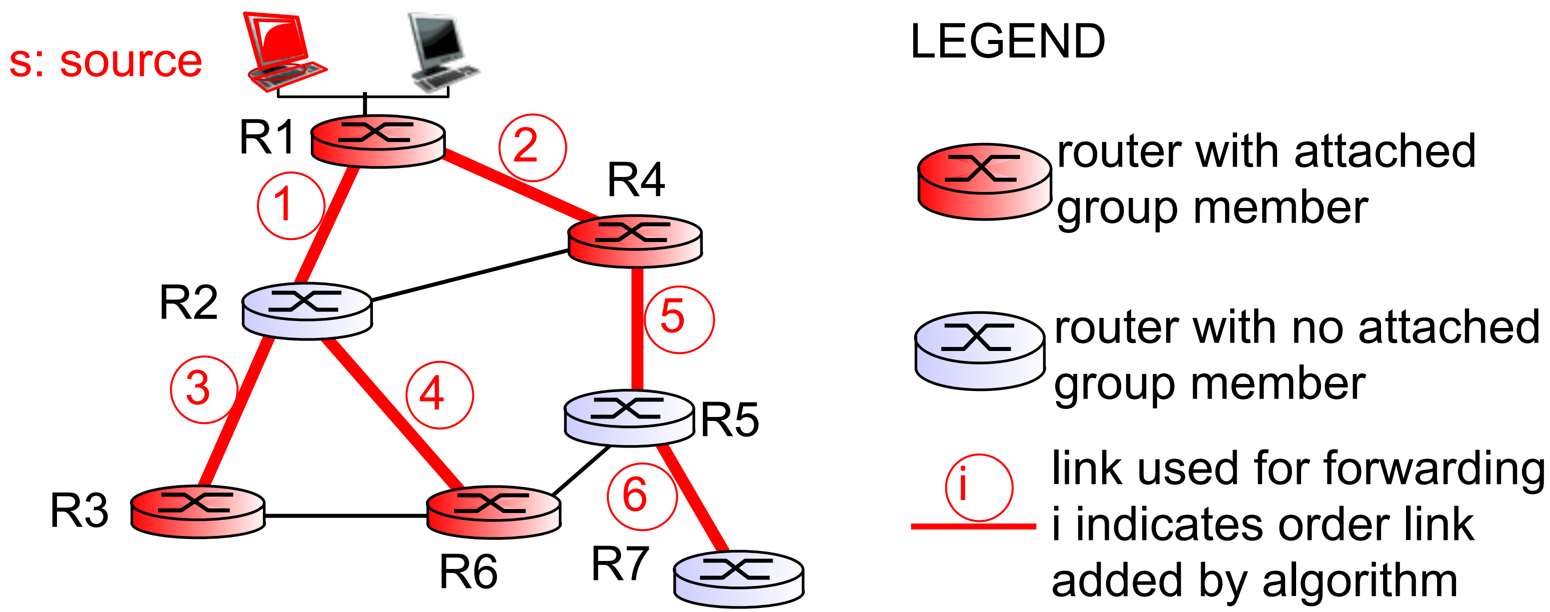
approaches:

- *source-based tree:* one tree per source
  - shortest path trees
  - reverse path forwarding
- *group-shared tree:* group uses one tree
  - minimal spanning (Steiner)
  - center-based trees

…we first look at basic approaches, then specific protocols adopting these approaches

Shortest path tree

- mcast forwarding tree: tree of shortest path routes from source to all receivers
  - Dijkstra's algorithm

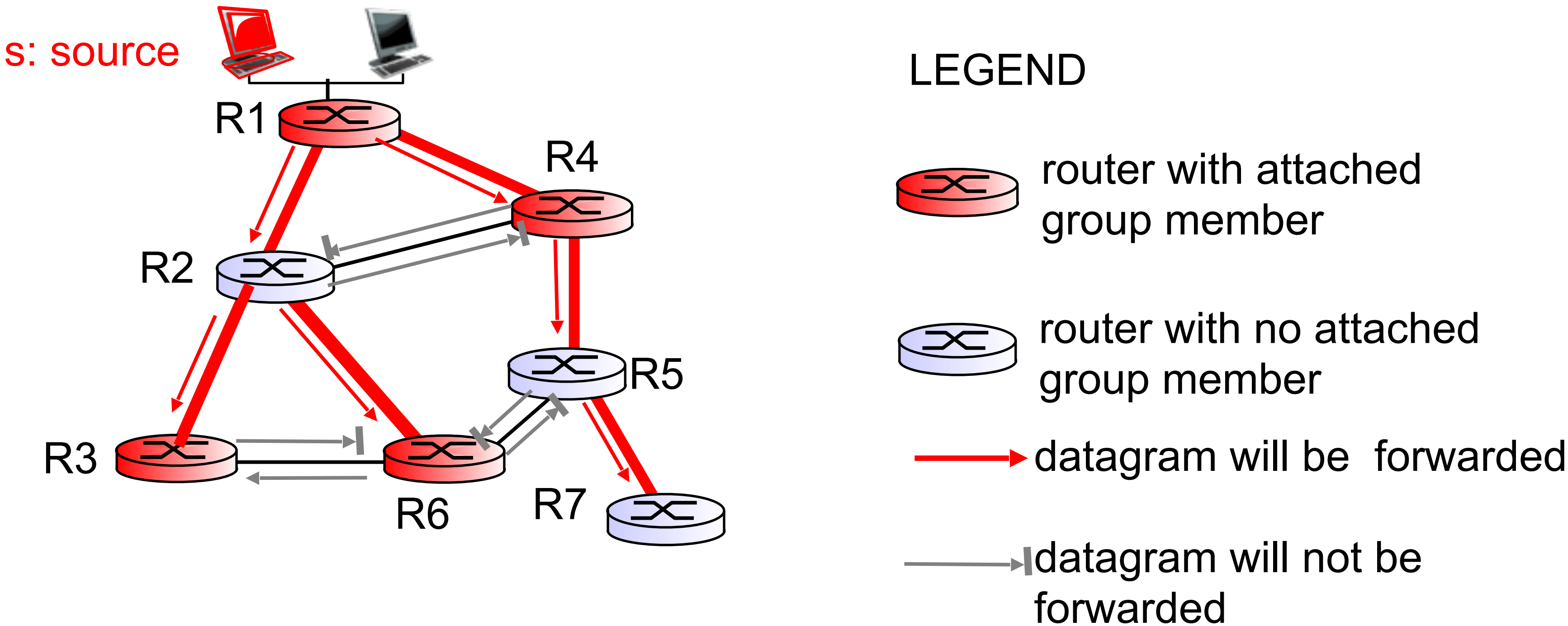


Reverse path forwarding

- rely on router's knowledge of unicast shortest path from it to sender
- each router has simple forwarding behavior:

if (mcast datagram received on incoming link on shortest path back to source)
then flood datagram onto all outgoing links
else ignore datagram

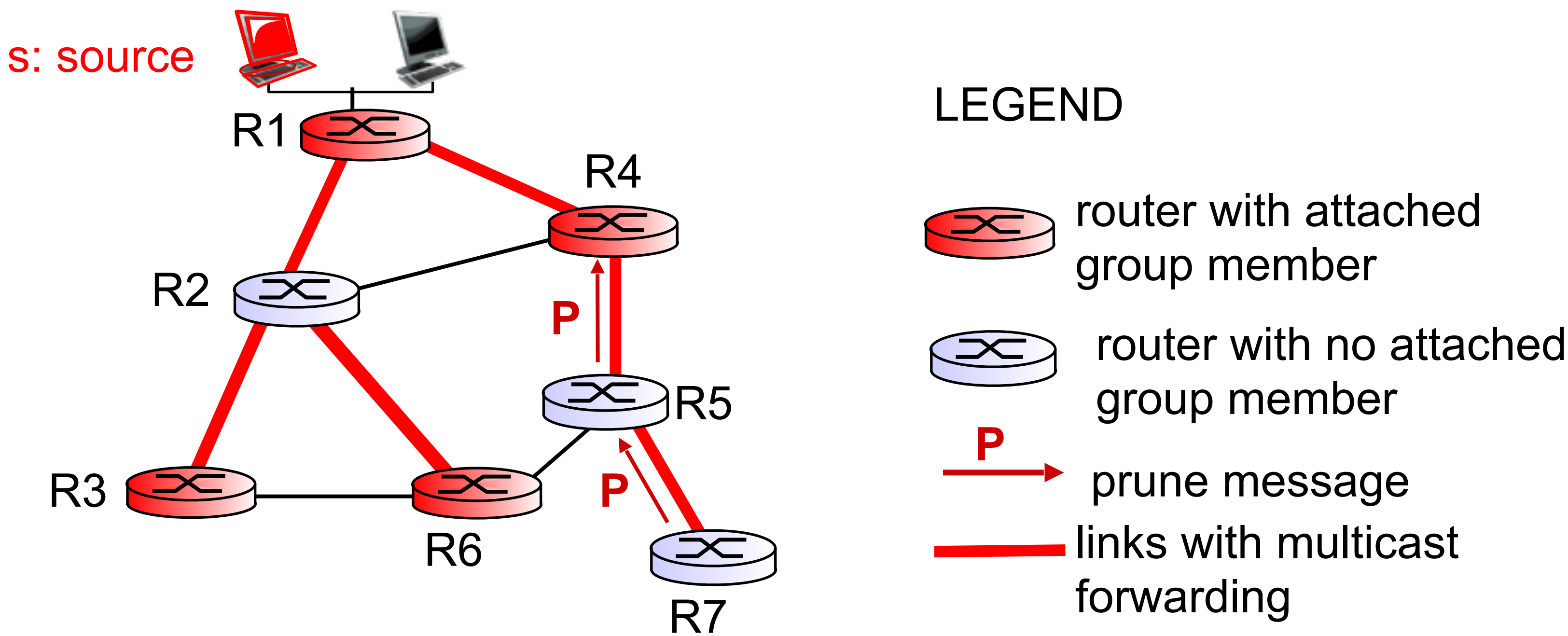
Reverse path forwarding: example



- result is a source-specific *reverse* SPT
  - Note: assumes symmetric links: shortest path R1→R2 same as R2→R1; not always true

Reverse path forwarding: pruning

- forwarding tree contains subtrees with no mcast group members
  - no need to forward datagrams down subtree
  - "prune" msgs sent upstream by router with no downstream group members



Shared-tree: Steiner tree

- *Steiner tree:* minimum cost tree connecting all routers with attached group members
- problem is NP-complete
- not used in practice:
  - computational complexity
  - information about entire network needed
  - monolithic: rerun whenever a router needs to join/ leave

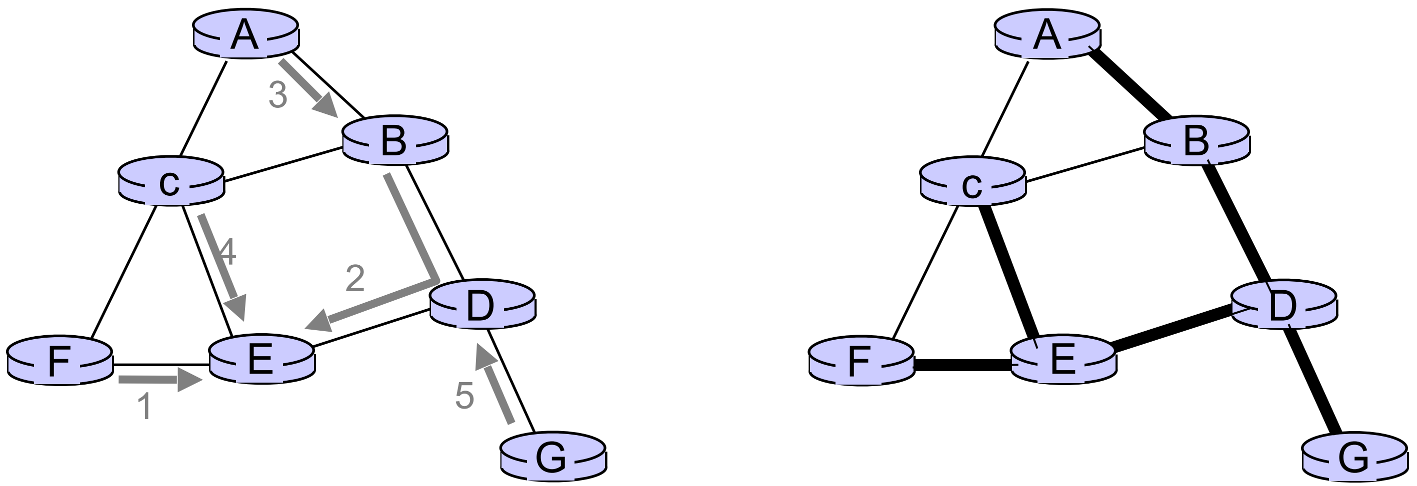
- good heuristics exist

Center-based tree (Heuristic)

- single delivery tree shared by all
- one router identified as *"center"* of tree
- to join:
  - edge router sends unicast *join-msg* addressed to center router
  - *join-msg* "processed" by intermediate routers and forwarded towards center
  - *join-msg* either hits existing tree branch for this center, or arrives at center
  - path taken by *join-msg* becomes new branch of tree for this router

Center-based tree: example

- Chose a center node
- each node sends unicast join message to center node
  - message forwarded until it arrives at a node already belonging to spanning tree



(a) stepwise construction of spanning tree (center: E)

(b) constructed spanning tree

Internet Multicasting Routing: DVMRP

- DVMRP: distance vector multicast routing protocol, RFC1075
- *flood and prune:* reverse path forwarding, source-based tree
  - RPF tree based on DVMRP's own routing tables constructed by communicating DVMRP routers
    - Relies on the distance vector (DV) unicast routing protocol
  - initial datagram to mcast group flooded everywhere via RPF
  - routers not wanting group: send upstream prune msgs

DVMRP: continued…

- *soft state:* DVMRP router periodically (1 min.) "forgets" branches:
  - mcast data again flows down unpruned branch
  - downstream router: reprune or else continue to receive data
- routers can quickly regraft to tree
  - following IGMP join at leaf
- DVMRP: commonly implemented in commercial router

PIM: Protocol Independent Multicast

- not dependent on any specific underlying unicast routing algorithm (works with all)
- two different multicast distribution scenarios :

dense:

- group members densely packed, in "close" proximity.
- bandwidth more plentiful

sparse:

- # networks with group members small wrt # interconnected networks
- group members "widely dispersed"
- bandwidth not plentiful

Consequences of sparse-dense dichotomy:

dense

- group membership by routers *assumed* until routers explicitly prune
- *data-driven* construction on mcast tree
- bandwidth and non-group-router processing *wasted*
- *Uses flood and prune RPF*

sparse:

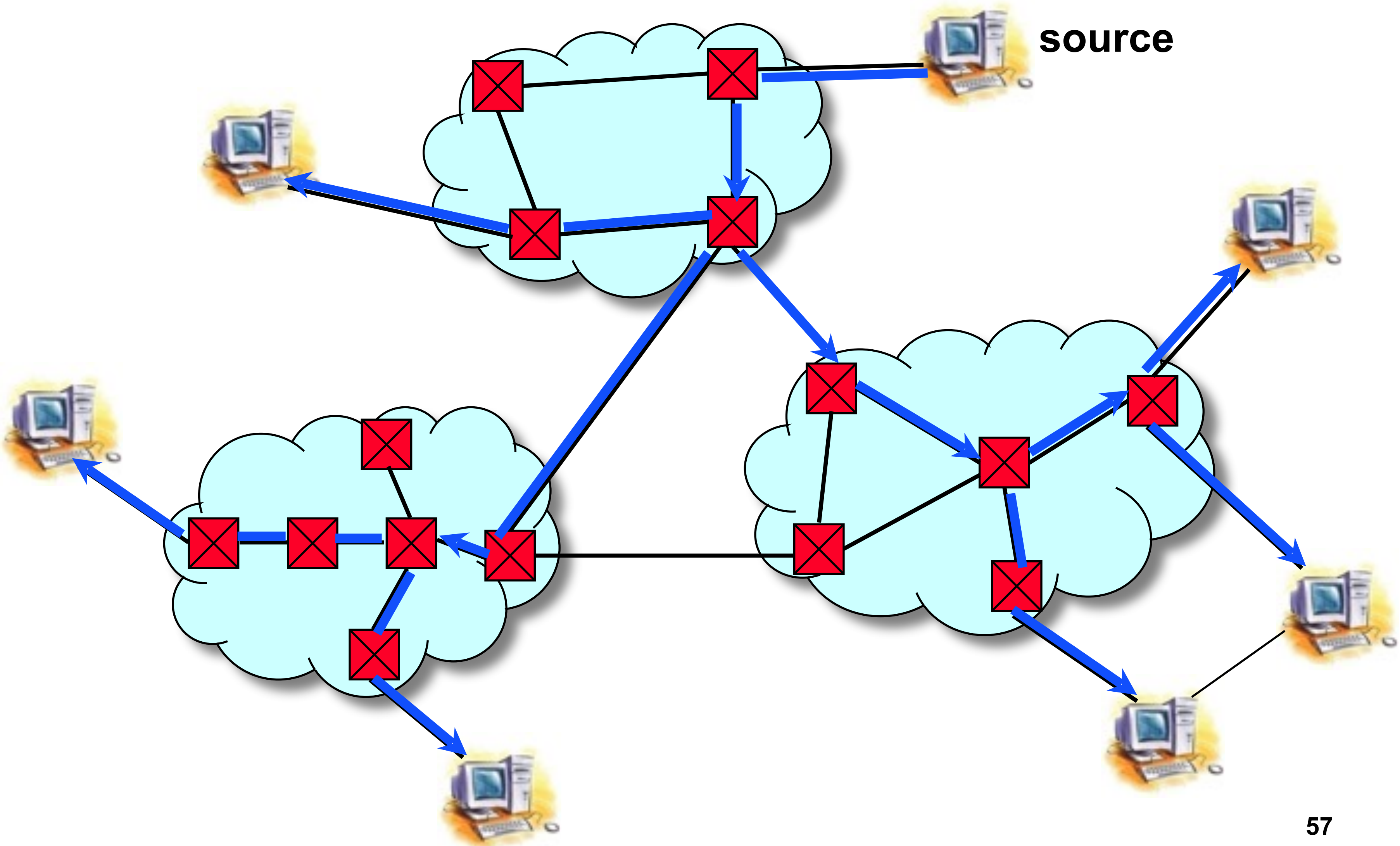
- no membership until routers explicitly join
- *receiver- driven* construction of mcast tree
- bandwidth and non-group-router processing *conservative*
- Uses center-based tree

Inter-Domain Multicast

- Domains (or Automatous Systems) are independent
  - ➔ a domain should not rely on center point of another
  - ➔ Each domain has its own center (rendezvous) point
- Then center points of different domains communicate with each other to create mcast group across domains

Overlays and Application Layer Multicast (ALM)

IP Multicast



What is wrong with IP Multicast?

- IP Multicast
  - Most efficient (packets traverse each link only once)
- What is wrong with IP Multicast?
  - Not enabled in many routers
  - Not scalable (core routers need to maintain state for each multicast sessions)
- → build mcast trees in the Application Layer
  - *Without* router involvement
  - → fast deployment (because we do not change anything in the core of the network)

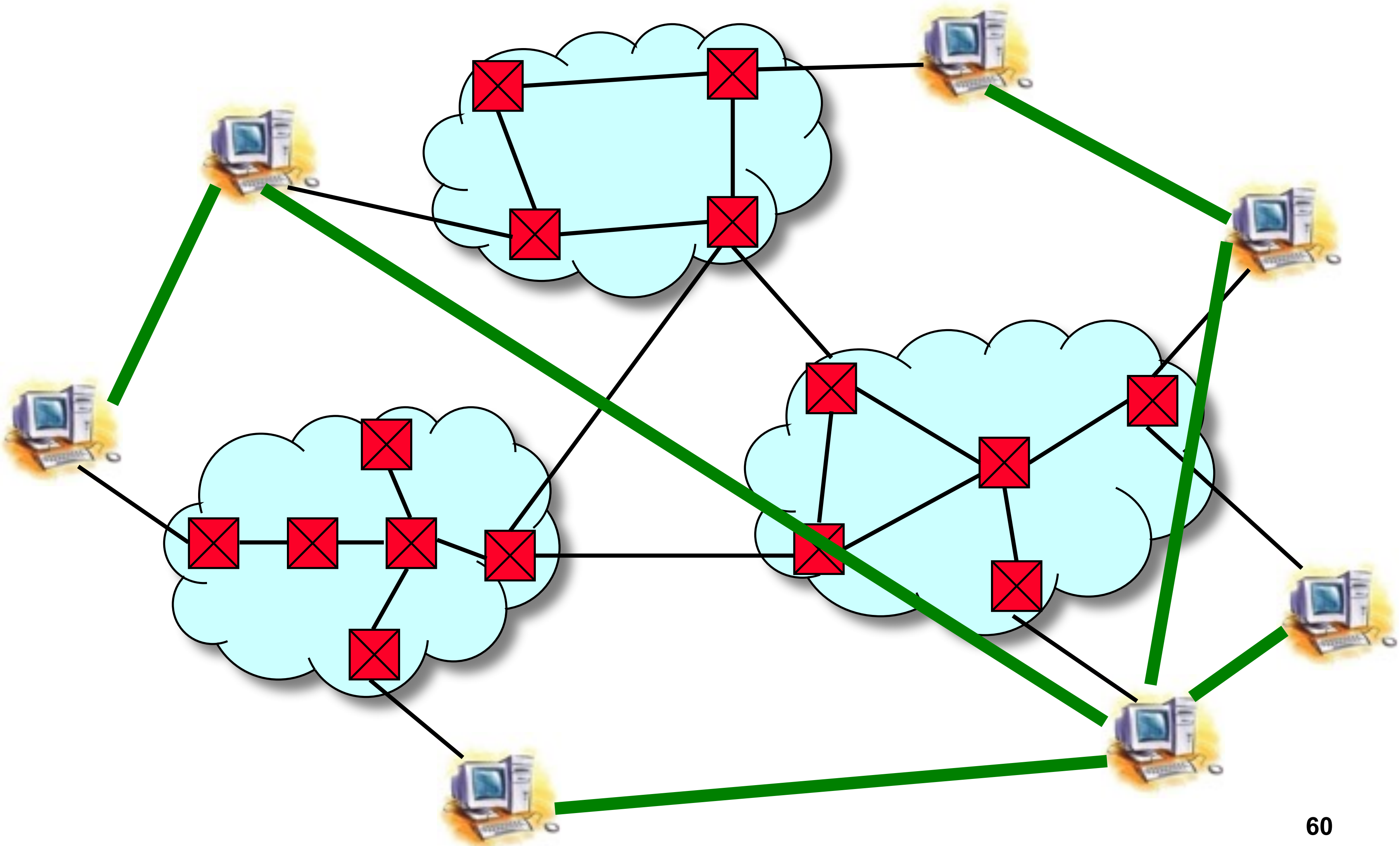
Overlay Network

- We create abstract layer on top of physical network called "overlay network"

- Nodes are end hosts (not routers) running certain software
  - We build mcast trees on those nodes

- Neighbors in overlay can be several hops away in physical network

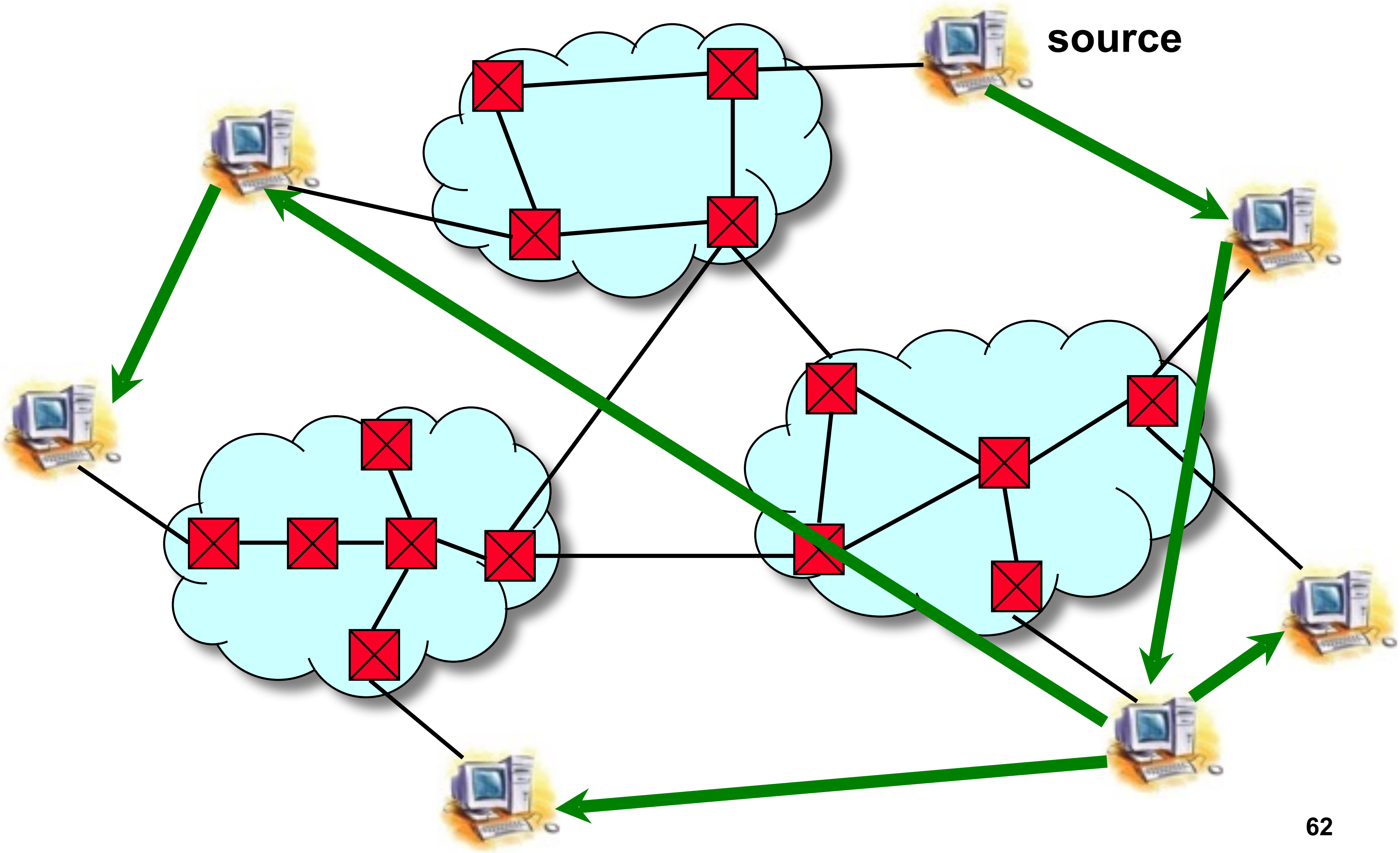
Overlay Network (cont'd)



Overlay Network (cont'd)

- Why do we need overlays?
- Flexibility in
  - Choosing neighbors
  - Forming and customizing topology to fit application's needs (e.g., short delay, reliability, high BW, …)
  - Designing communication protocols among nodes
- Get around limitations in *legacy* networks
- Enable new (and old!) network services

Application Level Multicast (ALM)



Overlay Network

- **Overlay design issues**
  - Select neighbors
  - Handle node arrivals, departures
  - Detect and handle failures (nodes, links)
  - Monitor and adapt to network dynamics
  - Match with the underlying physical network
    - Several optimization algorithms, e.g., see papers on ESM, NICE, Zigzag

Overlay Network (cont'd)

- Some applications that use overlays
  - Application level multicast, e.g., ESM, Zigzag, NICE, …
    - Build multicast tree(s) or mesh(es) in the application (not network) layer
  - Reliable inter-domain routing, e.g., RON
    - Improves BGP by finding robust routes faster
  - Content Distribution Networks (CDN)
    - To distribute bandwidth intensive content (software updates,…)
  - Peer-to-peer (P2P) file sharing
    - File exchange among peers
  - P2P streaming
    - Real time streaming

Overlay Networks and P2P Systems

Mohamed Hefeeda

P2P Computing: Definitions

- Peers cooperate to achieve desired functions
  - Peers:
    - End-systems (typically, user machines)
    - Interconnected through an *overlay* network
    - Peer ≡ Like the others (similar or behave in similar manner)
  - Cooperate:
    - Share resources, e.g., data, CPU cycles, storage, bandwidth
    - Participate in protocols, e.g., routing, replication, …
  - Functions:
    - File-sharing, distributed computing, communications, content distribution, streaming, …
- Note: the P2P concept is much wider than file sharing

When Did P2P Start?

- Napster (Late 1990’s)
  - Court shut Napster down in 2001
- Gnutella (2000)
- Then FastTrack (Kazaa, ...),
- Then BitTorrent, and many others
- Accompanied by significant research interest

Characteristics of P2P Systems

- **Characteristics of**
  - **Nodes** (peers) which constitute the
  - **System** that we build

Nodes in P2P Systems

- Typically:
  - Quite heterogeneous
    - Several order of magnitudes difference in resources
    - Compare bandwidth of dial-up peer vs high-speed LAN peer
  - Unreliable
    - Failure is the norm!
  - Offer limited capacity
    - Load sharing and balancing are critical
  - Autonomous
    - Rational, i.e., maximize their own benefits!
    - Motivations should be provided to peers to cooperate in a way that optimizes the system performance

P2P System Characteristics

- **System**
  - Scale
    - Numerous number of peers (millions)
  - Structure and topology
    - Ad-hoc: No control over peer joining/leaving
    - Highly dynamic
  - Membership/participation
    - Typically open ➔
  - More security concerns
    - Trust, privacy, data integrity, …
  - Cost of building and running
    - Small fraction of same-scale centralized systems
    - How much would it cost to build/run a super computer with processing power of that 3 Million SETI@Home PCs?

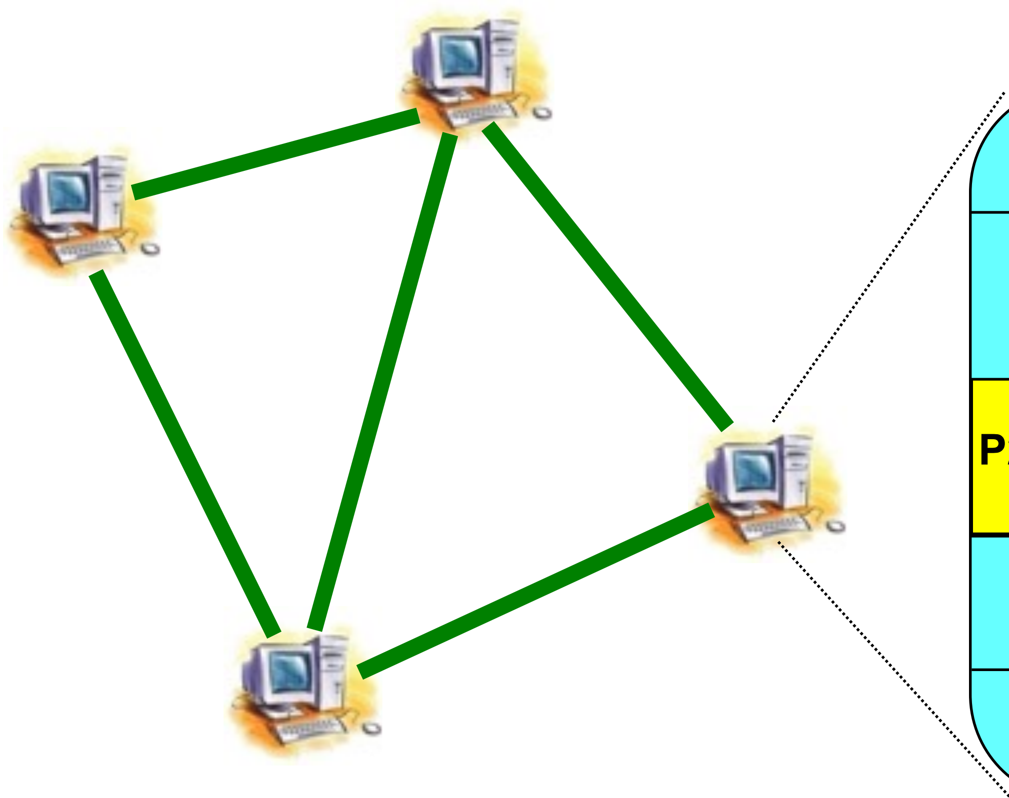
Requirements for P2P Systems

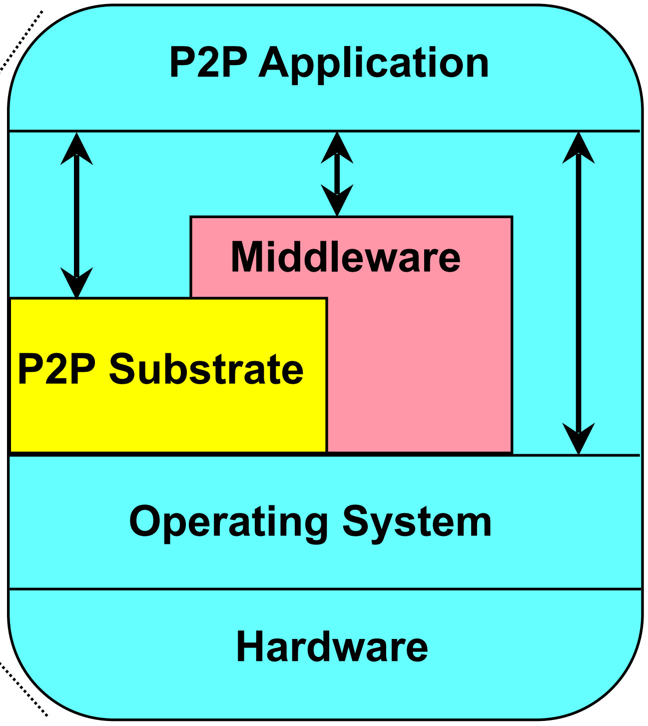
- We need to design lighter-weight algorithms and protocols to scale to millions of nodes given the new characteristics

Sample P2P Applications

- File sharing
  - BitTorrent, Overnet, eDonkey, Gnutella,, …
- Distributed cycle sharing
  - SETI@home, Gnome@home, …
- File and storage systems
  - OceanStore, CFS, Freenet, Farsite, …
- Media streaming and content distribution
  - SopCast, CoolStreaming, …
  - SplitStream, CoopNet, PeerCast, Bullet, Zigzag, NICE, …

P2P Systems: Simple Model





System architecture: Peers form an overlay according to the P2P Substrate

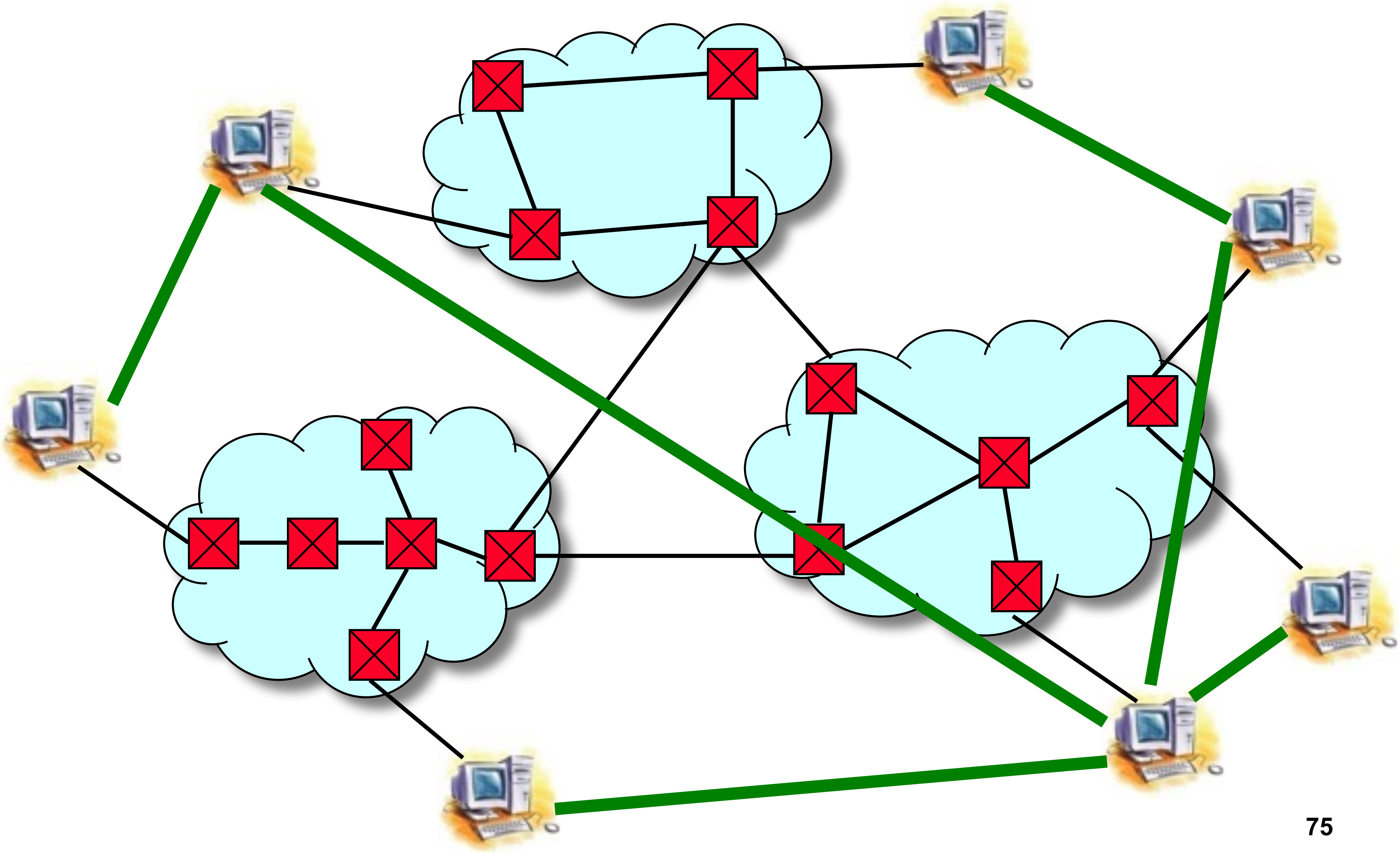
Software architecture model on a peer

Overlay Network

- An abstract layer built on top of physical network

- Neighbors in overlay can be several hops away in physical network

Overlay Network (cont'd)



Overlay Network (cont'd)

- **Why do we need overlays?**
- **Flexibility in**
  - **Choosing neighbors**
  - **Forming and customizing topology to fit application's needs (e.g., short delay, reliability, high BW, …)**
  - **Designing communication protocols among nodes**
- **Get around limitations in *legacy* networks**
- **Enable new (and old!) network services**

Overlay Network (cont'd)

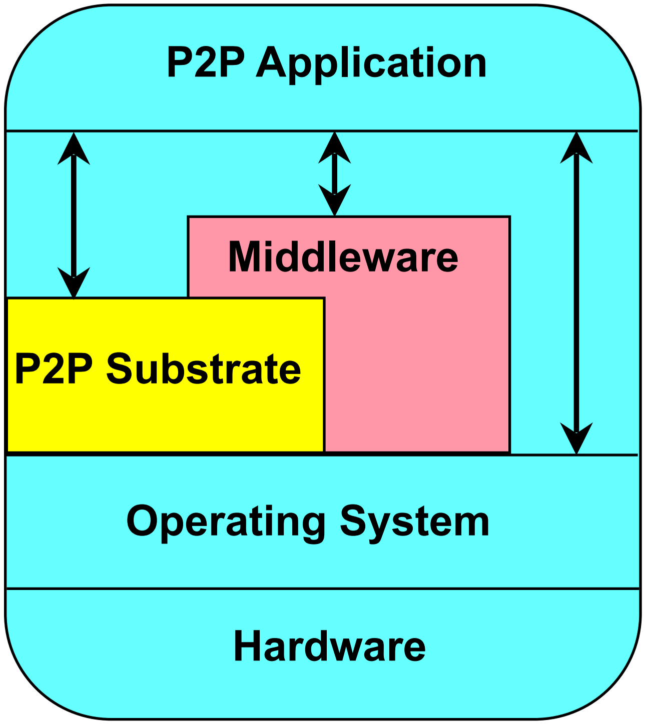
- Overlay design issues
  - Select neighbors
  - Handle node arrivals, departures
  - Detect and handle failures (nodes, links)
  - Monitor and adapt to network dynamics
  - Match with the underlying physical network

Overlay Network (cont'd)

- **Some applications that use overlays**
  - Application level multicast, e.g., ESM, Zigzag, NICE, …
    - Build multicast tree(s) or mesh(es) in the application (not network) layer
  - Reliable inter-domain routing, e.g., RON
    - Improves BGP by finding robust routes faster
  - Content Distribution Networks (CDN)
    - To distribute bandwidth intensive content (software updates,…)
  - Peer-to-peer file sharing
    - File exchange among peers
  - P2P streaming
    - Real time streaming

Peer Software Model

- A software client installed on each peer
- Three components:
  - P2P Substrate
  - Middleware
  - P2P Application



Software model on peer

Peer Software Model (cont'd)

- **P2P Substrate (key component)**
  - Overlay management
    - Construction
    - Maintenance (peer join/leave/fail and network dynamics)
  - Resource management
    - Allocation (storage)
    - Discovery (routing and lookup)
- **Ex: Pastry, CAN, Chord, …**
  - Described in research papers

Peer Software Model (cont'd)

- **Middleware**
  - Provides auxiliary services to P2P applications:
    - Peer selection
    - Trust management
    - Data integrity validation
    - Authentication and authorization
    - Membership management
    - Accounting (Economics and rationality)
    - …
  - Ex: CollectCast, EigenTrust, Micro payment
    - Described in research papers

Peer Software Model (cont'd)

- P2P Application
  - Potentially, there could be multiple applications running on top of a single P2P substrate
  - Applications include
    - File sharing
    - File and storage systems
    - Distributed cycle sharing
    - Content distribution
  - This layer provides some functions and bookkeeping relevant to target application
    - File assembly (file sharing)
    - Buffering and rate smoothing (streaming)
- Ex: Promise, Bullet, CFS

P2P Substrate

- Key component, which
  - Manages the Overlay
  - Allocates and discovers objects
- P2P Substrates can be
  - Structured
  - Unstructured
  - Based on the flexibility of placing objects at peers

P2P Substrates: Classification

- **Structured (or tightly controlled, aka DHT)**
  - **Objects are *rigidly* assigned to specific peers**
  - **Looks like as a Distributed Hash Table (DHT)**
  - **Efficient search & guarantee of finding**
  - **Lack of partial name and keyword queries**
  - **Maintenance overhead**
  - **Ex: Chord, CAN, Pastry, Tapestry, Kademila (Overnet)**

P2P Substrates: Classification

- Unstructured (or loosely controlled)
  - Objects can be anywhere
  - Support partial name and keyword queries
  - Inefficient search & no guarantee of finding
  - Some heuristics exist to enhance performance
  - Ex: Gnutella, Kazaa (super node), GIA [Chawathe et al. 03]

Structured P2P Substrates

- Objects are rigidly assigned to peers
  - Objects and peers have IDs (usually by hashing some attributes)
  - Objects are assigned to peers based on IDs
- Peers in overlay form specific *geometrical* shape, e.g.,
  - tree, ring, hypercube, butterfly network
- Shape (to some extent) determines
  - How neighbors are chosen, and
  - How messages are routed

Structured P2P Substrates (cont'd)

- Substrate provides a Distributed Hash Table (DHT)-like interface
  - InsertObject (key, value), findObject (key), …
  - In the literature, many authors refer to structured P2P substrates as DHTs
- It also provides peer management (join, leave, fail) operations
- Most of these operations are done in *O(log n)* steps, *n* is number of peers

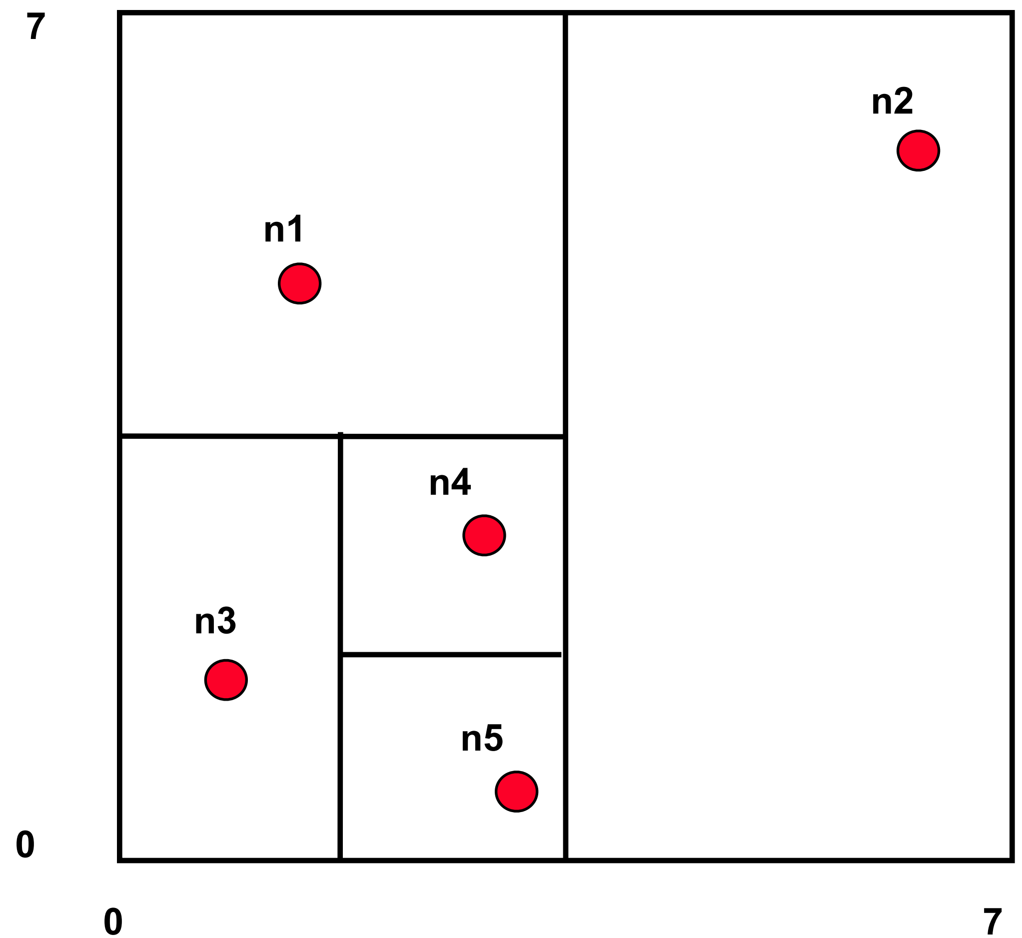
Structured P2P Substrates (cont'd)

- DHTs: Efficient search & guarantee of finding
- However,
  - Lack of partial name and keyword queries
  - Maintenance overhead, even *O(log n)* may be too much in very dynamic environments
- Ex: Chord, CAN, Pastry, Tapestry, Kademila (Overnet)

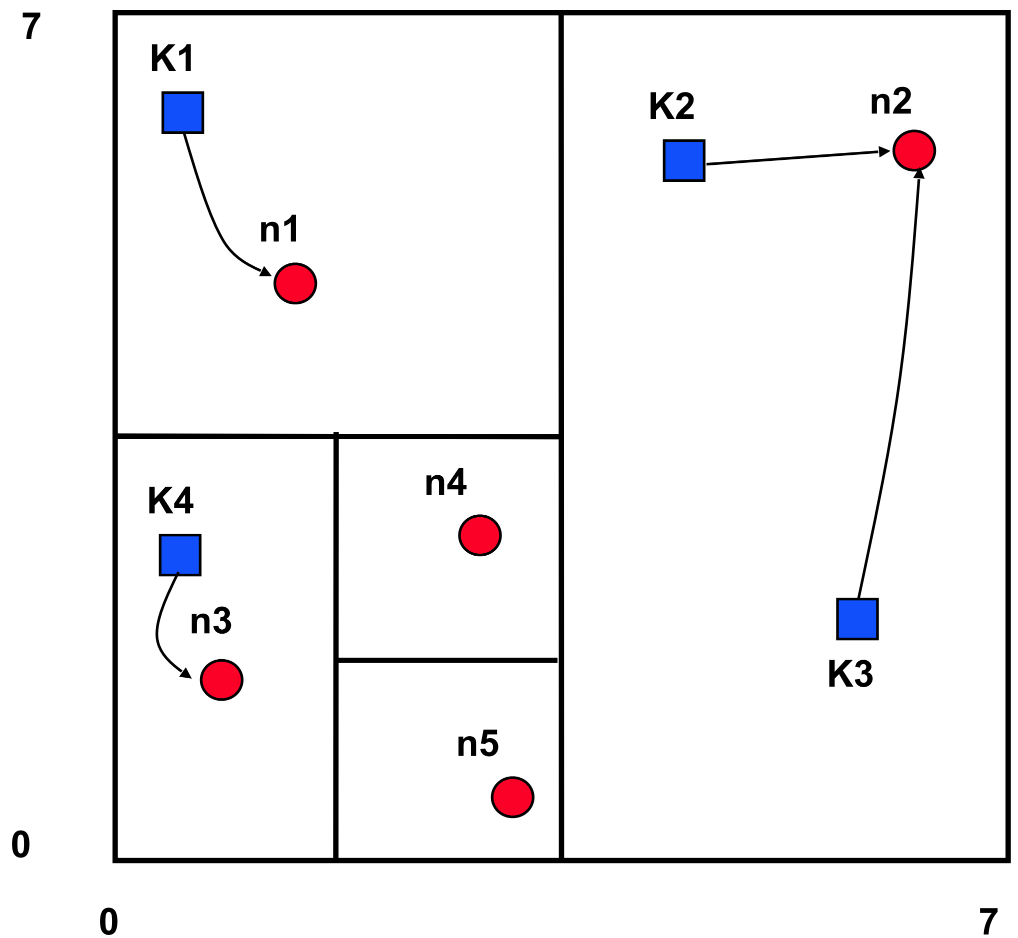
Example: Content Addressable Network (CAN) [Ratnasamy 01]

- Nodes form an overlay in *d*-dimensional space
  - Node IDs are chosen randomly from the d-space
  - Object IDs (keys) are chosen from the same d-space
- Space is dynamically partitioned into zones
- Each node owns a zone
- Zones are split and merged as nodes join and leave
- Each node stores
  - Portion of the hash table that belongs to its zone
  - Information about its immediate neighbors in the d-space

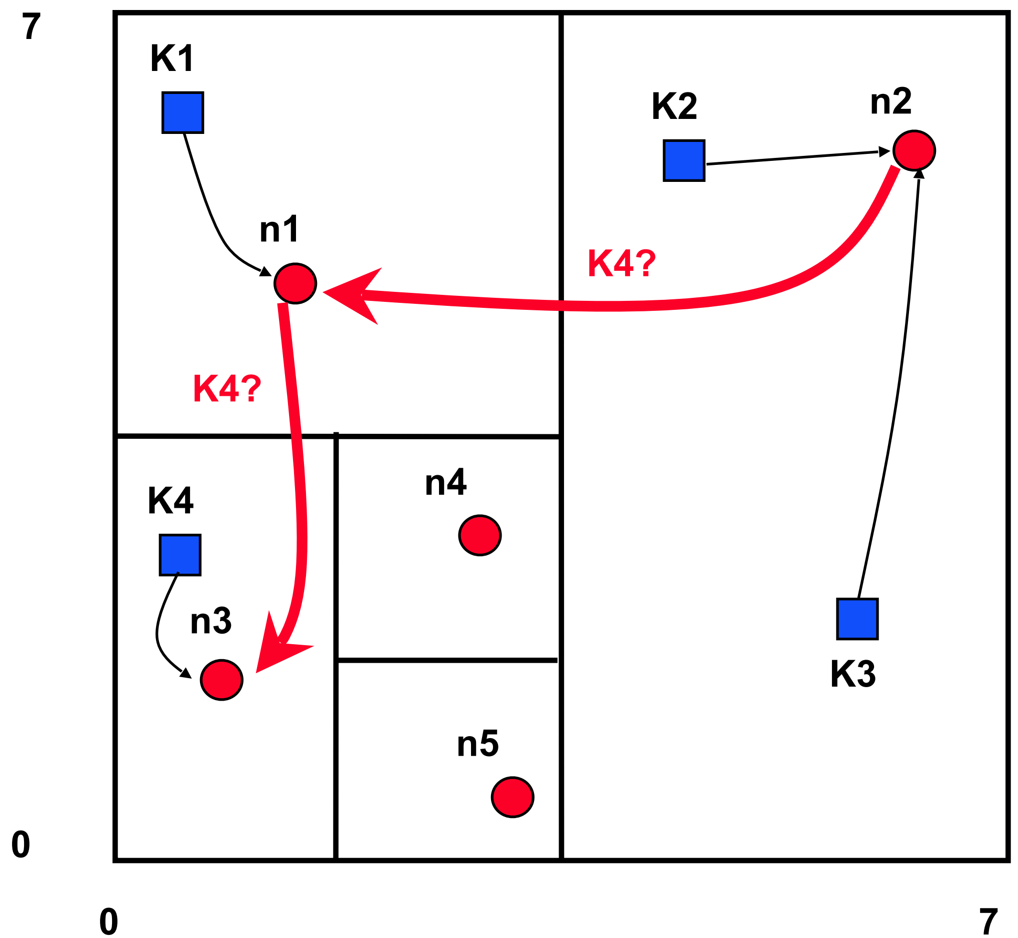
2-d CAN: Dynamic Space Division



2-d CAN: Key Assignment



2-d CAN: Routing (Lookup)



CAN: Routing

- Nodes keep $2d = O(d)$ state information (neighbor coordinates, IPs)
  - Constant, does not depend on number of nodes n
- Greedy routing
  - Route to the node that is closest to the destination
  - On average, is done in $O(n^{1/d}) = O(\log n)$ *when* $d = \log n / 2$

CAN: Node Join

- New node finds a node already in the CAN
  - (bootstrap: one (or a few) dedicated nodes outside the CAN maintain a partial list of active nodes)
- It finds a node whose zone will be split
  - Choose a random point P (will be its ID)
  - Forward a JOIN request to P through the existing node
- The node that owns P splits its zone and sends half of its routing table to the new node
- Neighbors of the split zone are notified

CAN: Node Leave, Fail

- **Graceful departure**
  - The leaving node hands over its zone to one of its neighbors
- **Failure**
  - Detected by the absence of heart beat messages sent periodically in regular operation
  - Neighbors initiate takeover timers, proportional to the volume of their zones
  - Neighbor with smallest timer takes over zone of dead node
    - notifies other neighbors so they cancel their timers (some negotiation between neighbors may occur)
  - Note: the (key, value) entries stored at the failed node are lost
    - Nodes that insert (key, value) pairs periodically refresh (or re-insert) them

CAN: Discussion

- **Scalable**
  - O(log n) steps for operations
  - State information is O(d) at each node
- **Locality**
  - Nodes are neighbors in the overlay, not in the physical network
  - Suggestion (for better routing)
    - Each node measures RTT between itself and its neighbors
    - Forwards the request to the neighbor with maximum ratio of progress to RTT

CAN: Discussion

- What is wrong with CAN (and DHTs in general)?
- Maintenance cost
  - Although logarithmic in number of nodes, still too much for very dynamic P2P systems
  - Peers are joining and leaving all the time

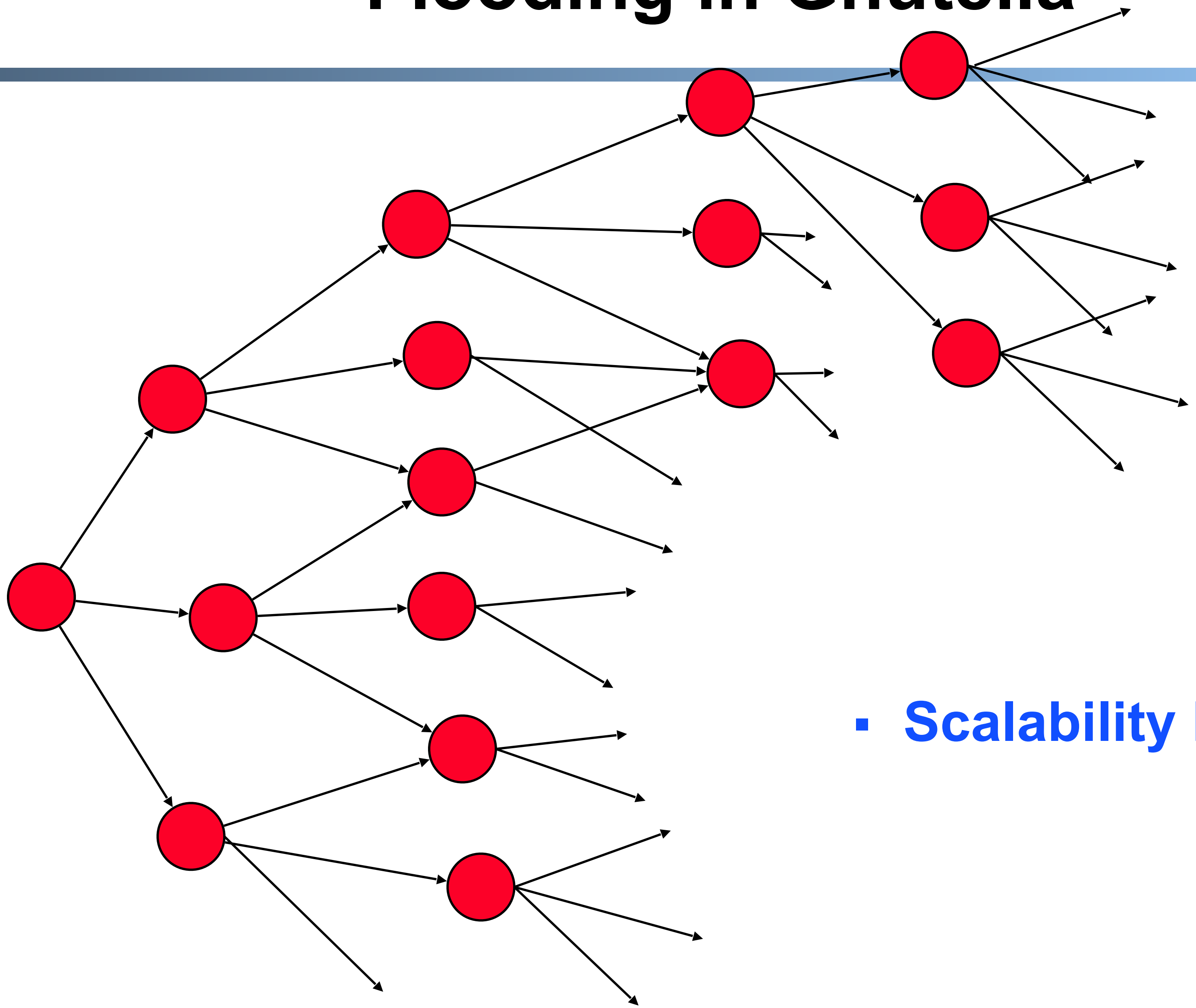
Unstructured P2P Substrates

- Objects can be anywhere → Loosely-controlled overlays
- The loose control
  - Makes overlay tolerate transient behavior of nodes
    - For example, when a peer leaves, nothing needs to be done because there is no structure to restore
  - Enables system to support flexible search queries
    - Queries are sent in plain text and every node runs a mini-database engine
- But, we loose on searching
  - Usually using flooding, inefficient
  - Some heuristics exist to enhance performance
  - No guarantee on locating a requested object (e.g., rarely requested objects)
- Ex: Gnutella, Kazaa (super node), GIA [Chawathe et al. 03]

Example: Gnutella

- Peers are called servents
- All peers form an unstructured overlay
- Peer join
  - Find an active peer already in Gnutella (e.g., contact known Gnutella hosts)
  - Send a Ping message through the active peer
  - Peers willing to accept new neighbors reply with Pong
- Peer leave, fail
  - Just drop out of the network!
- To search for a file
  - Send a Query message to all neighbors with a TTL (=7)
  - Upon receiving a Query message
    - Check local database and reply with a QueryHit to requester
    - Decrement TTL and forward to all neighbors if nonzero

Flooding in Gnutella



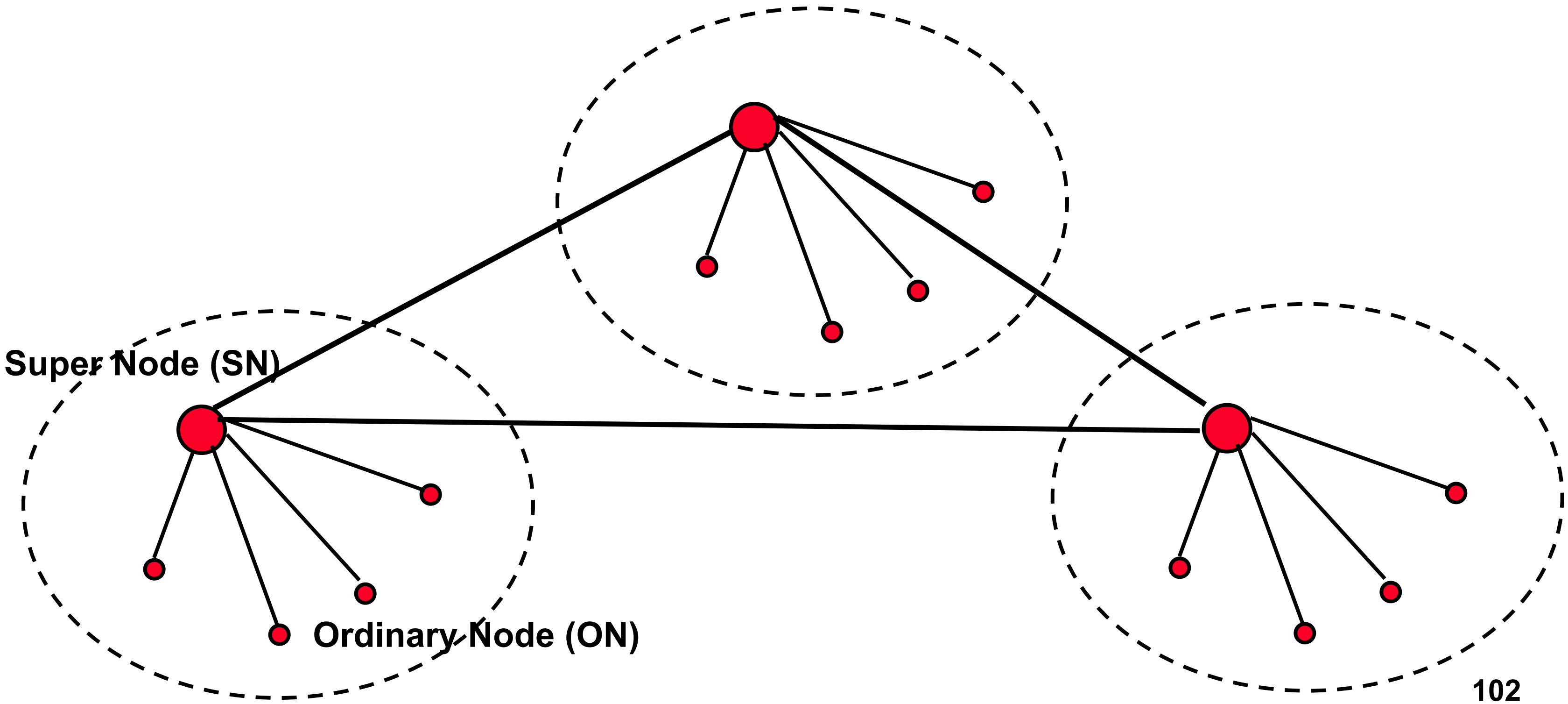
- **Scalability Problem**

Heuristics for Searching [Yang and Garcia-Molina 02]

- Iterative deepening
  - Multiple BFS with increasing TTLs
  - Reduce traffic but increase response time
- Directed BFS
  - Send to "good" neighbors (subset of your neighbors that returned many results in the past) → need to keep history
- Local Indices
  - Keep a small index over files stored on neighbors (within number of hops)
  - May answer queries on behalf of them
  - Save cost of sending queries over the network
  - Index currency?

Heuristics for Searching: Super Node

- Relatively powerful nodes play special role
  - maintain indexes over other peers



Super Node Systems

- File search
  - ON sends a query to its SN
  - SN replies with a list of IPs of ONs that have the file
  - SN may forward the query to other SNs
- Parallel downloads take place between ONs

Lessons from Deployed P2P Systems

- Distributed design
- Exploit heterogeneity
- Load balancing
- Locality in neighbor selection
- Connection Shuffling
  - If a peer searches for a file and does not find it, it may try later and gets it!
- Efficient gossiping algorithms
  - To learn about other SNs and perform shuffling
- Consider peers behind NATs and Firewalls
  - They are everywhere!

Network Layer: Summary

- **Network layer: forwarding and routing**
- **Routing: hierarchical**
  - **intra-AS: local, optimal**
  - **and Inter-AS (BGP): global, policy based**
- **Routing and protocols for Multicast**
- **IP Multicast vs. Application Layer Multicast**
- **Overlay Networks**

P2P Systems: Summary

- **In P2P computing paradigm:**
  - Peers cooperate to achieve desired functions
- **P2P characteristics**
  - heterogeneity, unreliability, rationality, scale, ad hoc
  - → new and lighter-weight algorithms are needed
- **Simple model for P2P systems:**
  - Peers form an abstract layer called overlay
  - A peer software client may have *three* components
    - P2P substrate, middleware, and P2P application
    - Borders between components may be blurred

Summary (cont'd)

- P2P substrate: key component, which
  - Manages the Overlay
  - Allocates and discovers objects
- P2P Substrates can be
  - Structured (DHT)
    - Example: CAN, Chord
  - Unstructured
    - Example: Gnutella