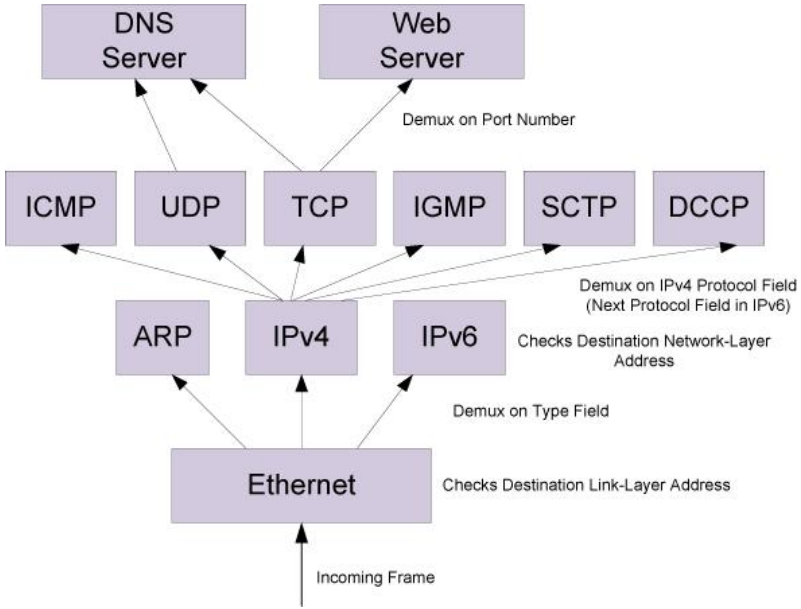
Internet

- Employs TCP/IP protocol suite



What it enables

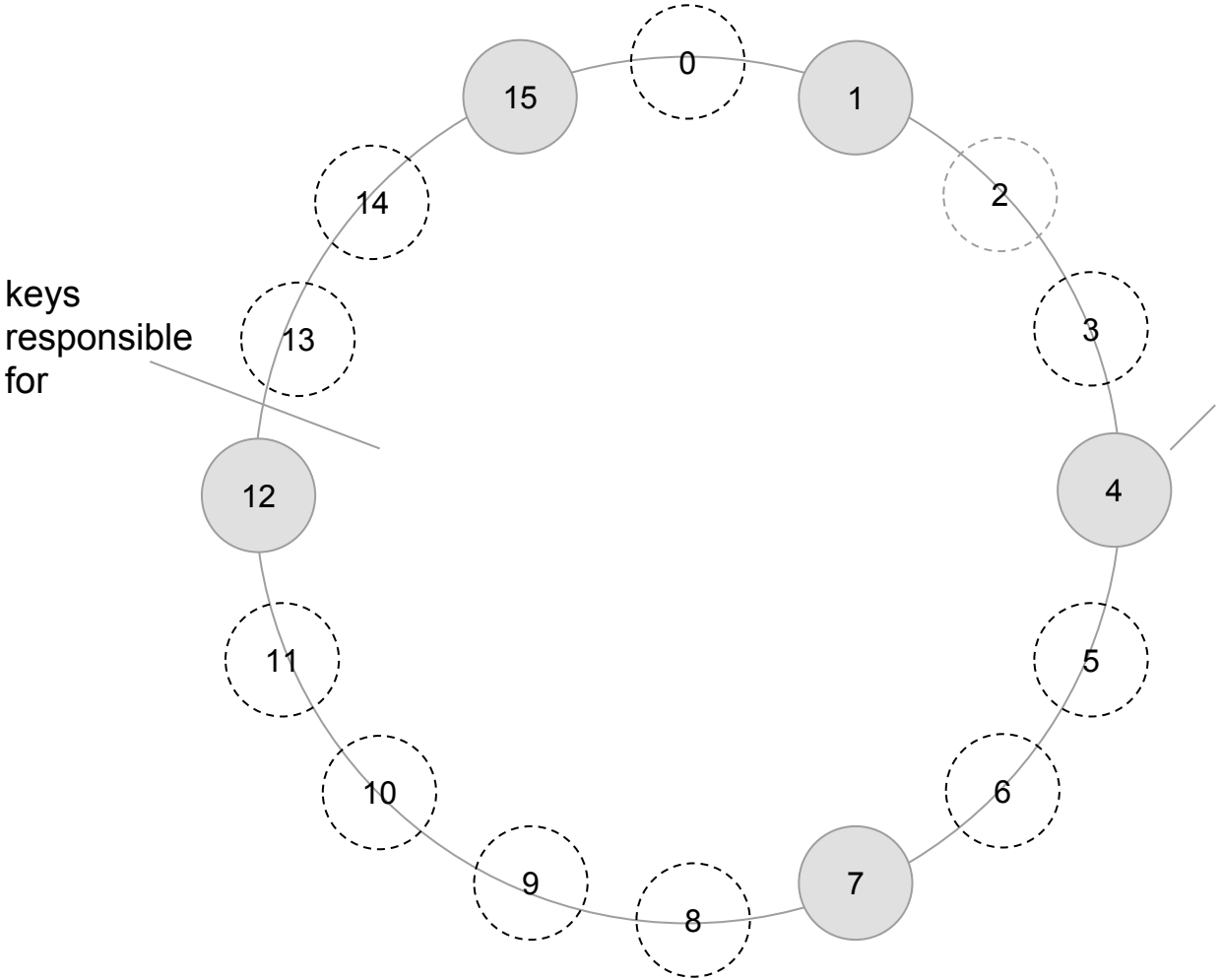
- End-to-end communication between agents of humans
  - E.g., OS processes
- UDP over IP = connectionless, packet-oriented, best-effort
- TCP over IP = connection-oriented, stream-oriented, reliable

- As an “application,” we can realize *overlay networks*.
  - E.g., p2p networks, with a Distributed Hash Table (DHT), e.g., Chord for finding content.

The Chord DHT

- *m*-bit *key* unique for every peer and piece of content.
- Define: *succ(k)* for any key *k* is peer that exists with smallest *id* ≥ *k*.
- Content with key *k* hosted by *succ(k)*.

- *m = 4*

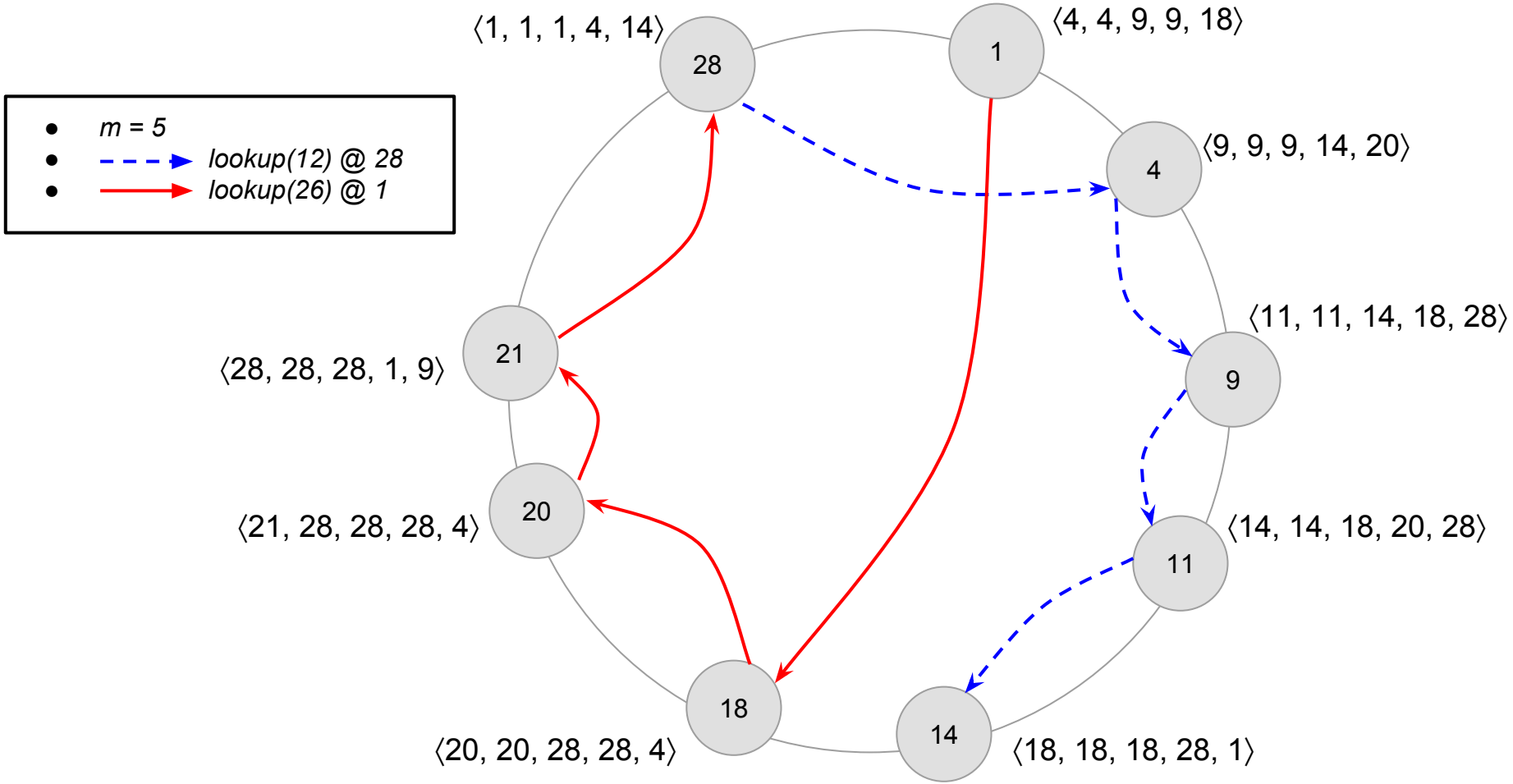


lookup(k)

- We may want to *lookup(k)* at any peer.
- Query is routed to *succ(k)*. How?

The Chord Approach

- Maintain a “finger table” (routing table) at Peer *p*, $FT_p[]$.
- *m* entries
- $FT_p[i] = succ(p + 2^{i-1})$, for all $i \in [1,m]$


m = 5
lookup(12) @ 28
lookup(26) @ 1
28
⟨1, 1, 1, 4, 14⟩
1
⟨4, 4, 9, 9, 18⟩
4
⟨9, 9, 9, 14, 20⟩
9
⟨11, 11, 14, 18, 28⟩
11
⟨14, 14, 18, 20, 28⟩
14
⟨18, 18, 18, 28, 1⟩
18
⟨20, 20, 28, 28, 4⟩
20
⟨21, 28, 28, 28, 4⟩
21
⟨28, 28, 28, 1, 9⟩

Lookup algorithm

- From the midterm preview:

query for key q at peer with ID p; finger table has e entries; $q, p \in [0, 2^m - 1]$, $e \in [1, m]$

1 **if** p *hosts the piece of content which has key* q **then return** p
2 **if** $p < q < FT_p[1] \pmod{2^m}$ **then return** $FT_p[1]$
3 **if** $q \geq FT_p[e] \pmod{2^m}$ **then return** $FT_p[e]$
4 **return** $FT_p[i]$ such that $FT_p[i] \leq q < FT_p[i+1] \pmod{2^m}$

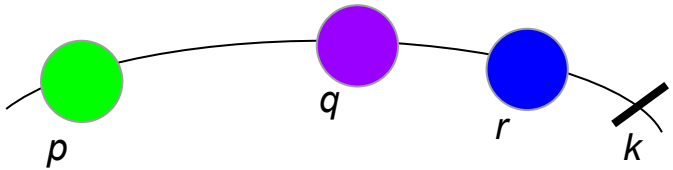
Claims

- We go around the circle at most once
  - Termination guaranteed
- Expected number of “hops” is *O(log n)*, where *n* = # peers.
  - Each hop covers half the arc-length.
  - Hand-wave: peers are equidistant under randomness assumption.

Each hop covers > half the arc-length…

Claim:

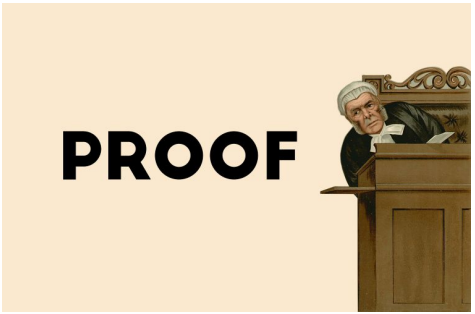
Suppose we do *lookup*(k) @ peer p. Assume that r is the peer immediately before *succ*(k). Suppose that the next hop at p is q. Then: q - p > (r - p)/2.



Proof

Suppose $q = FT_p[j]$.

Then, $q \geq p + 2^{j-1}$. And, $r < p + 2^j$.



Worst-case # hops

- Above assertion guarantees that *m* is the worst-case # hops in a lookup.
  - In Assignment 2 we established a slightly softer upper-bound, *m+1*.

IP Addressing, forwarding

IP Address

- 32-bit
- Written as *a.b.c.d*, where each of *a*, *b*, *c*, *d* ∈ [0,255]

Dotted-Quad Representation	Binary Representation
0.0.0.0	00000000 00000000 00000000 00000000
1.2.3.4	00000001 00000010 00000011 00000100
10.0.0.255	00001010 00000000 00000000 11111111
165.195.130.107	10100101 11000011 10000010 01101011
255.255.255.255	11111111 11111111 11111111 11111111

IP Address, contd.

- An IP address is not flat

- In modern times, we use Classless Inter-Domain Routing (CIDR)
  - *m* can be any value
  - In the past, *m* could be only certain values: 8, 16 and 24.

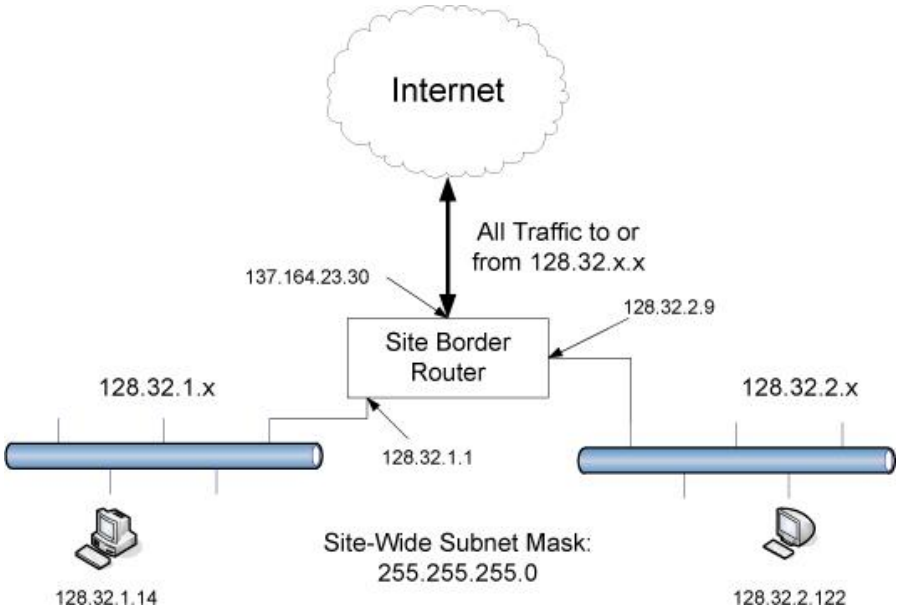
IP Address, contd.

- Length of network ID portion identified with a *mask* or *prefix-length*

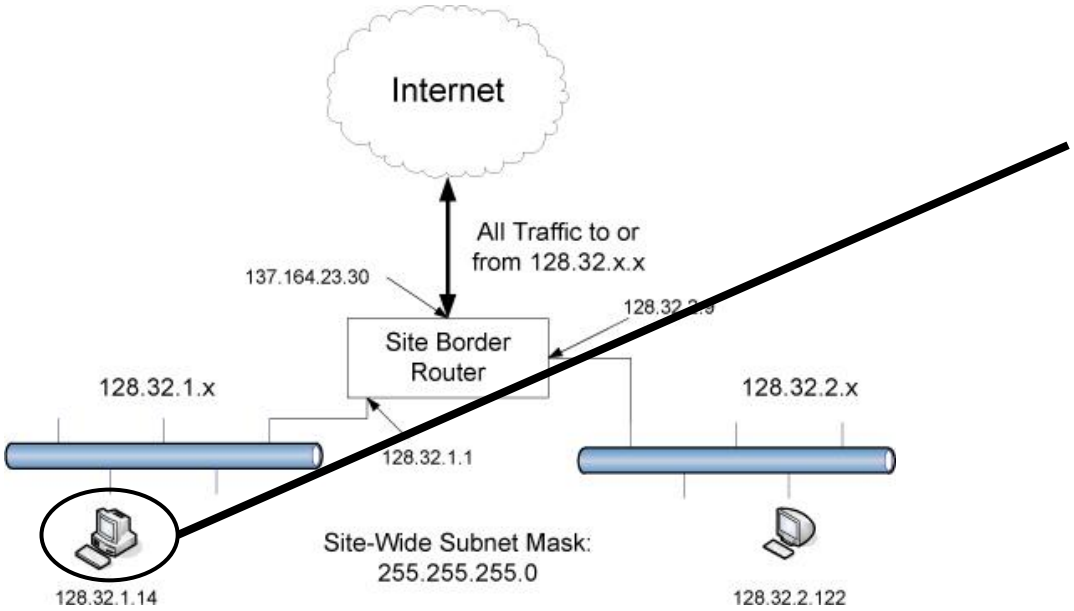
Dotted-Decimal Representation	Shorthand (Prefix Length)	Binary Representation
128.0.0.0	/1	10000000 00000000 00000000 00000000
255.0.0.0	/8	11111111 00000000 00000000 00000000
255.192.0.0	/10	11111111 11000000 00000000 00000000
255.255.0.0	/16	11111111 11111111 00000000 00000000
255.255.254.0	/23	11111111 11111111 11111110 00000000
255.255.255.192	/27	11111111 11111111 11111111 11100000
255.255.255.255	/32	11111111 11111111 11111111 11111111

- To express network ID only, convention is to write 0's for host ID.
  - E.g., 128.6.12.0/22

Example of IP subnetting

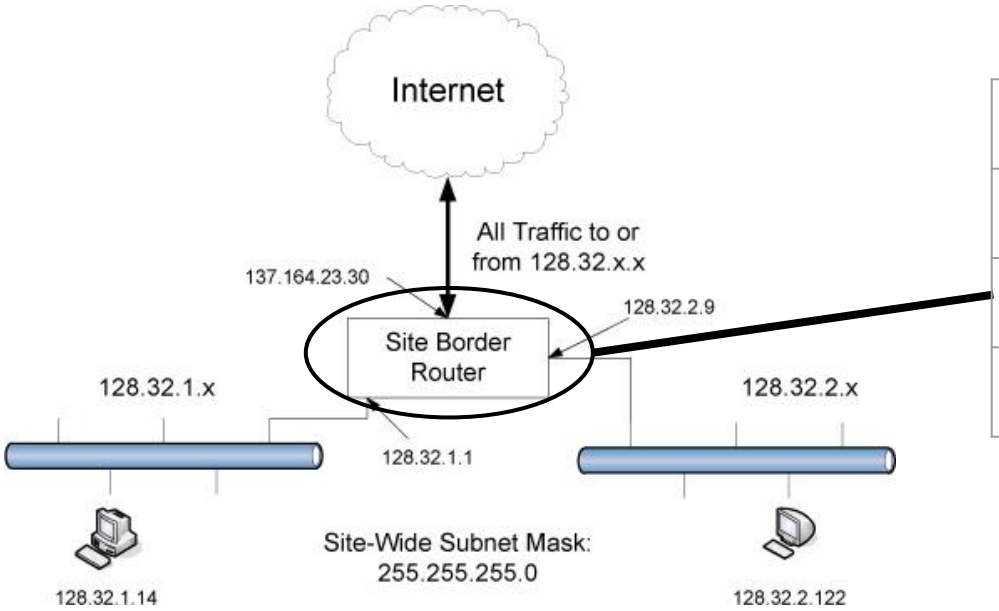


Routing or forwarding table



Destination	*Next hop*
128.32.1.0/24	myself
0.0.0.0/0	128.32.1.1

Routing or forwarding table



Destination	*Next hop*	*Interface*
128.32.1.0/24	myself	(128.32.1.1)
128.32.2.0/24	myself	(128.32.2.9)
0.0.0.0/0	137.x.y.z	(137.164.23.30)

Lookup algorithm

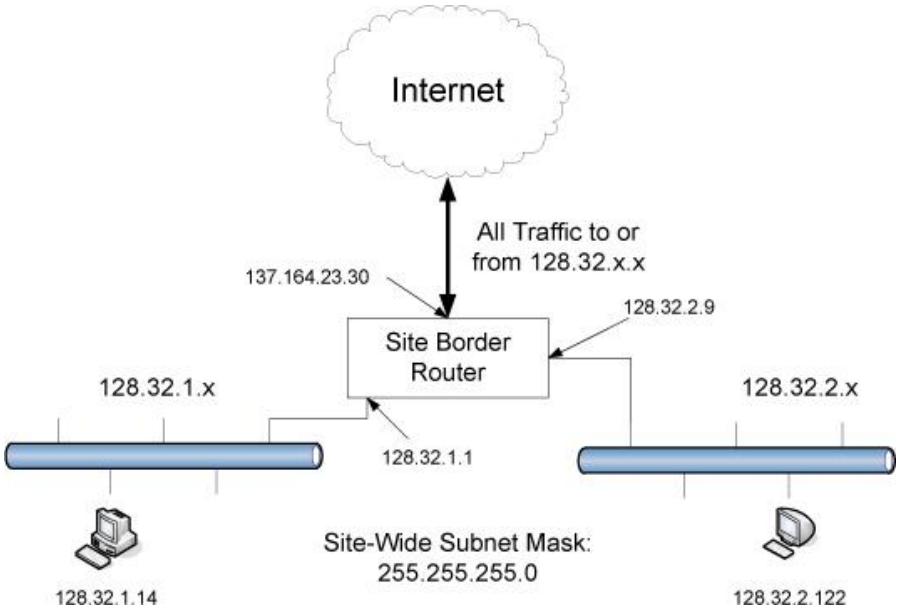
- Longest prefix match of destination IP address
- E.g., destination 128.32.2.20 matches both 128.32.2.0/24, 128.0.0.0/8 and 0.0.0.0/0
- But 128.32.2.0/24 is longest prefix match

IP forwarding table lookup- example

Destination	Mask	Next hop	Interface
0.0.0.0	0.0.0.0	10.0.0.1	10.0.0.100
10.0.0.0	255.255.255.128	10.0.0.100	10.0.0.100

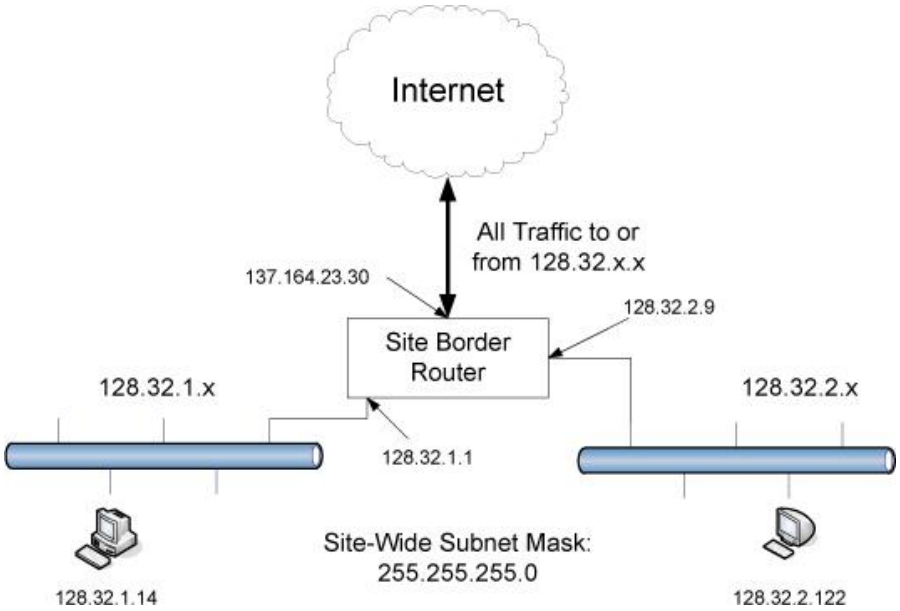
- 10.0.0.19 matches both entries. 2nd entry is best-match.
- 178.162.3.4 matches 1st entry, not 2nd
- 10.0.0.131 matches 1st entry, not 2nd

ARP: IP to MAC address mapping



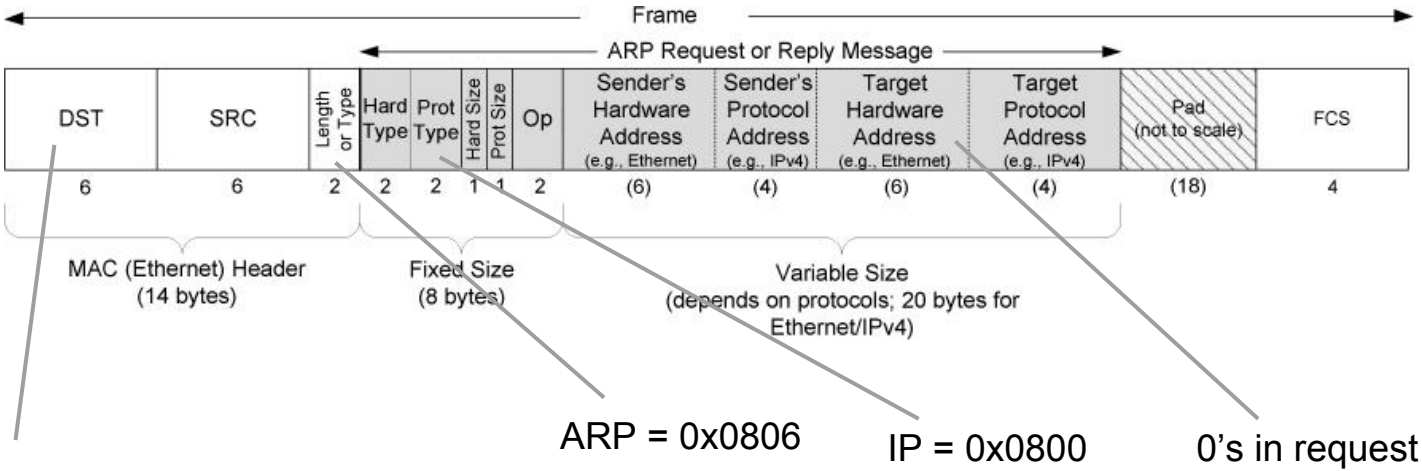
- Want to send an IP packet from 128.32.1.14 to 128.32.2.122
- We need to traverse from one subnet to another
- 128.32.1.14 first sends packet to 128.32.1.1
- Router uses internal switching fabric to forward packet to (128.32.2.9) interface.
- 128.32.2.9 sends to 128.32.2.122

ARP, contd.



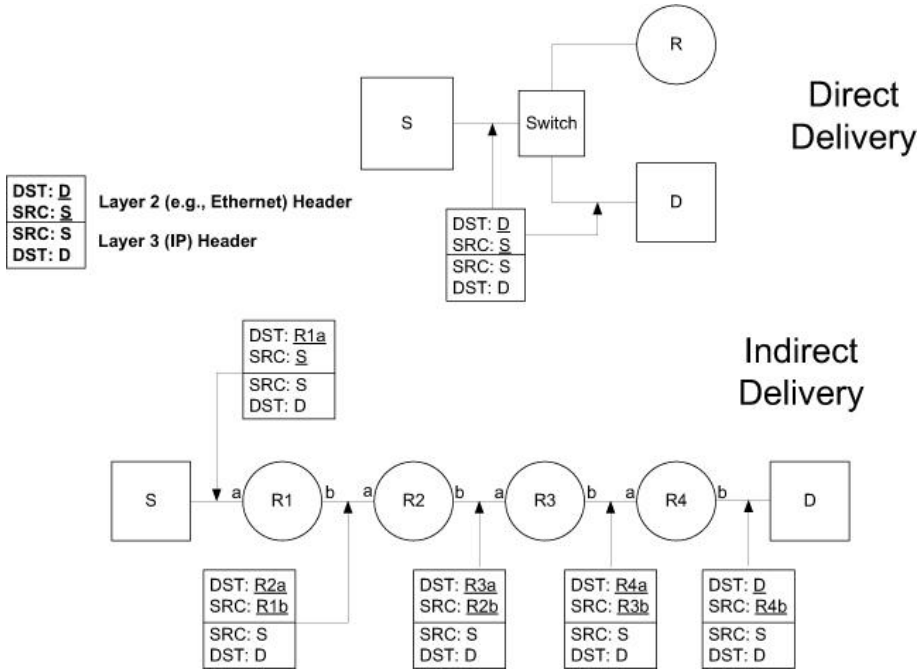
- Want to send an IP packet from 128.32.1.14 to 128.32.2.122
- Src, Dst IP addresses do not change
- But we need to use Link layer, e.g., Ethernet to send within a subnet first.
- Process:
    a. Host to the left looks up routing table, discovers that for dest 128.32.2.122 must send to 128.32.1.1
    b. It sends out an ARP request, gets MAC address of 128.32.1.1
    c. Sends IP packet encapsulated in Ethernet frame with dst Ethernet address = that of 128.32.1.1

ARP Frame encapsulated in Ethernet Frame



DST = link-layer broadcast address in a request. Unicast in a response.

IP forwarding - “direct” vs. “indirect” delivery



How is routing table instantiated?

- Manually
- DHCP + simple algorithm
- Routing algorithm + protocol

Two broad routing-algorithm strategies

- Link-state routing
- Distance-vector routing

Routing algorithm classification

Q: global or decentralized information?

global:

- all routers have complete topology, link cost info
- "link state" algorithms

decentralized:

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

Q: static or dynamic?

static:

- routes change slowly over time

dynamic:

- routes change more quickly
  - periodic update
  - in response to link cost changes

Distance vector algorithm

Bellman-Ford equation (dynamic programming)

let

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

then

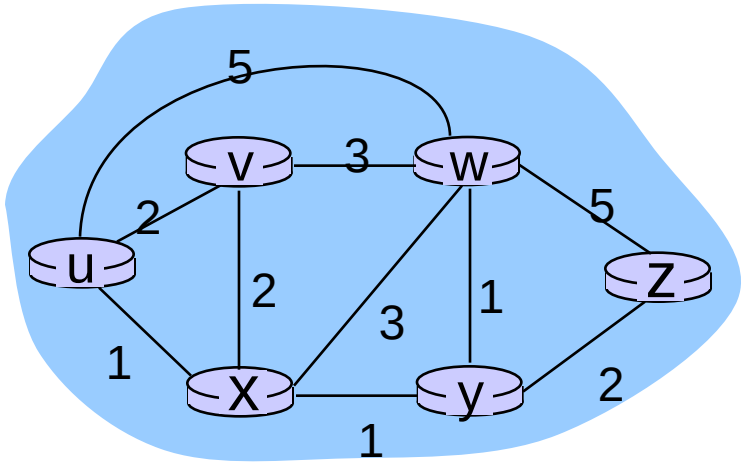
$$d_x(y) = \min_v \{c(x,v) + d_v(y)\}$$

cost from neighbor v to destination y

cost to neighbor v

min taken over all neighbors v of x

Bellman-Ford example



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

B-F equation says:

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

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 neighbors' distance vectors. For each neighbor v, x maintains $\mathbf{D}_v = [D_v(y): y \in N]$

Distance vector algorithm

key idea:

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

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

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

Distance vector algorithm

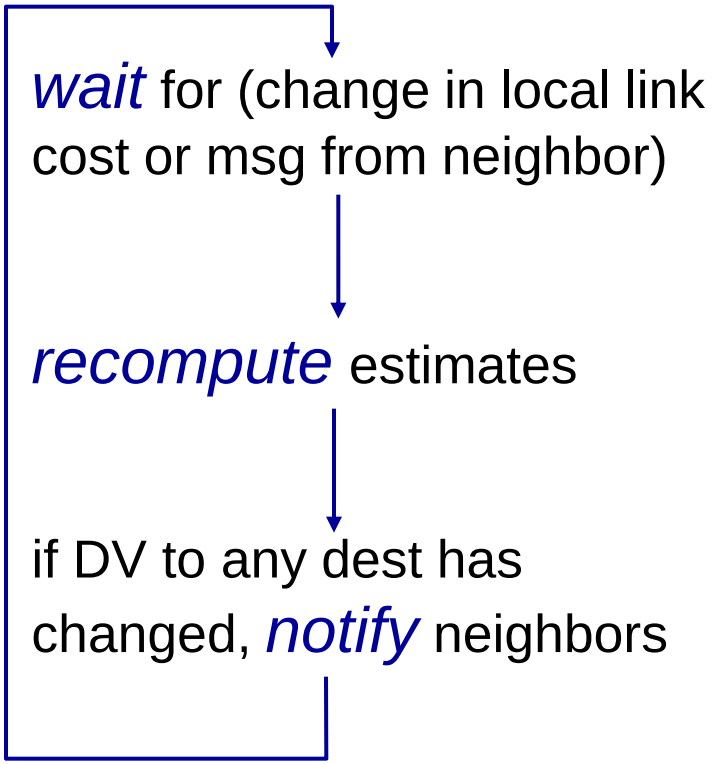
iterative, asynchronous:
each local iteration caused by:

- local link cost change
- DV update message from neighbor

distributed:

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

each node:



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

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

node x table

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

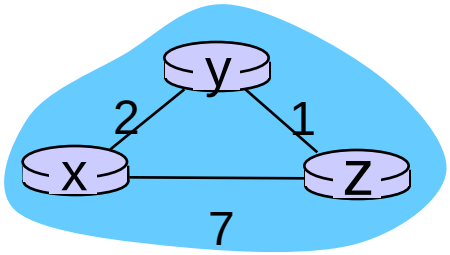
from	cost to x	y	z
x	0	2	3
y	2	0	1
z	7	1	0

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

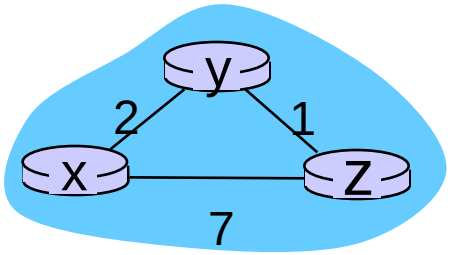
from \ cost to	x	y	z
x	0	2	3
y	2	0	1
z	3	1	0

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

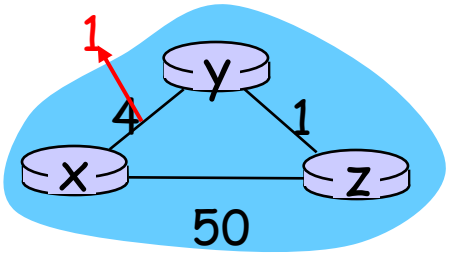


time

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.

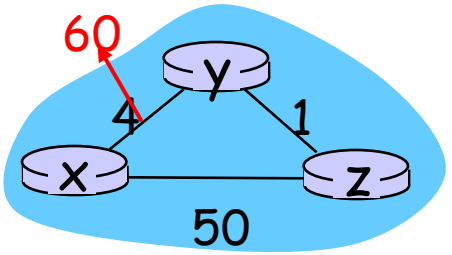
t_1 : *z* receives update from *y*, updates its table, computes new least cost to *x* , sends its neighbors its DV.

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: see text



poisoned reverse:

- If Z routes through Y to get to X :
  - Z tells Y its (Z's) distance to X is infinite (so Y won't route to X via Z)
- will this completely solve count to infinity problem?