IP VERSION 6

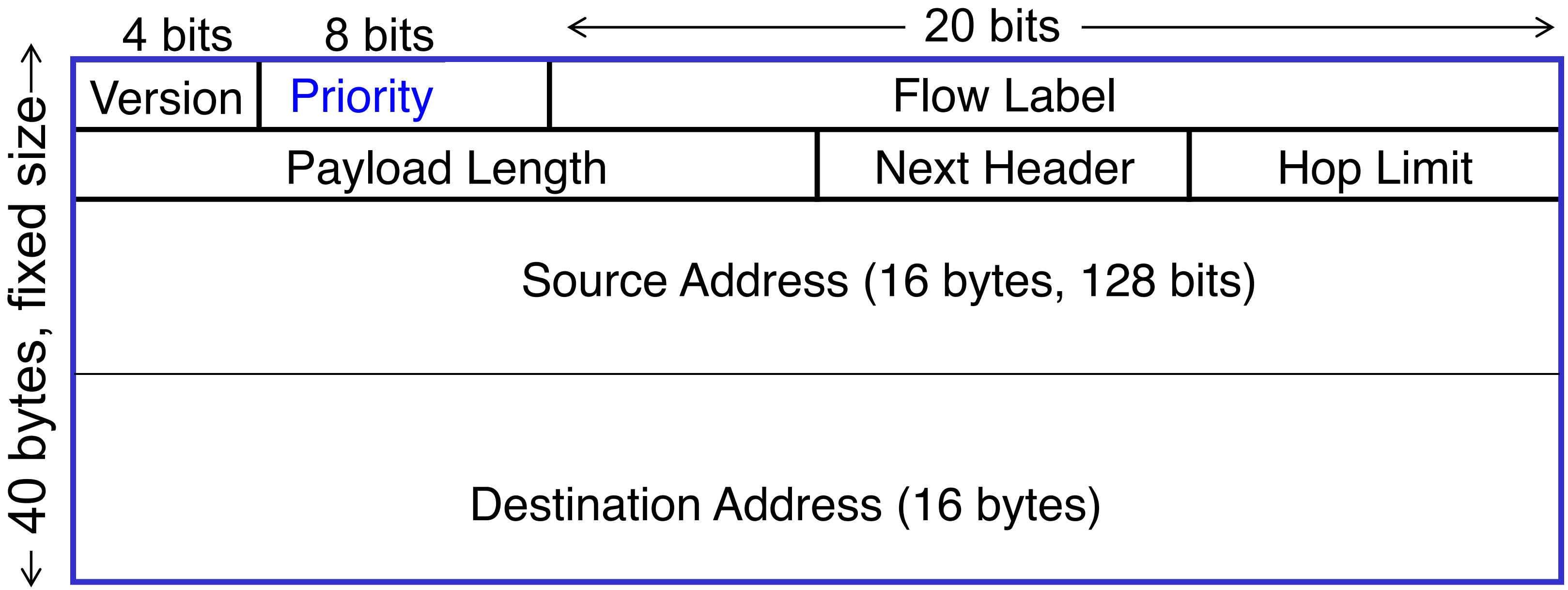
https://tools.ietf.org/html/rfc4291

IPv6

- Motivation: 32-bit address space exhaustion
- Take the opportunity for some clean-up
- IPv6 packet format:
  - Fixed-length 40byte header, length field excludes header; Header Length field eliminated
  - Address length: 32 bits → 128 bits
  - fragmentation fields & IP options: moved out of base header
  - Header Checksum eliminated
  - Type of Service → Traffic Class
  - TTL →Hop Limit, Protocol → Next Header
  - added Flow Label field

important

IPv6 header format



Changes from IPv4:

Priority: *usage yet to be finalized*

Flow Label: identify packets in same "flow"

(but flow is yet to be defined)

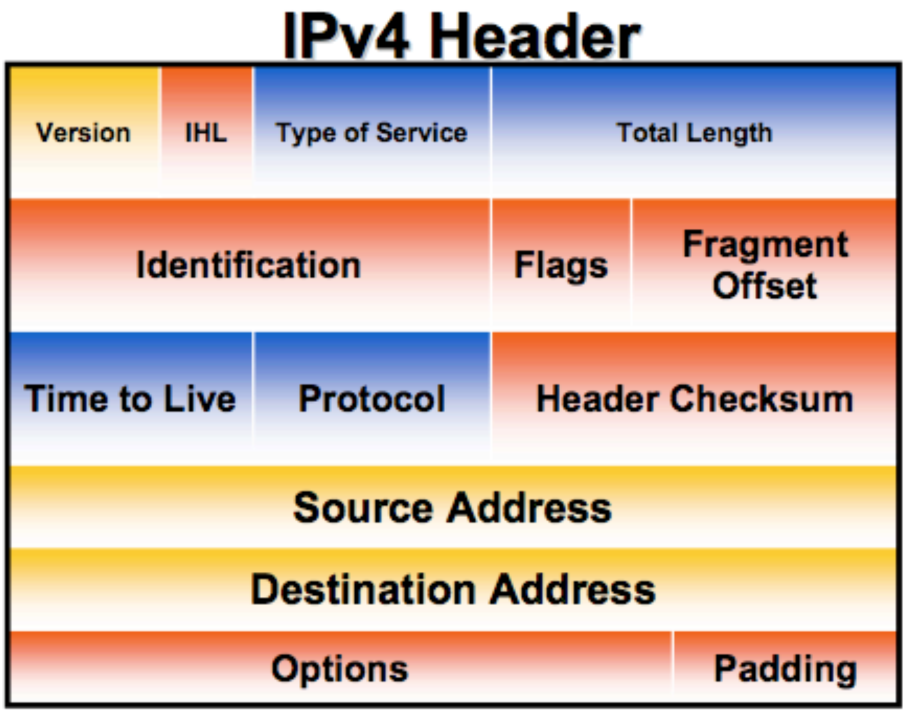
Next header: identify upper layer protocol for data

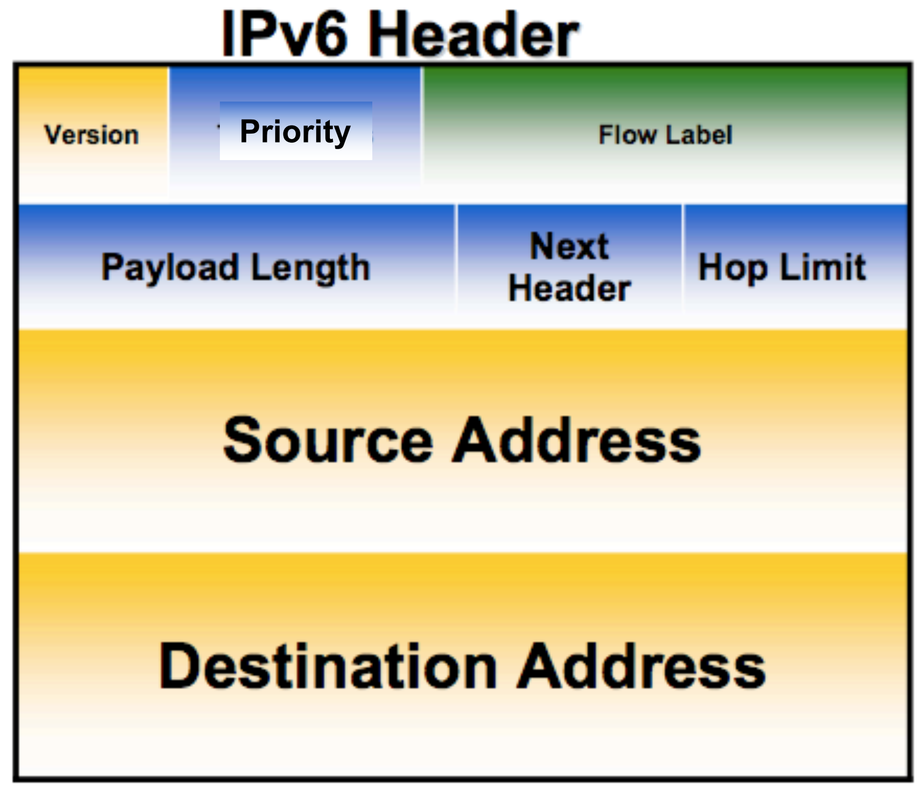
Options: outside of the basic header, indicated by "Next Header" field

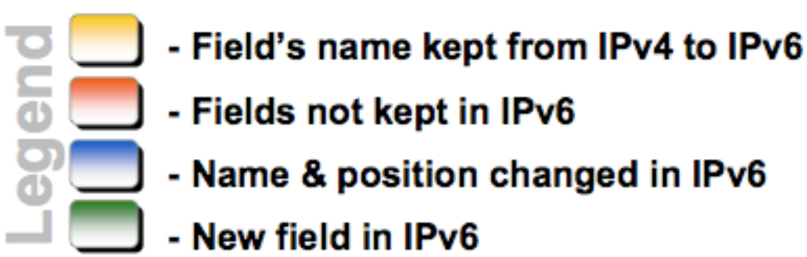
Header Checksum: removed

IPv4 : IPv6 Header Comparison

important







Options: outside the basic header

• indicated by "Next Header" field

Checksum: removed

What about IP options (and other things)?

FYI

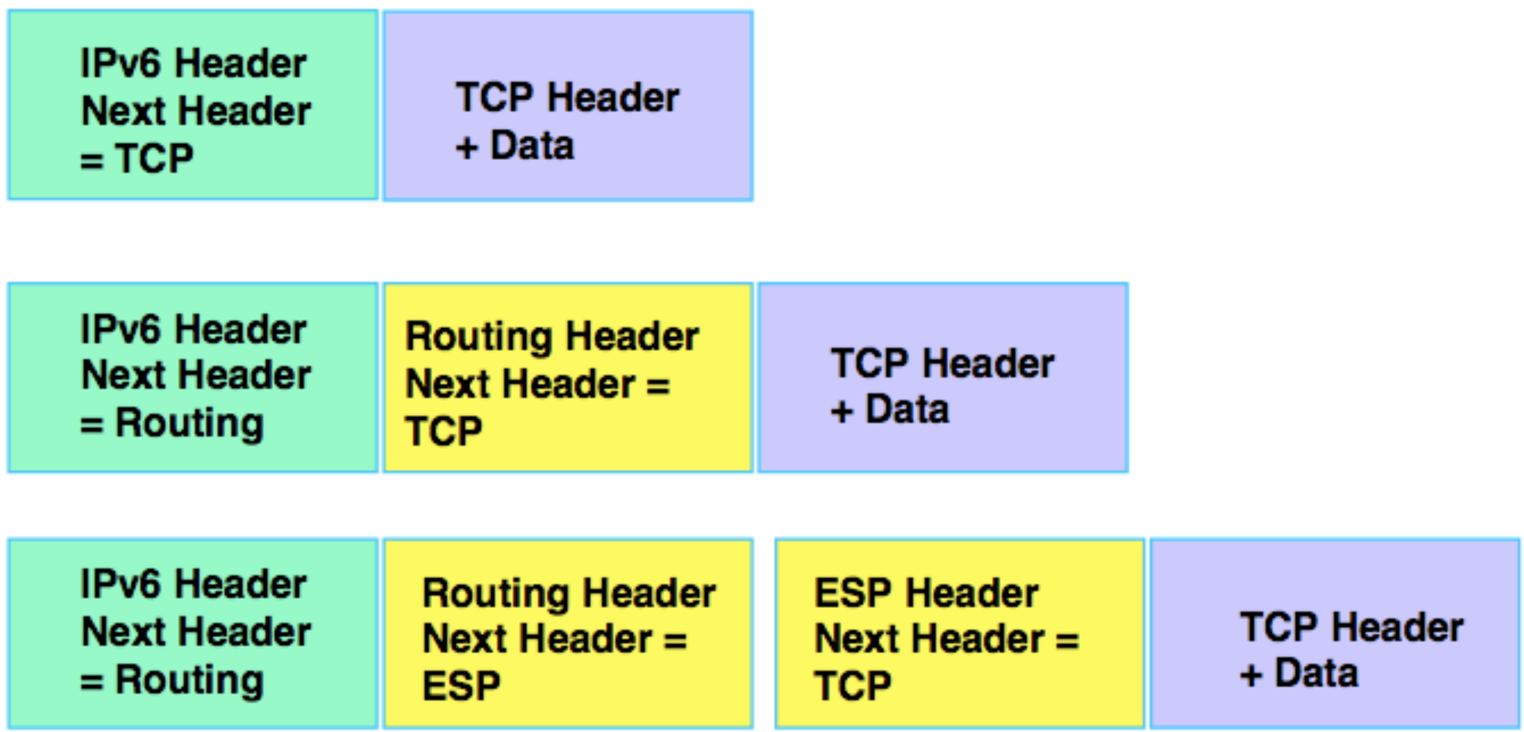
IPv6 has extension header

- Routing: Loose or tight source routing
- Fragmentation: only source can fragment
- Authentication
- Hop-by-Hop Options
- Most extension headers are examined only at destination

Encoding options in IPv6 header

FYI

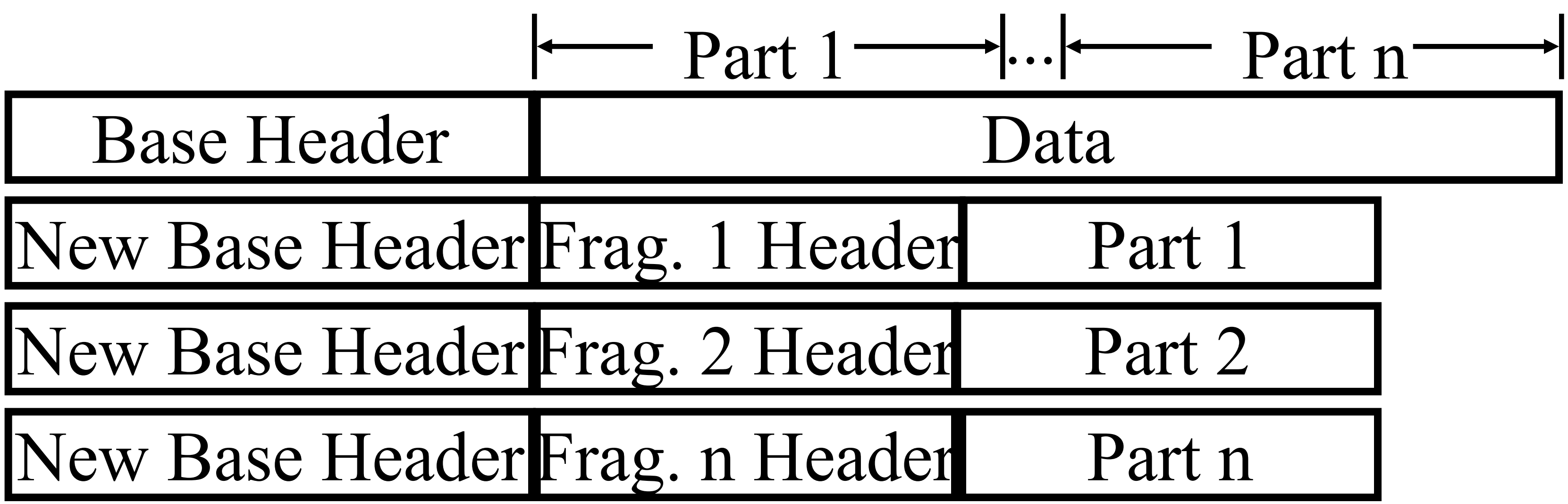
- Basic header: fixed length
- "next header" field specifies how long it may be
- Daisy chained



FYI: IP fragmentation in IPv6

FYI

- Only source host can fragment ⇒ need path MTU discovery
- Fragmentation requires an extension header
  - Payload is divided into pieces
  - A new base header is created for each fragment



FYI: how to write IPv6 addresses?

- **IPv6: Colon-Hex:**

  2607:F010:03f9:0000:0000:0000:0004:0001

  - Can skip leading zeros of each word

    2607:F010:3f9:0:0:0:4:1

  - Can skip **one** sequence of zero words (compressed representation), e.g.,

    2607:f010:3f9**::**4:1

  - Can leave the last 32 bits in dot-decimal

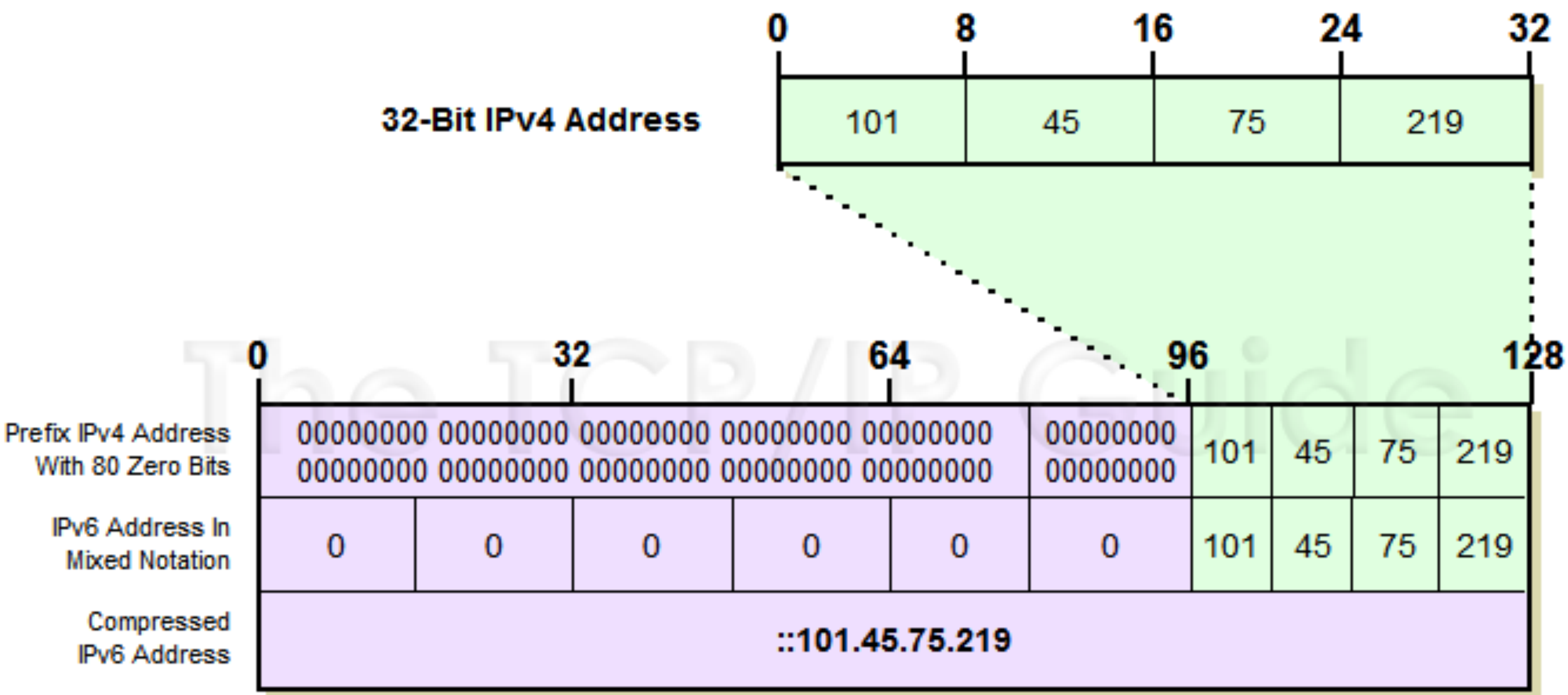
    2607:f010:3f9**::0.4.0.1**

  - Can specify a prefix by /length

    - 2607:f010:3f9::**/64**

IPv4 vs. IPv6

FYI



IPv6 Special Addresses

FYI

- ::/128 - Unspecified
- ::1/128 - Loopback
- ::ffff:0:0/96 - IP4-mapped address
- 2002::/16 - 6to4
- ff00::/8 - Multicast
- fe80::/10 - Link-Local Unicast

- no broadcast addresses, function superseded by multicast

http://www.iana.org/assignments/iana-ipv6-special-registry/iana-ipv6-special-registry.xhtml

IPv6 Address Calculations

- Represent in compressed representation and with IPv4-dot notation of last 32 bits
  - `2001:4860:4860:0000:0000:0000:0000:8888`
  - `2a03:2880:2040:7f21:face:b00c:0000:25de`
  - `2620:0000:0ccc:0000:0000:0000:0000:0002`
  - `0000:0000:0000:0000:0000:0000:0000:0001`
  - `2605:e000:1521:0073:f4f0:6edb:fa4e:ed6f`
- Show the expanded representation
  - 2607:f010:bfc:e009::2/64
  - ::ffff:131.179.196.70

A Little Bit More

- Number of addresses in the network, first address, last address
  - 2607:f010:bfc:e009::2/64
  - 2620:0:1c00::/40
  - 2620:107:3000::/44
  - 2600:1406:32::/48

Ways to Represent IP Address

FYI

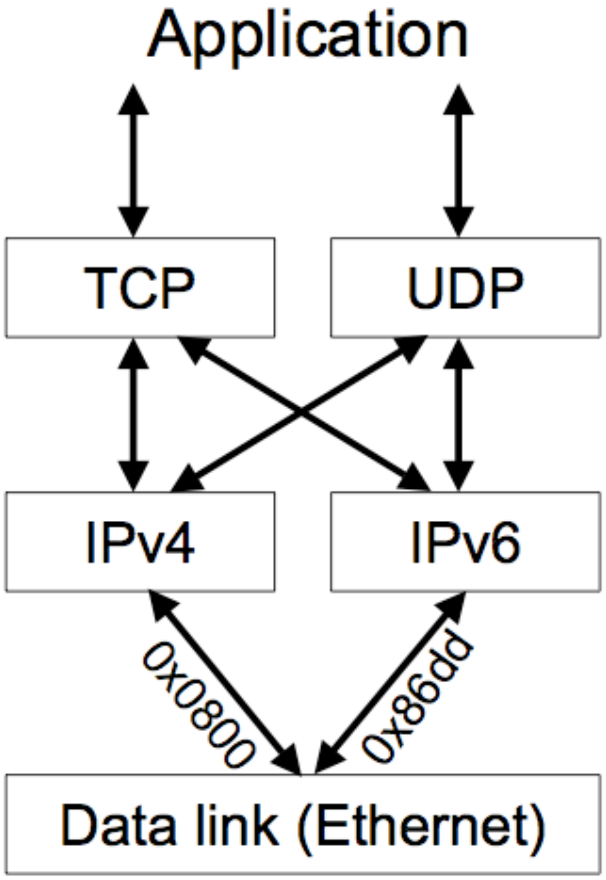
- **Dot-Decimal**
  - **131.179.196.70**
- Dot-Hexadecimal
  - 0x83.0xb3.0xC4.0x46
- Doc-Octal
  - 0203.0263.0304.0106
- Decimal
  - 2209596486
- Hexadecimal
  - 0x83B3C446
- Octal
  - 020354742106

FYI: Transition From IPv4 To IPv6

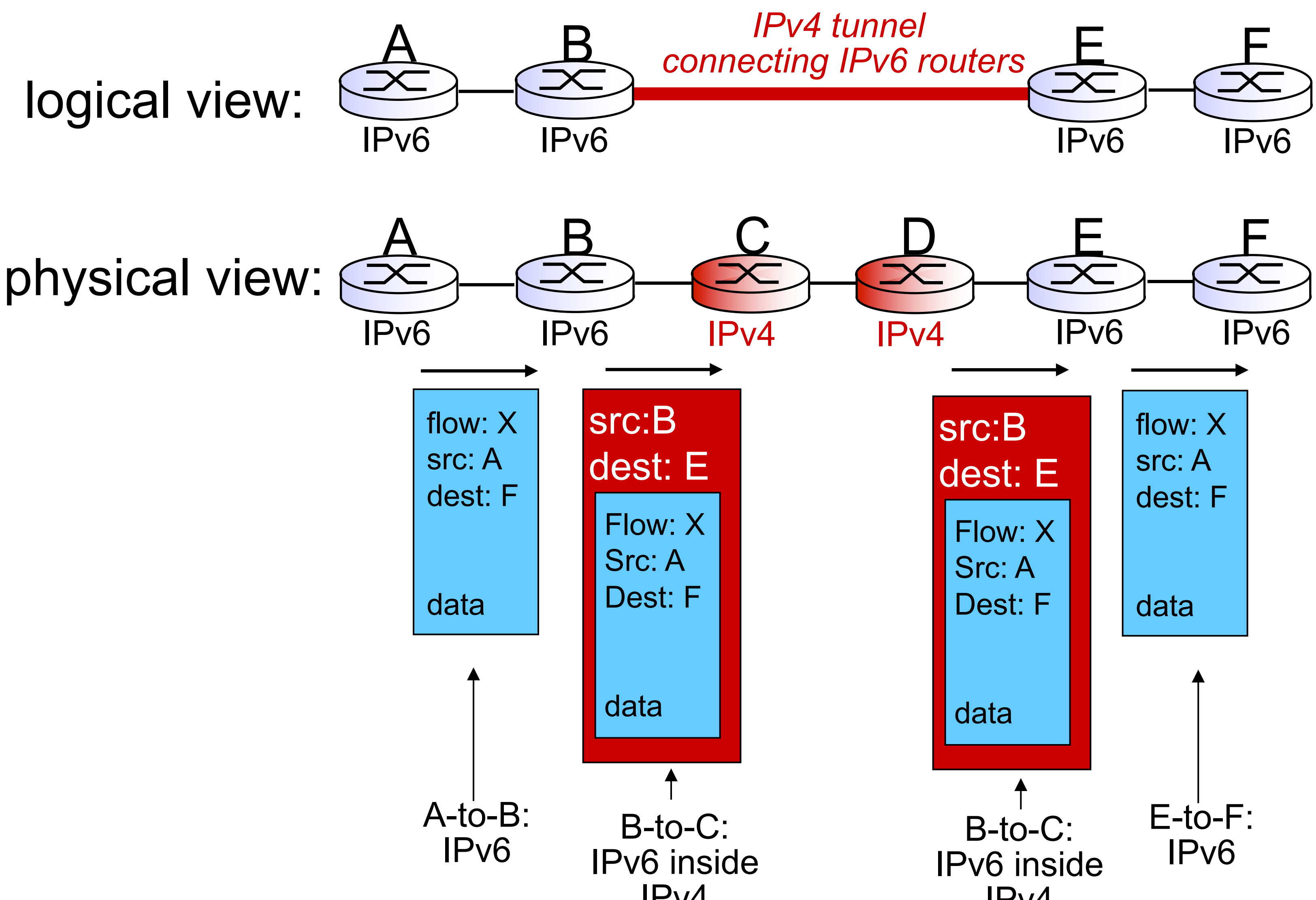
- Not all routers can be upgraded simultaneous
- Must allow the Internet operate with mixed IPv4 and IPv6 routers

Solution: Dual stack

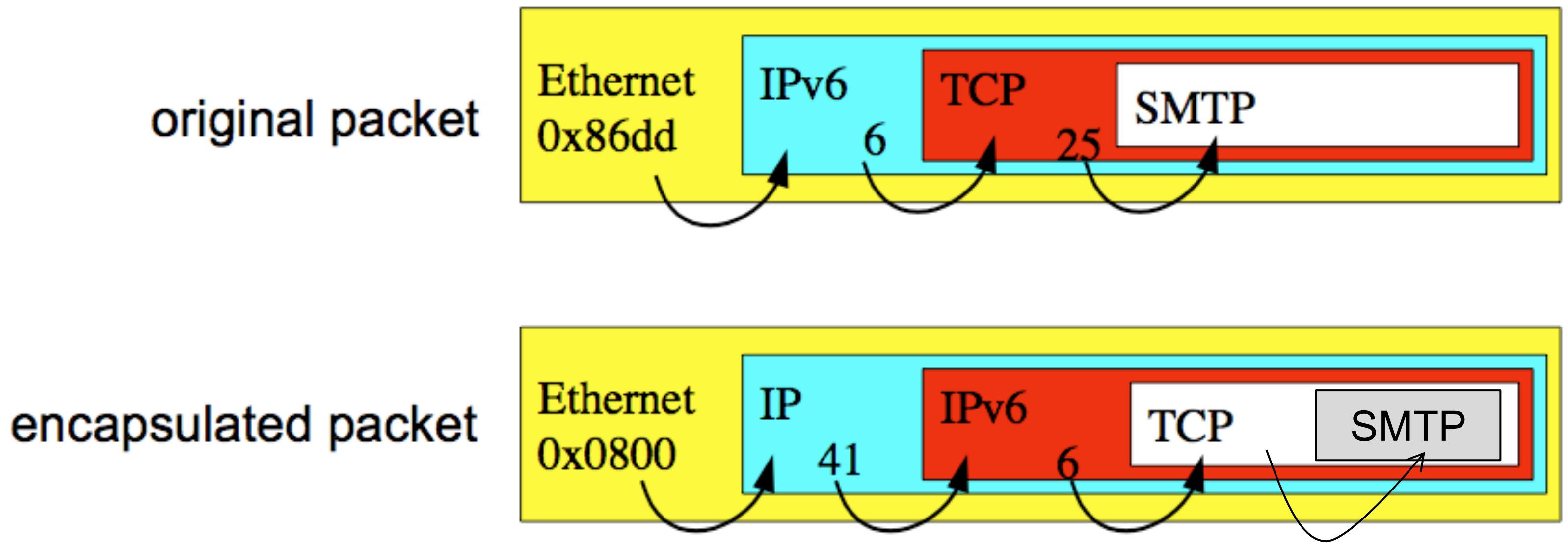
- Drawbacks: Doesn't solve the lack of IPv4 addresses

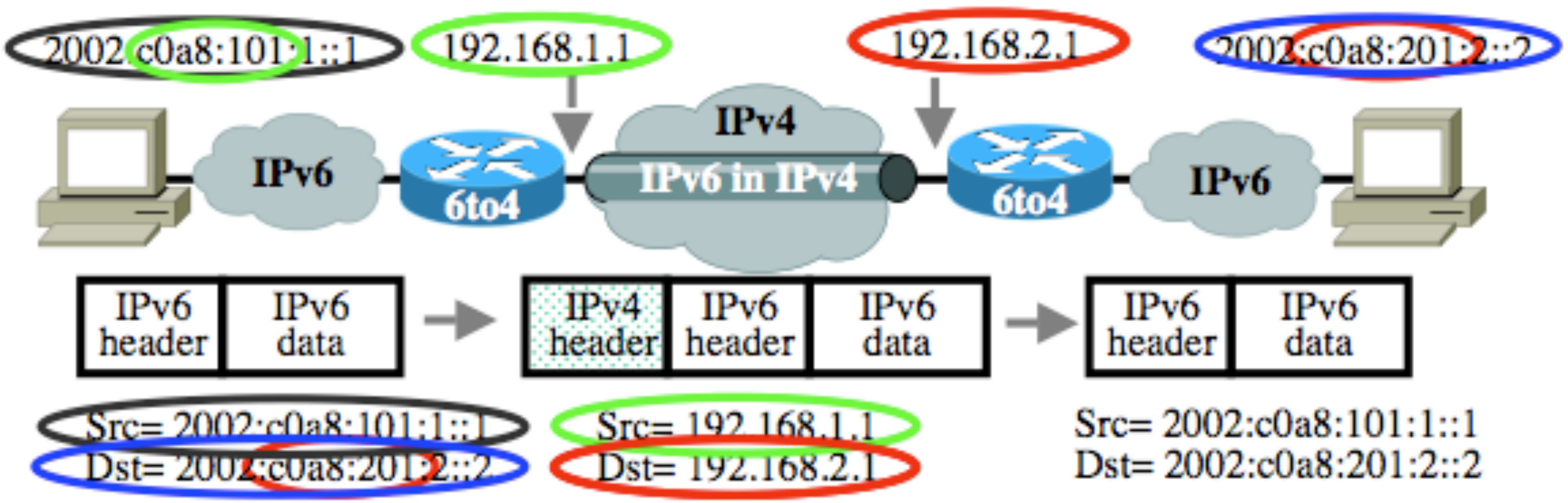


Transition IPv4 → IPv6: tunneling



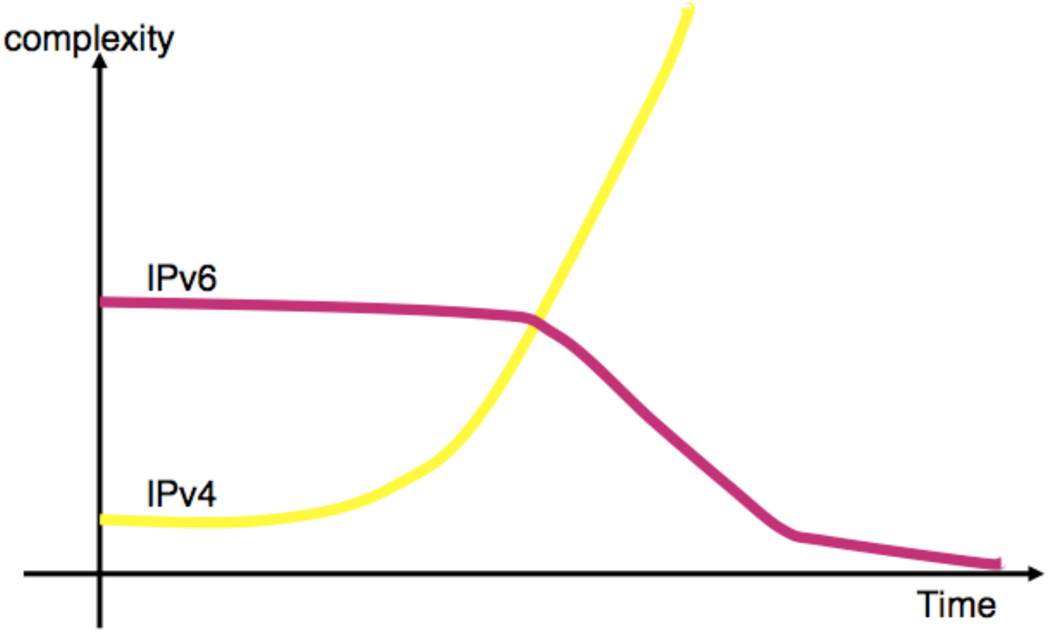
Tunneling


original packet
Ethernet
0x86dd
IPv6
6
TCP
25
SMTP
encapsulated packet
Ethernet
0x0800
IP
41
IPv6
6
TCP
SMTP


2002:c0a8:101:1::1
192.168.1.1
192.168.2.1
2002:c0a8:201:2::2
IPv6
6to4
IPv4
IPv6 in IPv4
6to4
IPv6
IPv6 header
IPv6 data
IPv4 header
IPv6 header
IPv6 data
IPv6 header
IPv6 data
Src= 2002:c0a8:101:1::1
Dst= 2002:c0a8:201:2::2
Src= 192.168.1.1
Dst= 192.168.2.1
Src= 2002:c0a8:101:1::1
Dst= 2002:c0a8:201:2::2

Will IPv6 get deployed eventually?

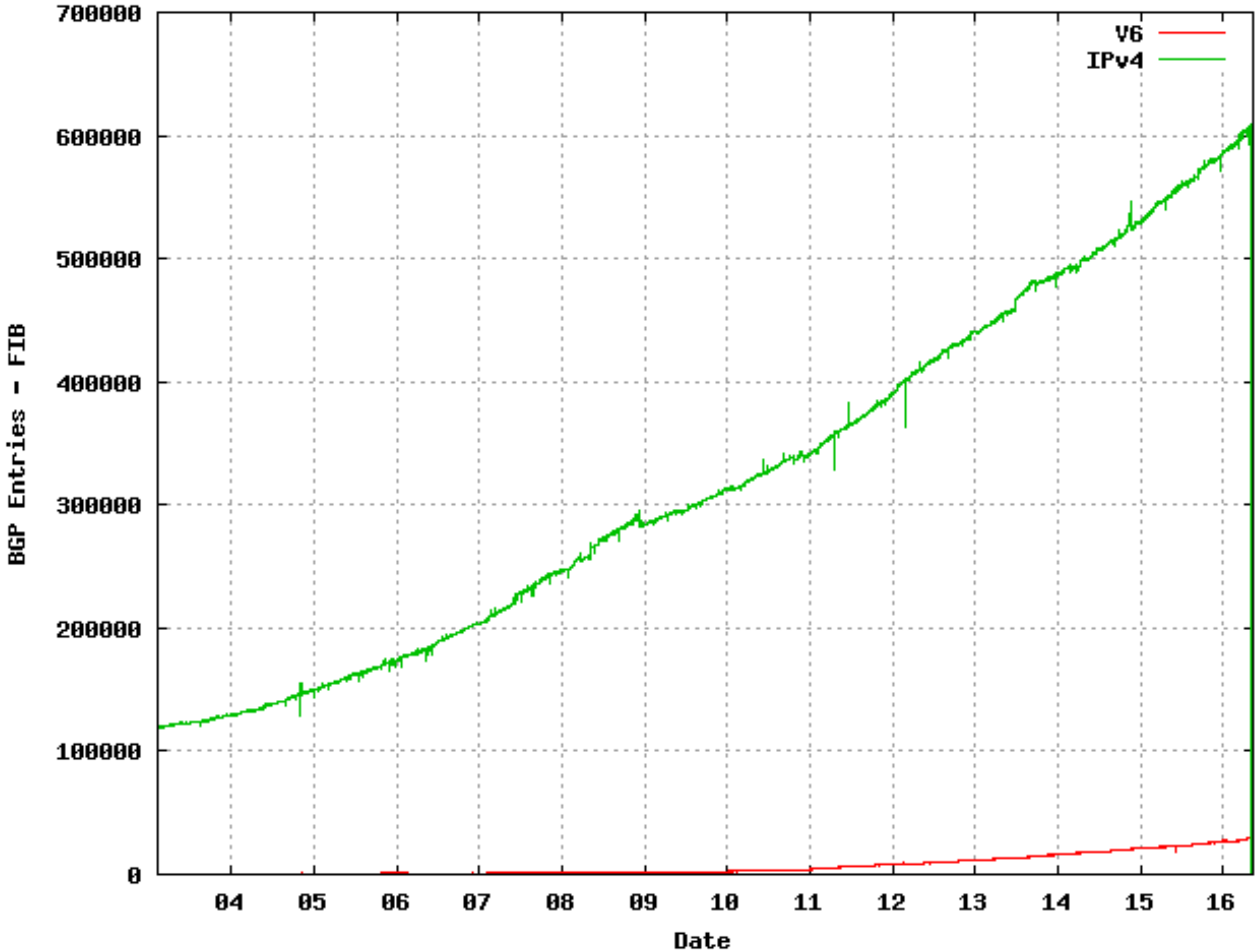
Here is one prediction:



- NAT-traversal induced complexity is increasing rapidly in the IPv4 world
  - New applications desire end-to-end reachability
  - NAT traversal → End of end-to-end
- Current fix: towards a layer-7 network
  - higher costs
  - Barriers for new applications
- Since last year IETF has been making special efforts in rolling out IPv6 deployment

IPv6 Status

http://bgp.potaroo.net/v6/v6rpt.html



ICMP: INTERNET CONTROL MESSAGE PROTOCOL

ICMP: Internet Control Message Protocol

- Used by hosts & routers for feedback, status checking, error reporting
  - unreachable host, network, port, protocol
  - echo request/reply
- ICMP msgs are carried in IP packets
- ICMP message format

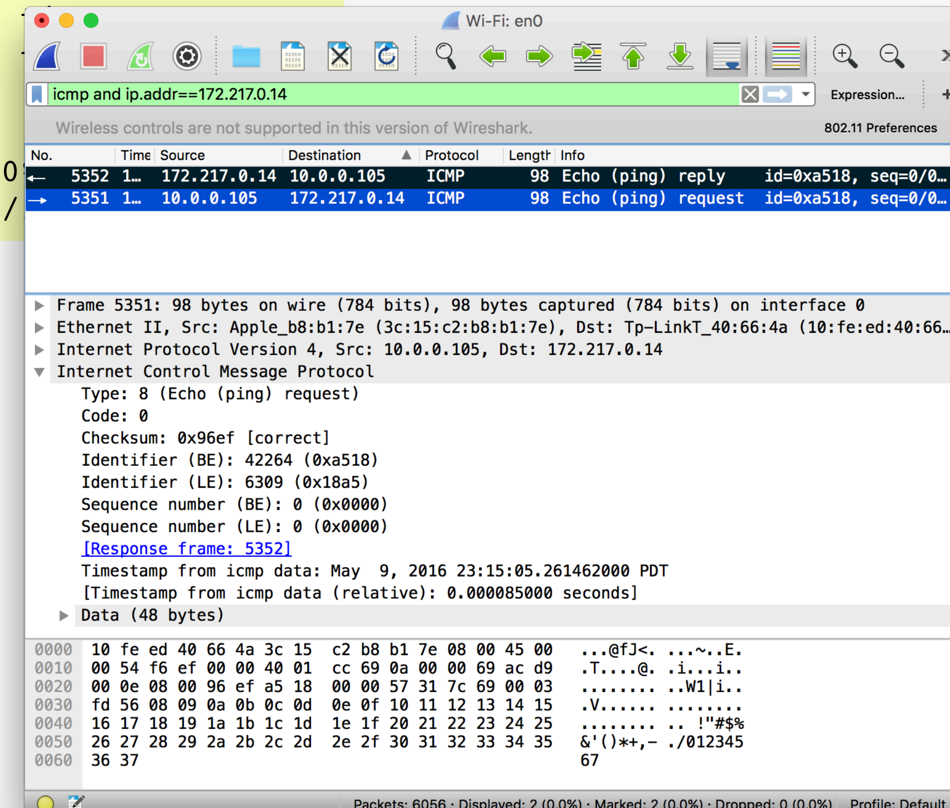
IP header		
type	code	checksum
unused (or used by certain ICMP types)		
IP header and first 64bits of data Or data (according to ICMP types)		

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 (send by router for congest. control)
8	0	echo request (ping)
9	0	route advertisement
10	0	router discovery
11	**0**	**TTL expired**
12	0	bad IP header

Example ICMP usage: ping

- Using ICMP's ECHO_REQUEST message to elicit an ICMP ECHO_RESPONSE

```
✔  23:15 ~ $ ping google.com
PING google.com (172.217.0.14): 56 data bytes
64 bytes from 172.217.0.14: icmp_seq=0 ttl=53 
64 bytes from 172.217.0.14: icmp_seq=1 ttl=53 
^C
--- google.com ping statistics ---
2 packets transmitted, 2 packets received, 0.0
round-trip min/avg/max/stddev = 12.498/12.616/
```



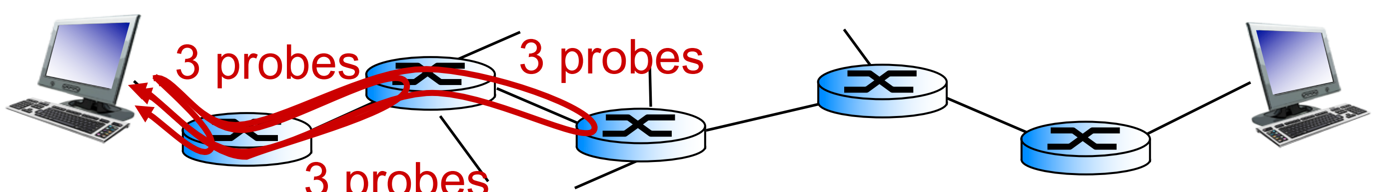
Another Example: Traceroute

- Source sends series of UDP segments to dest.
  - First has TTL =1
  - Second has TTL=2, etc.
  - unlikely port number
- When n^{th} packet arrives to n^{th} router:
  - Router discards packet
  - sends to source an ICMP message (type 11, code 0)
  - Message includes name of router& IP address

- When ICMP message arrives, source calculates RTT
  - Source waits for 5 sec. for ICMP msg before giving up
- Traceroute does this 3 times per hop

Stopping criterion

- UDP packet eventually arrives at destination host
- destination returns ICMP "port unreachable" message (type 3, code 3)
- Source stops

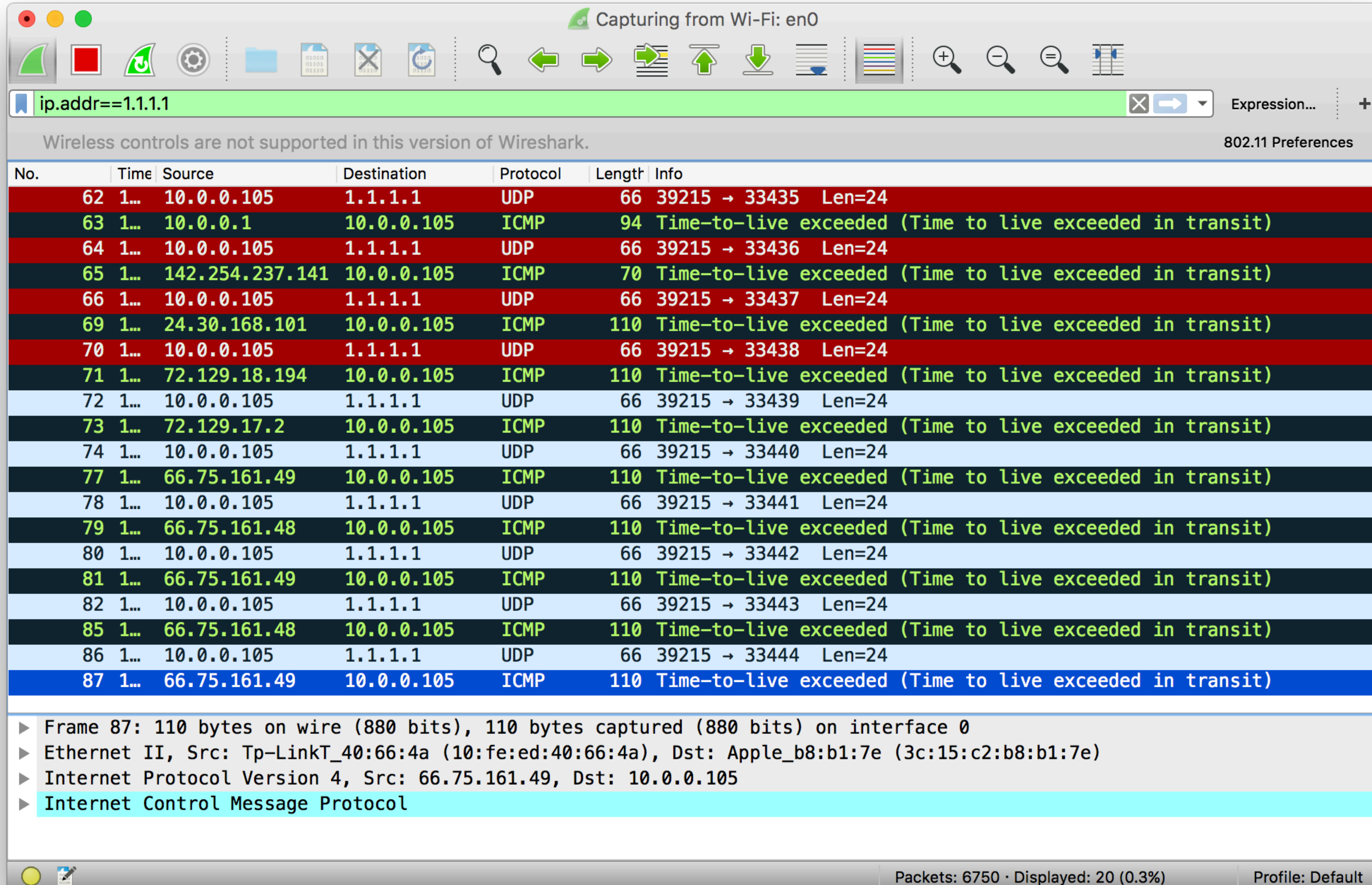


Example of Traceroute

```
✘  23:19 ~ $ traceroute 1.1.1.1
traceroute to 1.1.1.1 (1.1.1.1), 64 hops max, 52 byte packets
 1  10.0.0.1 (10.0.0.1)  1.830 ms  0.938 ms  0.903 ms
 2  142.254.237.141 (142.254.237.141)  8.992 ms  10.832 ms  9.196 ms
 3  agg52.lsaicaev02h.socal.rr.com (24.30.168.101)  10.172 ms  10.038 ms  9.413 ms
 4  agg11.lsaicaev02r.socal.rr.com (72.129.18.194)  10.793 ms  10.876 ms  11.365 ms
 5  agg26.tustcaft01r.socal.rr.com (72.129.17.2)  16.856 ms  14.507 ms  16.039 ms
 6  agg10.lsancarc01r.socal.rr.com (66.75.161.49)  13.222 ms  14.607 ms  12.141 ms
 7  agg10.tustcaft01r.socal.rr.com (66.75.161.48)  22.192 ms  18.932 ms  15.810 ms
 8  agg10.lsancarc01r.socal.rr.com (66.75.161.49)  17.771 ms  18.690 ms  16.683 ms
 9  agg10.tustcaft01r.socal.rr.com (66.75.161.48)  17.803 ms  18.802 ms  19.819 ms
10  agg10.lsancarc01r.socal.rr.com (66.75.161.49)  18.072 ms  18.784 ms  19.713 ms
11  agg10.tustcaft01r.socal.rr.com (66.75.161.48)  22.245 ms  23.096 ms  19.891 ms
12  agg10.lsancarc01r.socal.rr.com (66.75.161.49)  21.794 ms  22.719 ms  24.092 ms
13  agg10.tustcaft01r.socal.rr.com (66.75.161.48)  26.010 ms  23.315 ms  23.835 ms
14  agg10.lsancarc01r.socal.rr.com (66.75.161.49)  25.923 ms  26.916 ms  24.202 ms
15  agg10.tustcaft01r.socal.rr.com (66.75.161.48)  26.905 ms  26.878 ms  28.018 ms
16  agg10.lsancarc01r.socal.rr.com (66.75.161.49)  29.139 ms  30.537 ms  27.753 ms
17  agg10.tustcaft01r.socal.rr.com (66.75.161.48)  31.245 ms  30.677 ms  31.927 ms
18  agg10.lsancarc01r.socal.rr.com (66.75.161.49)  32.904 ms  30.632 ms  32.194 ms
19  agg10.tustcaft01r.socal.rr.com (66.75.161.48)  34.848 ms  34.874 ms  31.802 ms
20  agg10.lsancarc01r.socal.rr.com (66.75.161.49)  33.044 ms  35.003 ms  35.773 ms
21  agg10.tustcaft01r.socal.rr.com (66.75.161.48)  35.060 ms  38.687 ms  44.133 ms
22  agg10.lsancarc01r.socal.rr.com (66.75.161.49)  36.898 ms  38.519 ms  39.894 ms
23  agg10.tustcaft01r.socal.rr.com (66.75.161.48)  38.569 ms  39.370 ms  39.827 ms
24  agg10.lsancarc01r.socal.rr.com (66.75.161.49)  41.021 ms  38.862 ms  48.181 ms
25  agg10.tustcaft01r.socal.rr.com (66.75.161.48)  42.752 ms  43.131 ms  40.068 ms
26  agg10.lsancarc01r.socal.rr.com (66.75.161.49)  45.036 ms  42.723 ms  44.211 ms
27  agg10.tustcaft01r.socal.rr.com (66.75.161.48)  47.678 ms  46.014 ms  43.966 ms
....
```

What's a limit here?

Behind the Scenes of Traceroute



ROUTING ALGORITHMS

Where we are in the book

4.5 Routing algorithms (this/next lecture)

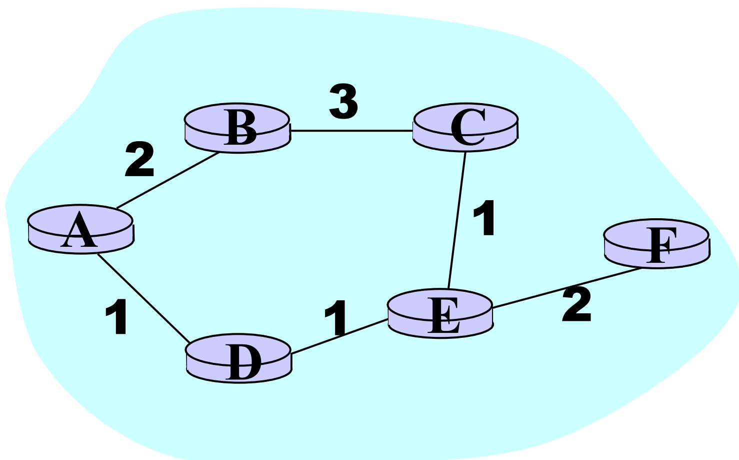
- Link state/Dijkstra algorithm
- Distance Vector/Bellman-Ford algorithm
- Hierarchical routing

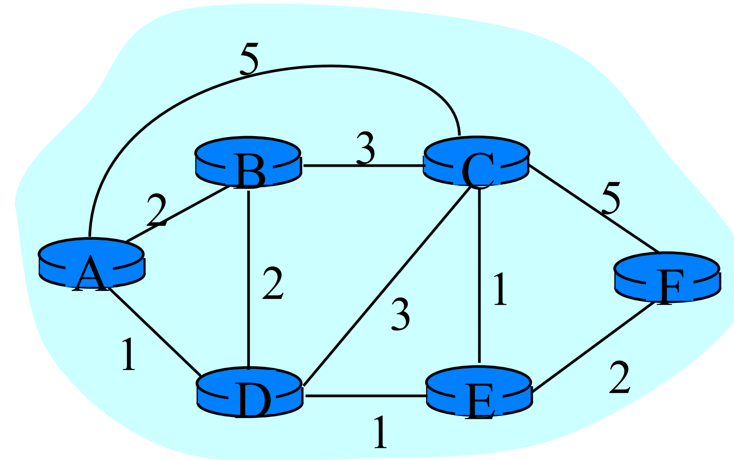
4.6 Routing in the Internet (next lecture)

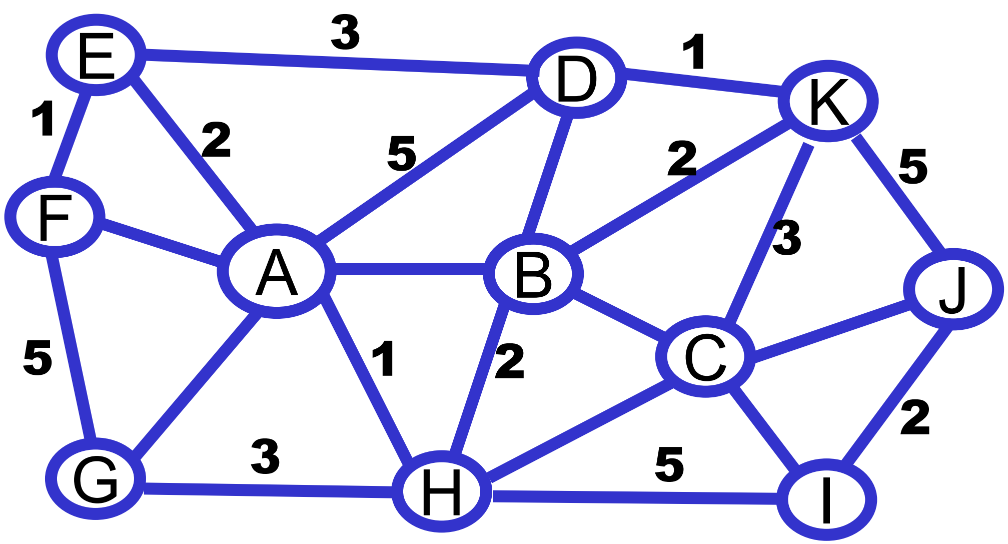
- RIP: Routing Information Protocol
- OSPF: Open Shortest Path First Protocol
- BGP: Border Gateway Protocol

4.7 Broadcast and multicast routing (next Tuesday)

How to find the best path to a destination?

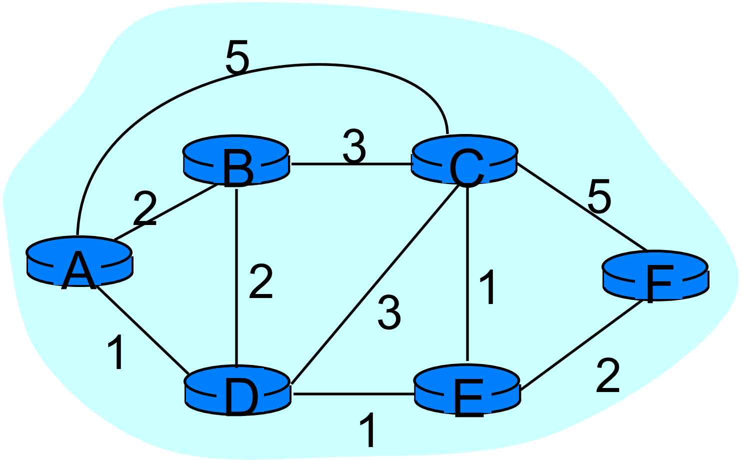






Need computation algorithm to find the best path from one point to any other point

Network Graph Abstraction



Graph: G = (N,E)

N = set of routers = { A, B, C, D, E, F }

E = set of links ={ (A,B), (A,C), (A,D), (B,C), (B,D), (C,D), (C,E), (C,F), (E,F) }

Cost of link (a, b) = C(a, b)

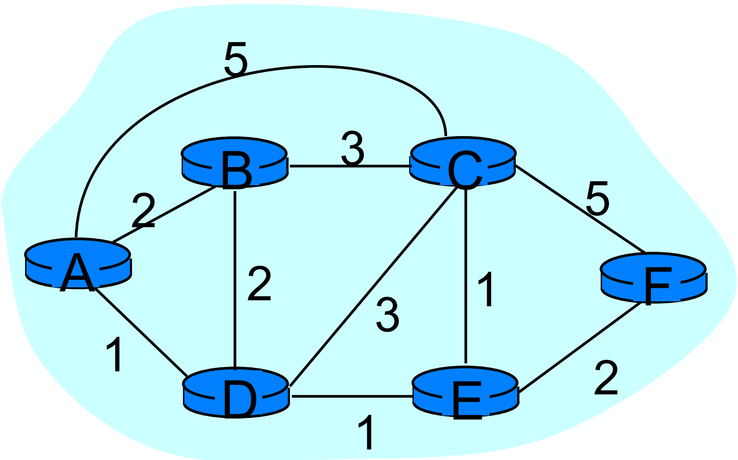
– e.g. C(A, B) = 2

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

https://en.wikipedia.org/wiki/Shortest_path_problem

Routing algorithm: given a graph, find least-cost path from a given node to all the other nodes in the graph

Network Routing: algorithms vs. protocols



Route computation algorithms:
given a complete topology graph with all link costs

link-state (Dijkstra**):**

- each router computes shortest paths to all destinations

distance-vector (Bellman-Ford**):**

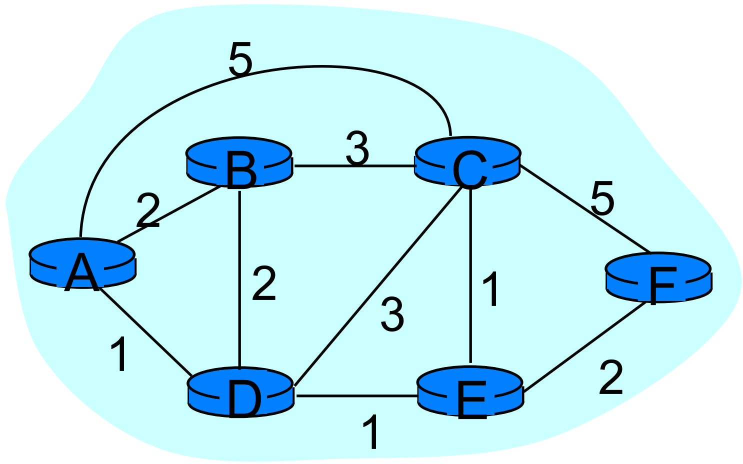
- each router computes its shortest paths based on the shortest paths of all its neighbors

Routing protocols

- define the format of routing information exchanges
- define the computation upon receiving routing updates
- network topology changes over time → routing protocol must continuously update the routers with latest changes

Link-State algorithm: basic operations

- every node knows the network topology graph with link cost
- Each node computes the “shortest” (the least cost) paths from itself to all other nodes
  - Figure out the next hop of the best path
- iterative: after *k* iterations, a node know the best paths to *k* destinations

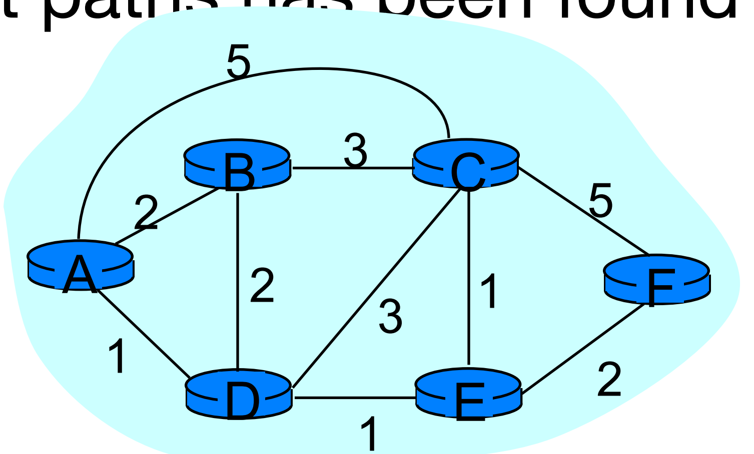


Link-State algorithm: basic notations

- c(x,y): link cost from neighbor node x to node y
- D(v): current value of cost of path from source to destination v
  - Initializes to ∞ if (x, y) are not direct neighbors

    from A's view: D(e) = D(f) = ∞
- p(v): the node right before v along best path from source to v

    from A's view, best path from A to C is A-D-E-C, p(c)=E
- N': set of nodes whose least-cost paths has been found

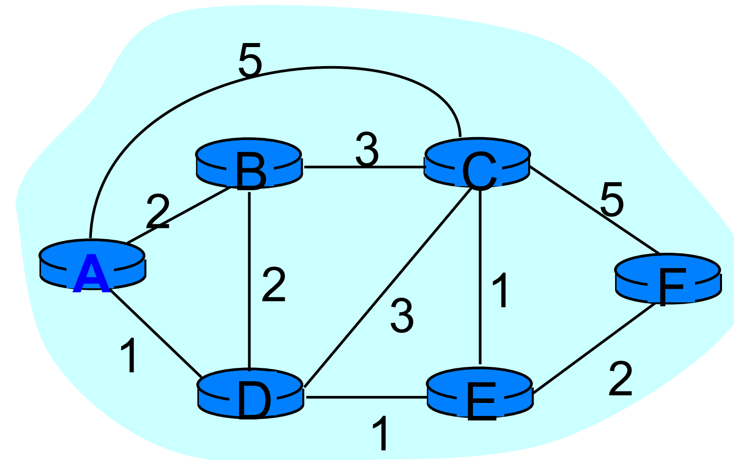


Link-State algorithm

(consider the computation done at node **A**)

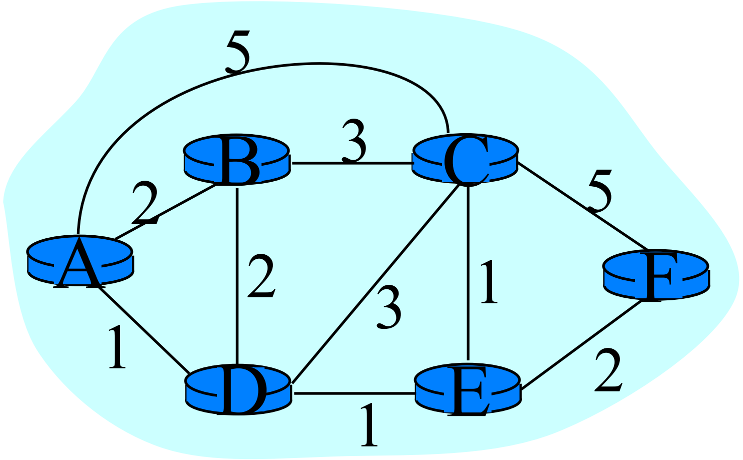
```
Initialization:
 N' = {A}
 for all nodes v
   if v adjacent to A
     then D(v) = c(A,v)
     else D(v) = ∞

Loop
  find w not in N' such that D(w) is minimum:
    add w to N'
    update D(v) for all v adjacent to w and not in N':
       D(v) = min( D(v), D(w) + c(w,v) ), p(v) = w
 /* new cost to v is either the old cost, or the
    shortest path cost to w plus the cost from w to v */
 until all nodes in N'
```



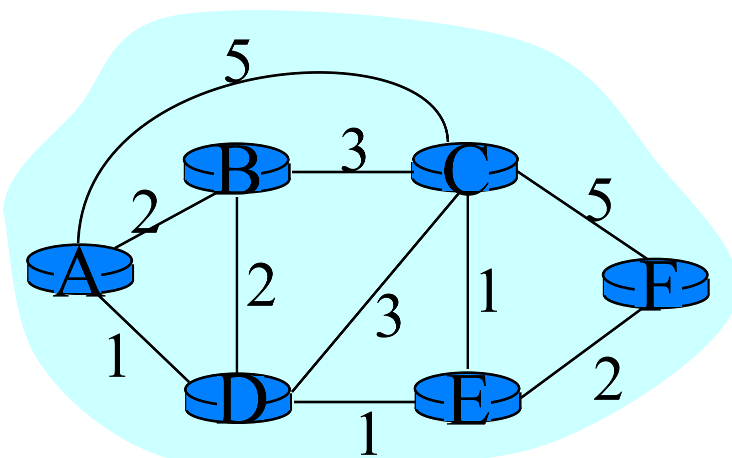
Link-State algorithm: example

Step	start N'	D(B),p(B)	D(C),p(C)	D(D),p(D)	D(E),p(E)	D(F),p(F)
0	A	2, A	5, A	1, A	∞	∞



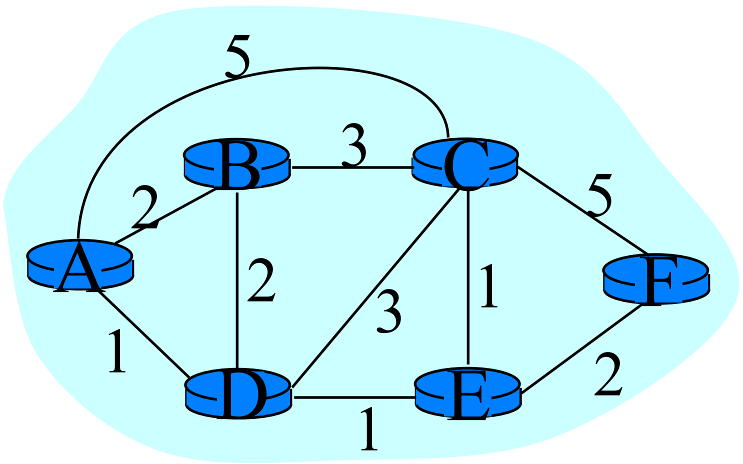
Link-State algorithm: example

Step	start N'	D(B),p(B)	D(C),p(C)	D(D),p(D)	D(E),p(E)	D(F),p(F)
0	A	2, A	5, A	1, A	∞	∞
1	AD	2, A	4, D		2, D	



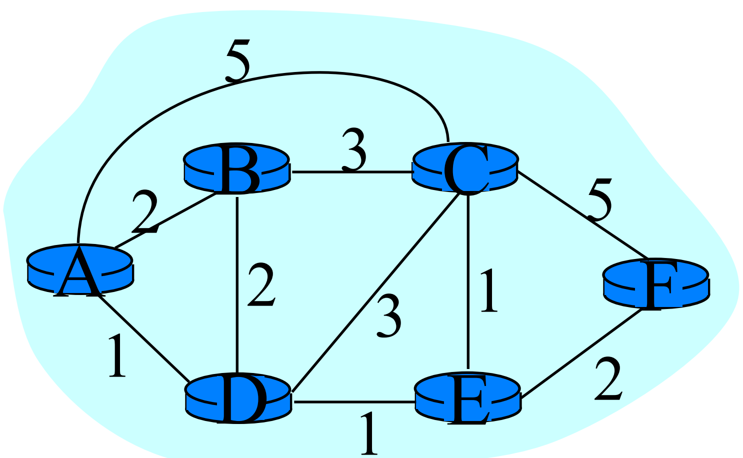
Link-State algorithm: example

Step	start N'	D(B),p(B)	D(C),p(C)	D(D),p(D)	D(E),p(E)	D(F),p(F)
0	A	2, A	5, A	1,A	∞	∞
1	AD	2, A	4, D		2, D	
2	ADE		3, E			4, E



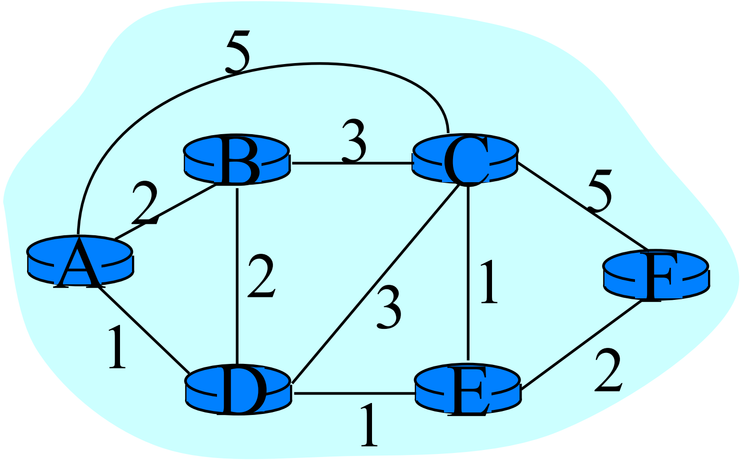
Link-State algorithm: example

Step	start N'	D(B),p(B)	D(C),p(C)	D(D),p(D)	D(E),p(E)	D(F),p(F)
0	A	2, A	5, A	1, A	∞	∞
1	AD	2, A	4, D		2, D	
2	ADE		3, E			4, E
3	ADEB		3, E			



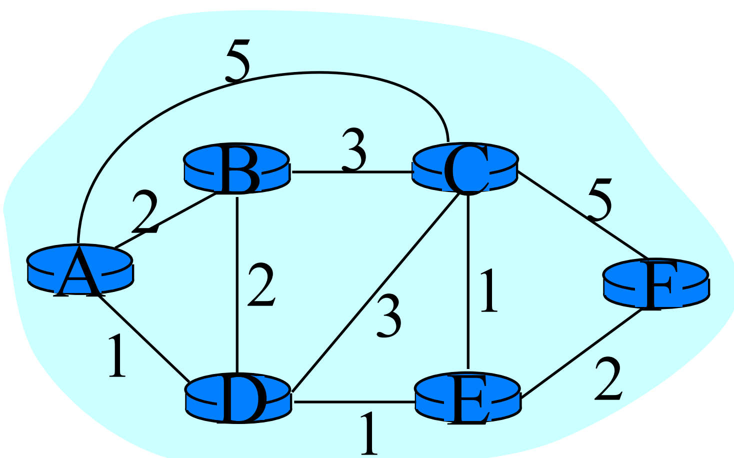
Link-State algorithm: example

Step	start N'	D(B),p(B)	D(C),p(C)	D(D),p(D)	D(E),p(E)	D(F),p(F)
0	A	2, A	5, A	1, A	∞	∞
1	AD	2, A	4, D		2, D	
2	ADE		3, E			4, E
3	ADEB		3, E			
4	ADEBC					4, E



Link-State algorithm: example

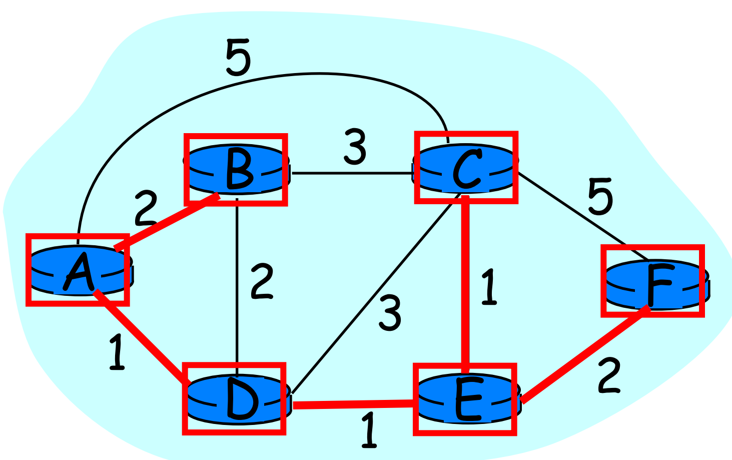
Step	start N'	D(B),p(B)	D(C),p(C)	D(D),p(D)	D(E),p(E)	D(F),p(F)
0	A	2, A	5, A	1, A	∞	∞
1	AD	2, A	4, D		2, D	
2	ADE		3, E			4, E
3	ADEB		3, E			
4	ADEBC					4, E
5	ADEBCF					



Link-State algorithm: example

Step	start N'	D(B),p(B)	D(C),p(C)	D(D),p(D)	D(E),p(E)	D(F),p(F)
0	A	2,A	5,A	1,A	∞	∞
1	AD	2,A	4,D		2,D	
2	ADE	2,A	3,E			4,E
3	ADEB		3,E			4,E
4	ADEBC					4,E
5	ADEBCF					

Resulting shortest-path tree for A:



Resulting forwarding table at A:

destination	Link/interface
B	(A, B)
D	(A, D)
E	(A, D)
C	(A, D)
F	(A, D)

Link-State algorithm: discussion

Algorithm complexity: for a graph with n nodes

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

Oscillations possible:

- e.g., link cost = amount of carried traffic

Distance Vector Algorithm

Recall Link-State algorithm:

- Each node knows the complete topology graph with link costs
- Each node calculate the shortest path to all other nodes

For Distance-Vector algorithm:

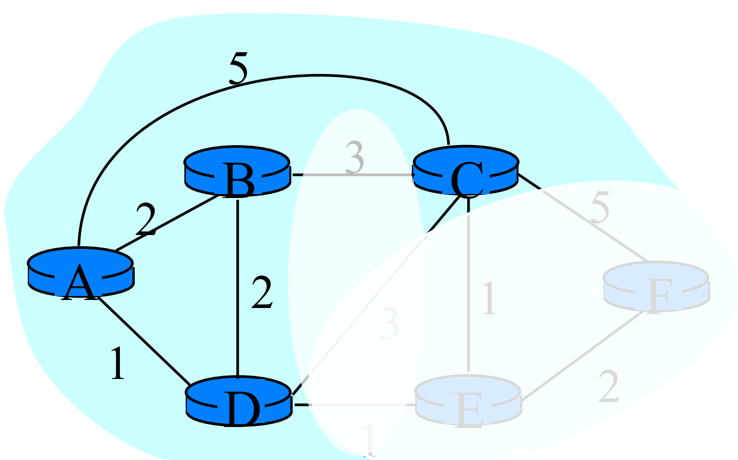
- each node know *only* needs from each direct neighbor its list of distances to all destinations
- Each node computes shortest path based on the input from all its neighbors

Distance Vector Equation

Define: $D_x(y)$:= cost of best path from x to y

Then $\boxed{D_x(y) = \min \{c(x,v) + D_v(y)\}}$

- where min is taken over *all* neighbors v of x



$$D_A(F) = \min \{c(A,B) + D_B(F),$$
$$c(A,D) + D_D(F),$$
$$c(A,C) + D_C(F)\}$$
$$= \min \{2 + 5,$$
$$1 + 3,$$
$$5 + 3\} = 4$$

Node leading to shortest path is next hop ➜ forwarding table

Distance Vector: what a node does

- Node *x* knows link cost to neighbor v: $c(x,v)$
- Node *x* maintains $\mathbf{D}_x = [D_x(y): y \in N]$
  - $D_x(y)$ = estimate of least cost from x to y
- Node *x* sends the distance vector, $\mathbf{D}_x$, to all its neighbors
- Node x receives $\mathbf{D}_v$ from each neighbor v, then calculate $D'_x(y) = \min \{c(x,v) + D_v(y)\}$

  If $D'_x(y) < D_x(y)$:
  - $D_x(y) = D'_x(y)$
  - next hop to y = v
  - Send out the updated $\mathbf{D}_x$

Distance Vector Protocol

Iterative, asynchronous:

- each local iteration caused by:
  - local link cost change
  - DV update message from neighbor
- continues until no nodes exchange info.

Distributed:

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

asynchronous: nodes need *not* exchange info/iterate in lock step

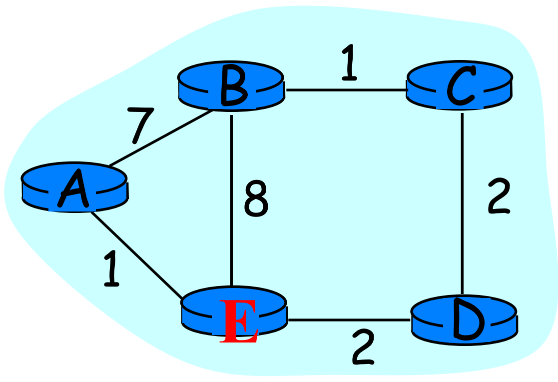
Each node:

wait for (change in local link cost of msg from neighbor)

recompute estimates

if DV to any dest has changed, *notify* neighbors

Distance Table: example



cost to destination via

$D^E()$	A	B	D
A	1	14	5

destination

$$D^E(A,B) = c(E,B) + \min_w\{D^B(A,w)\}$$
$$= 8+6 = 14$$

$$D^E(A,D) = c(E,D) + \min_w\{D^D(A,w)\}$$
$$= 2+3 = 5$$

$$D^E(C,D) = c(E,D) + \min_w\{D^D(C,w)\}$$
$$= 2+2 = 4$$

Row: for each possible destination
Column: for each directly-attached neighbor node

Routing table produces forwarding table

cost to destination via

$D^E()$	A	B	D
A	1	14	5
B	7	8	5
C	6	9	4
D	4	11	2

destination

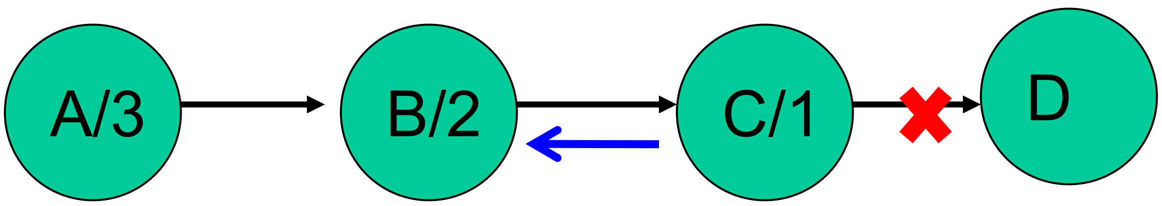
E's routing table → E's forwarding table

	Next hop
A	A
B	D
C	D
D	D

destination

Count-To-Infinity Problem

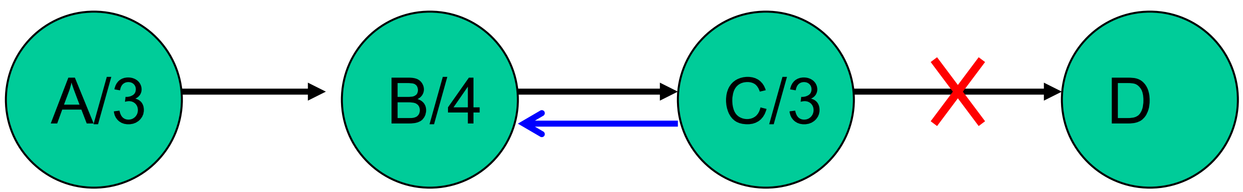
- Assume we use hop count as metric
  - A uses B to reach D with cost 3
  - B uses C to reach D with cost 2
  - C reaches D with cost 1



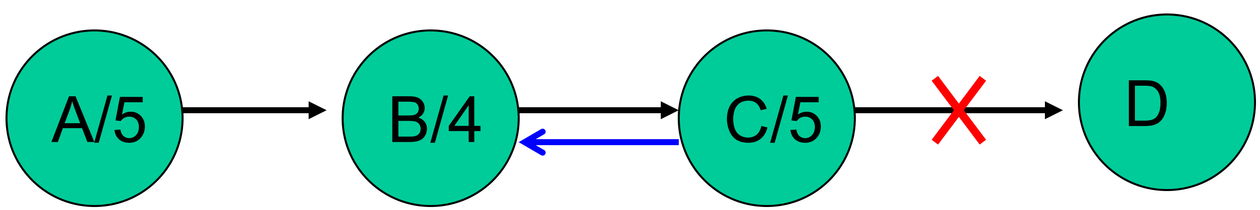
- Suppose link between C and D breaks
  - C switches to B, increase its cost to B's + 1 = 3

Count-To-Infinity Problem (cont.)

- B’s path cost is now 4
  - A has not realized what has happened yet



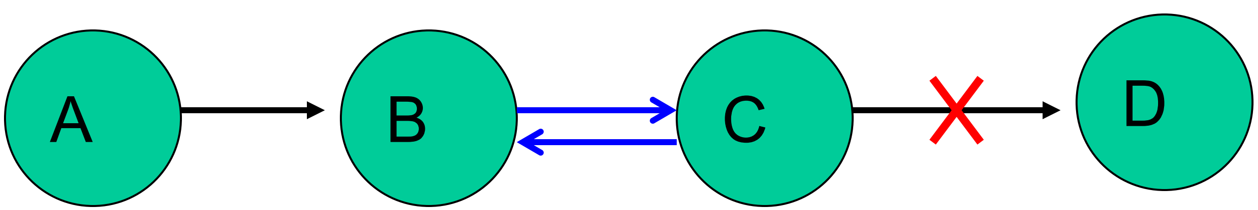
- Then, A’s and C’s cost are now 5



- B’s path cost is changed to 6
  - Cycle repeats while “counting to infinity”

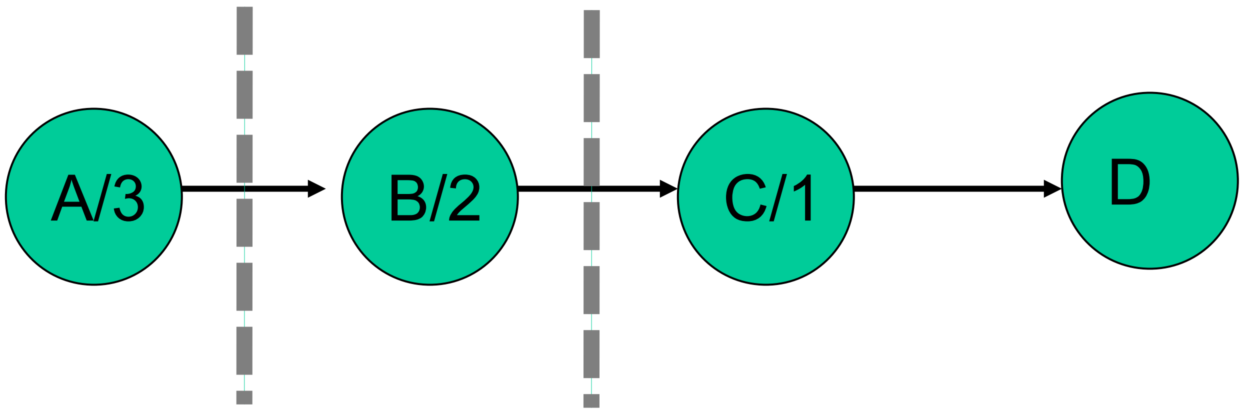
Routing Loops

- In this cases, the packets with destination of D at router A:
  - Go to router B
  - Then go to router C
  - Then go back to router B



Split Horizon

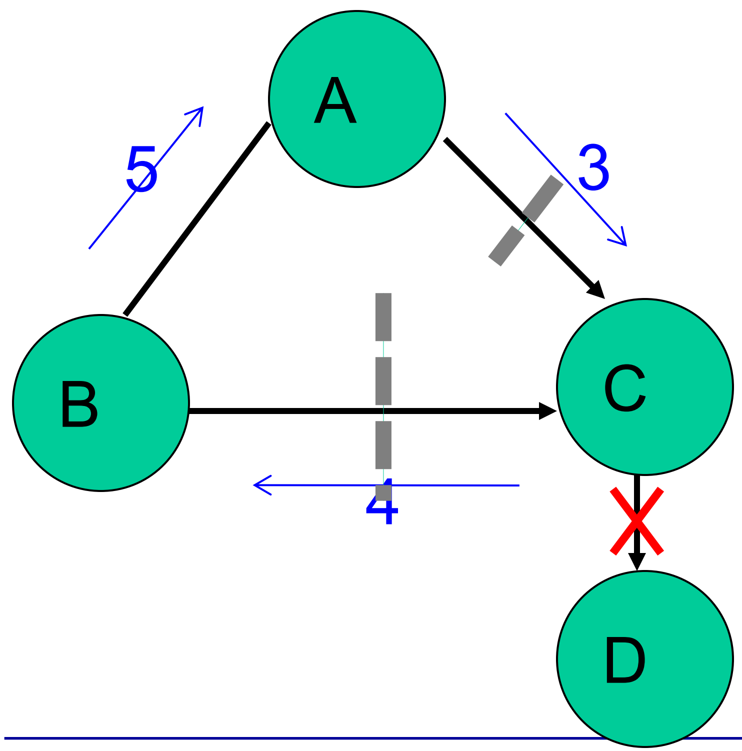
- In Split Horizon: B does not tell C that B can reach D
  - So C does not know that B can reach D



- Once C-D link breaks: C would not switch to go through B to reach D

Split Horizon --- Might Not Work

- Split Horizon doen't eliminate loops in all cases
- Suppose the link between C and D breaks

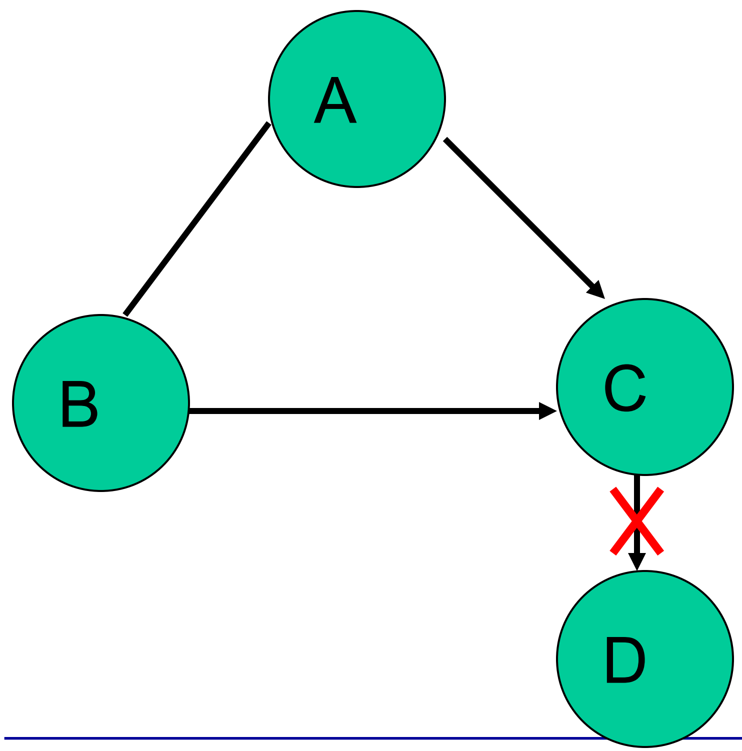


1. A and B do not tell C their distance to D
2. After C–D failure, A learns that B can reach D, so sends new route to C
3. C sends route learned from A to B
4. B sends route learned from C to A
5. A sends route learned from B to C

Routing loop still exists

Split Horizon with poison reverse

- If B goes through C to reach D :
  - B tells C that its (B's) distance to D is *infinite* (so C never attempts to reach D via B)



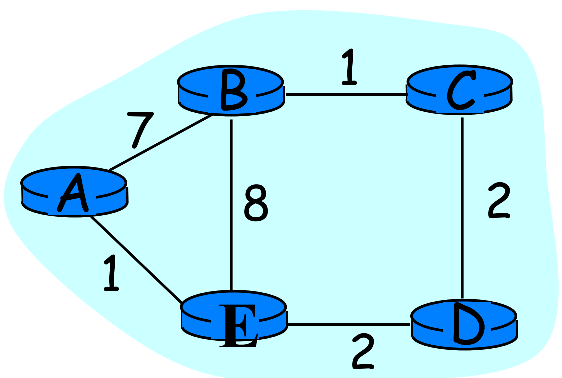
1. A and B tell C that their distance to D is infinite
2. When link C-D fails, C realized that it lost reachability to D
3. C sends to A and B: D_D = infinite

Comparison of LS and DV algorithms

- Performance measure: Message overhead, time to convergence
- distance vector:
  - distribute to neighbors the distances to all destinations
    - Each update msg can be large in size, but travels over one link
  - each node only knows *distances* to other destinations
- link state
  - Broadcast to entire net one's distance to all neighbors
    - Each update msg is small in size, but travels over all links in the network
  - each node knows entire topology

what happens if a router malfunctions?

- Link-state
  - A node can advertise incorrect *link* cost
  - each node computes its *own* table
- Distance vector
  - A node can advertise incorrect *path* cost
  - one node's distance-list is used by its neighbors for their own routing selection



Node-D: "I have 0 cost to all other nodes"

Link-State:

- updates from A & B: not connected to D
- Updates from C & E: cost not 0

Distance-Vector:

- other nodes do not have info to verify