CS 43: Computer Networks

20: NAT, ICMP, Routing

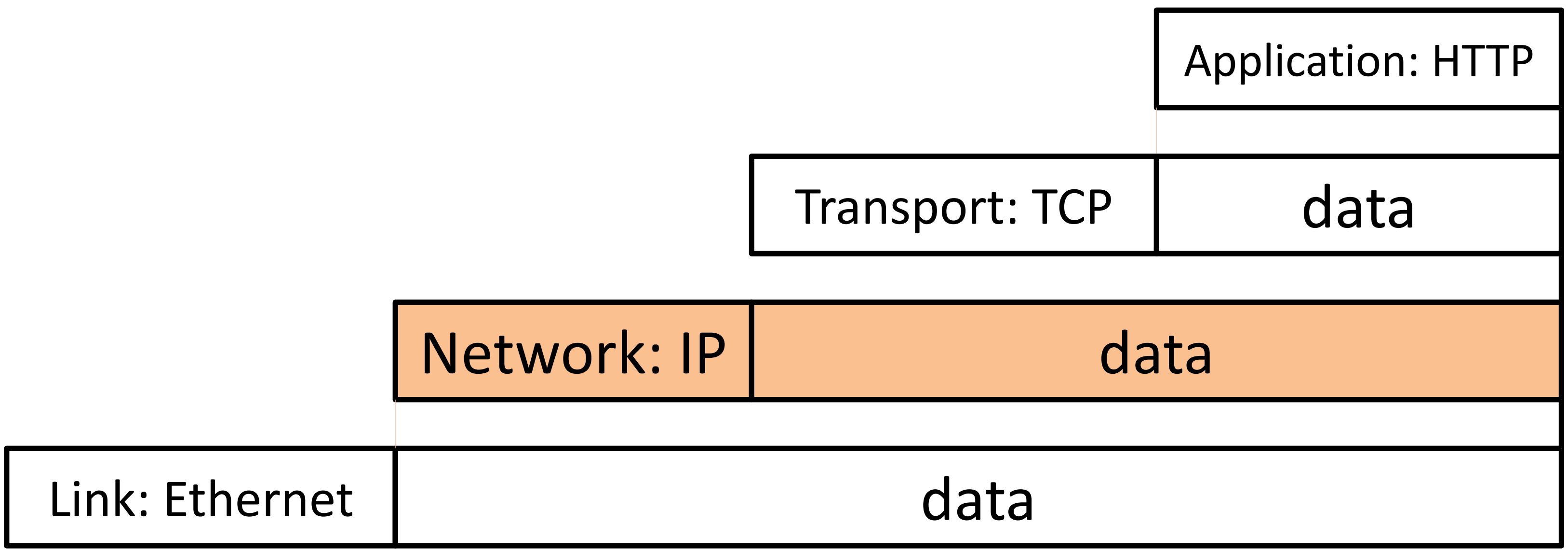
November 19, 2019

SWARTHMORE COLLEGE

Reading Quiz

Network Layer

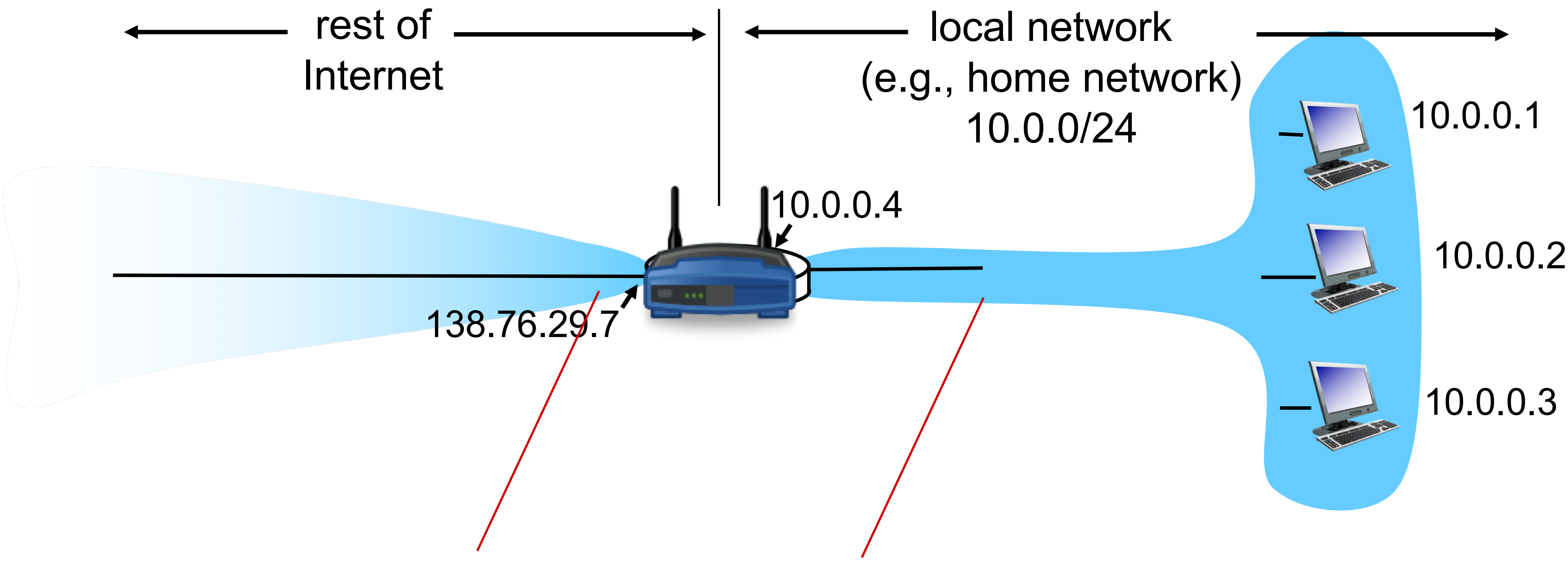
- Function: Route packets end-to-end on a network, through multiple hops



Network Layer Functions

- Forwarding: move packets from router's input to appropriate router output
  - Look up in a table
- Routing: determine route taken by packets from source to destination.
  - Populating the table

NAT: Network Address Translation



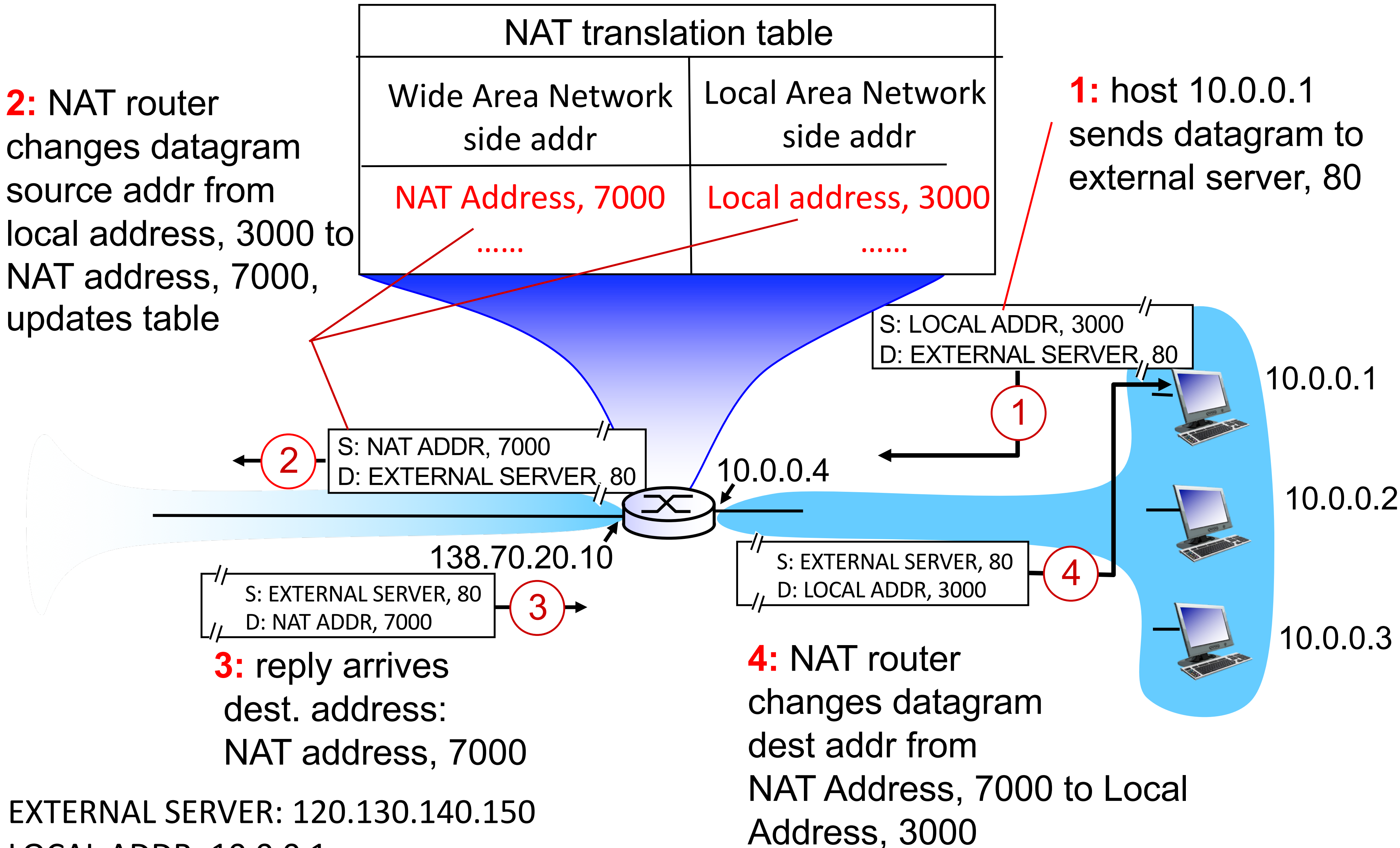
all datagrams leaving local network have same single source NAT IP address: 138.76.29.7,different source port numbers

datagrams with source or destination in this network have 10.0.0/24 address for source, destination (as usual)

Implementing NAT

- Two hosts communicate with same destination
  - Destination needs to differentiate the two
- Map outgoing packets
  - Change source address and source port
- Maintain a translation table
  - Map of (src addr, port #) to (NAT addr, new port #)
- Map incoming packets
  - Map the destination address/port to the local host

NAT: network address translation



EXTERNAL SERVER: 120.130.140.150
LOCAL ADDR: 10.0.0.1
NAT ADDR: 138.70.20.10

How do we feel about NAT?

A. NAT is great! It conserves IP addresses and makes it harder to reach non-public machines.

B. NAT is mostly good, but has a few negative features. No big deal.

C. NAT is mostly bad, but in some cases, it’s a necessary evil.

D. NAT is an abomination that violates the end to end principle, and we should not use it!

Principled Objections Against NAT

- Routers are not supposed to look at port #s
  - Network layer should care only about *IP* header
  - … and not be looking at the *port numbers* at all
- NAT violates the end-to-end argument
  - Network nodes should not modify the packets
- IPv6 is a cleaner solution
  - Better to migrate than to limp along with a hack

That's what happens when network puts power in hands of end users!

IPv6

- Initial motivation: 32-bit address space soon to be completely allocated, any day now™.

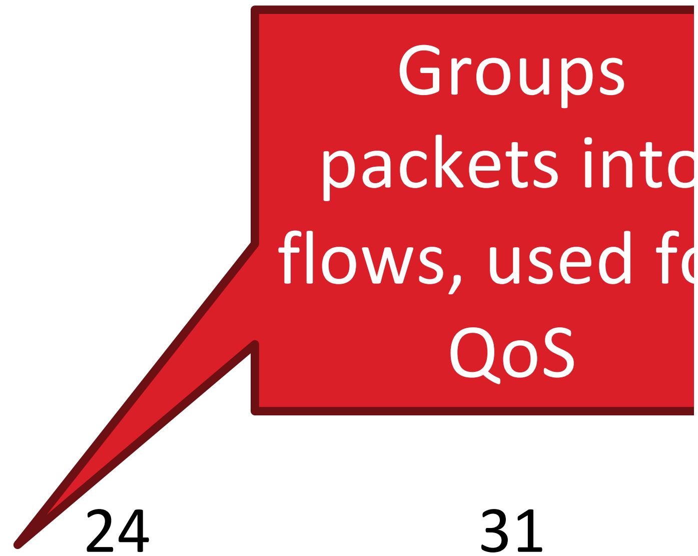
- Additional motivation:
  - header format helps speed processing/forwarding
  - header changes to facilitate QoS

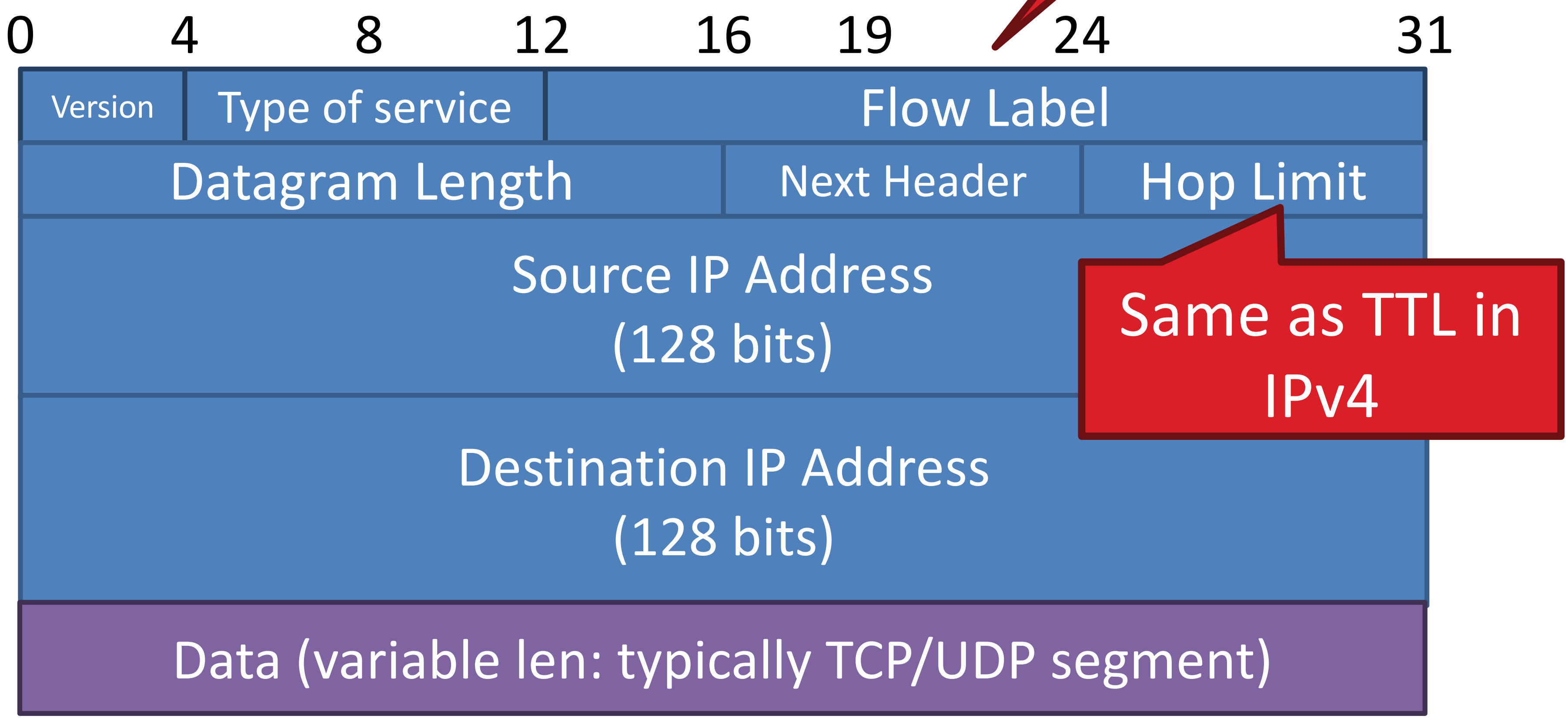
IPv6 datagram format:

- fixed-length 40 byte header
- no fragmentation allowed

IPv6 Header

- Double the size of IPv4 (320 bits vs. 160 bits)





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

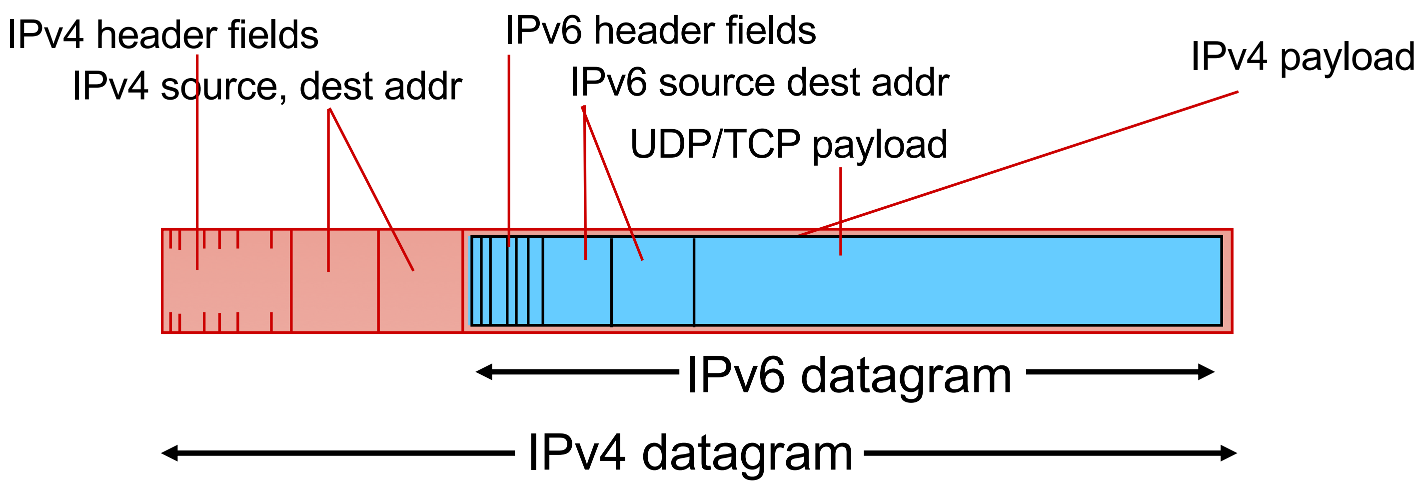
Transitioning to IPv6

- Option 1: “Flag day”
  - How do we get *everyone* on the Internet to agree?
  - Whose authority to decide when?
  - Can you imagine how much would break?

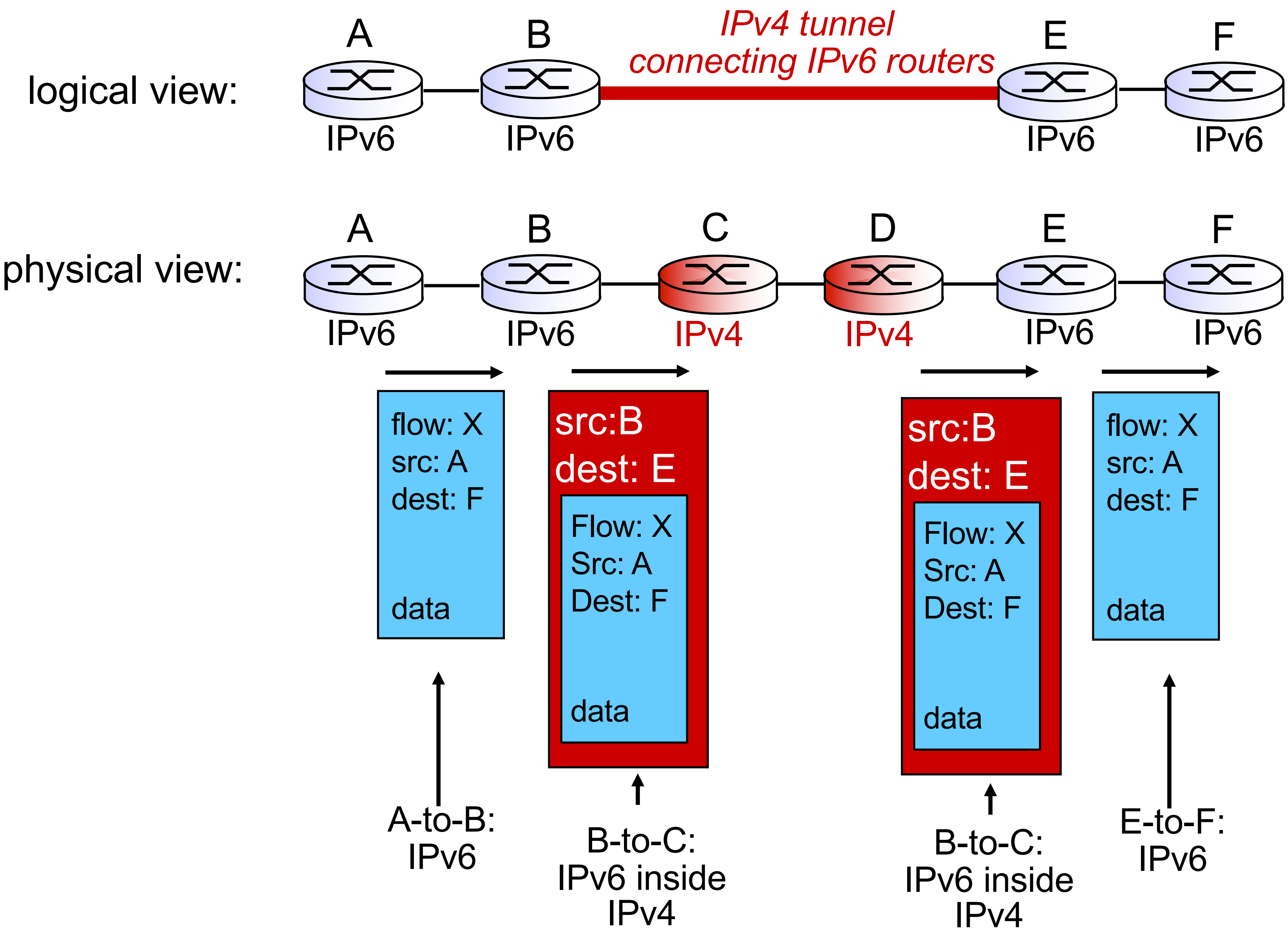
- Option 2: Slow transition
  - Some hosts/routers speak both versions
  - Must have some way to deal with those who don’t
  - Lack of incentive to switch

Tunneling

- IPv6 datagram carried as *payload* in IPv4 datagram among IPv4 routers



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

ICMP: Internet Control Message Protocol

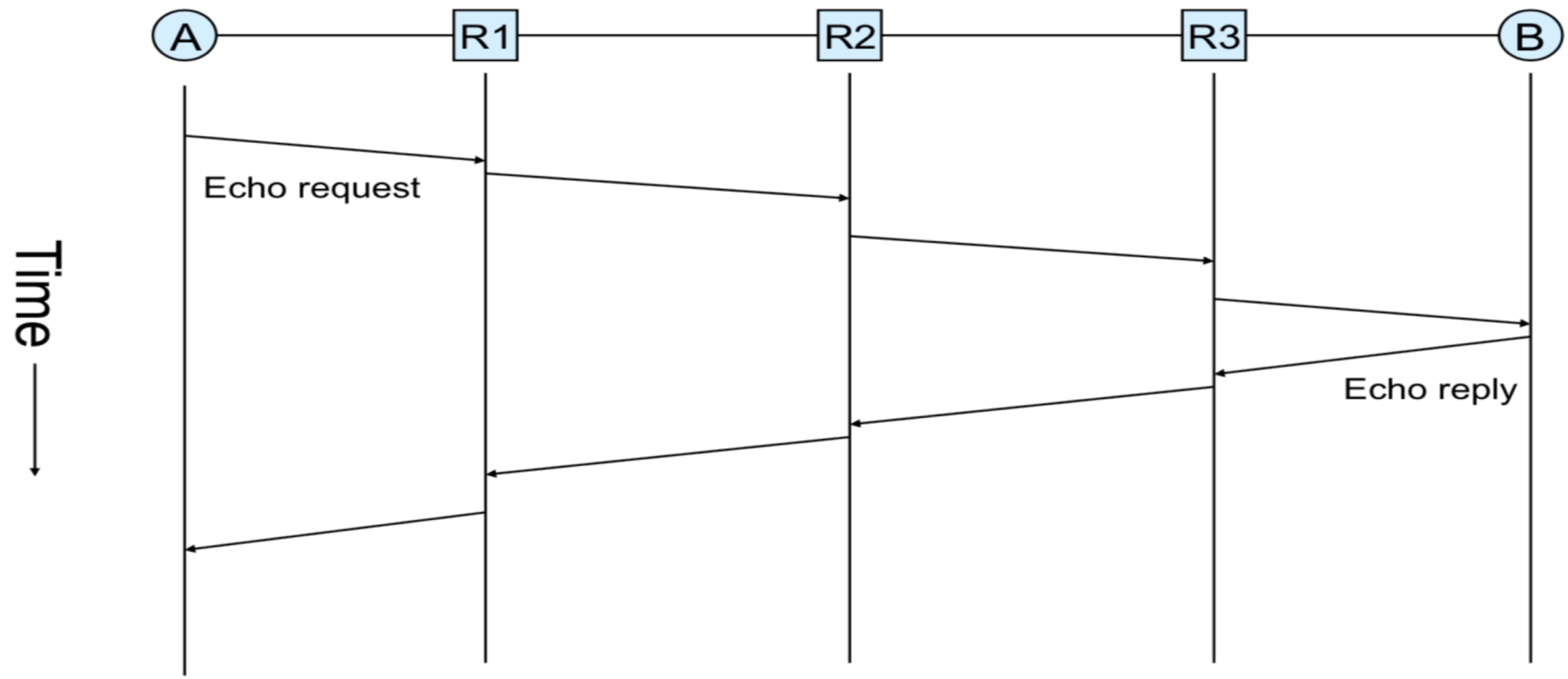
- Used to communicate network information
  - "Control messages", i.e., not data themselves
  - Error reporting
    - Unreachable host
    - Unreachable network
    - Unreachable port
    - TTL expired
  - Test connectivity
    - Echo request/response (ping)

ICMP: Internet Control Message Protocol

- Header:
  - 1-byte type
  - 1-byte code
  - 2-byte checksum
  - 4 bytes vary by type
- Sits above IP
  - Type 1 in IP header
  - Usually considered part of IP

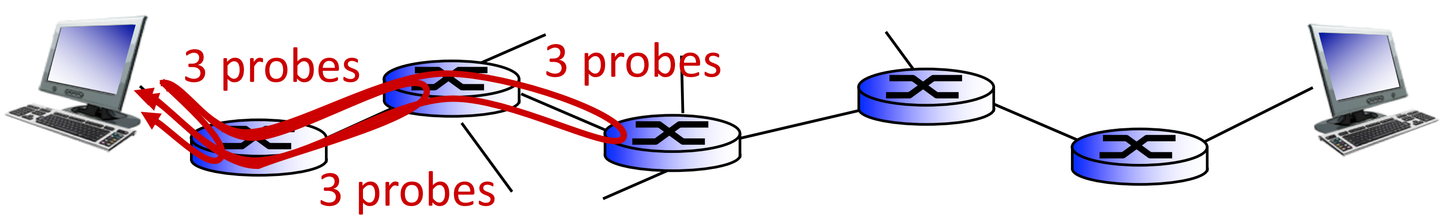
Type	Code	Description
0	0	echo reply (ping)
3	0	dest. network unreachable
3	1	dest host unreachable
3	2	dest protocol unreachable
3	3	dest port unreachable
3	6	dest network unknown
3	7	dest host unknown
4	0	source quench (congestion control - not used)
8	0	echo request (ping)
9	0	route advertisement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

Ping



Traceroute and ICMP

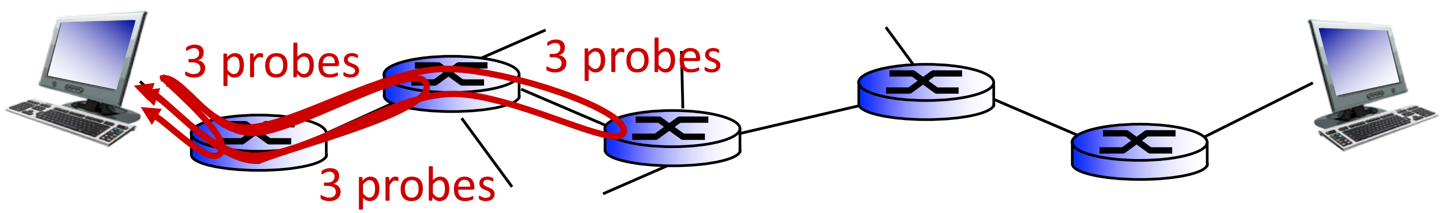
- Source sends sets of UDP segments (usually 3) to dest
  - first set has TTL =1
  - second set has TTL=2, etc.
  - unlikely port number
- When *n*th set of datagrams arrives to nth router:
  - router discards datagrams
  - and sends source ICMP messages (type 11, code 0)
  - ICMP messages includes name of router & IP address
- When ICMP messages arrives, source records RTTs



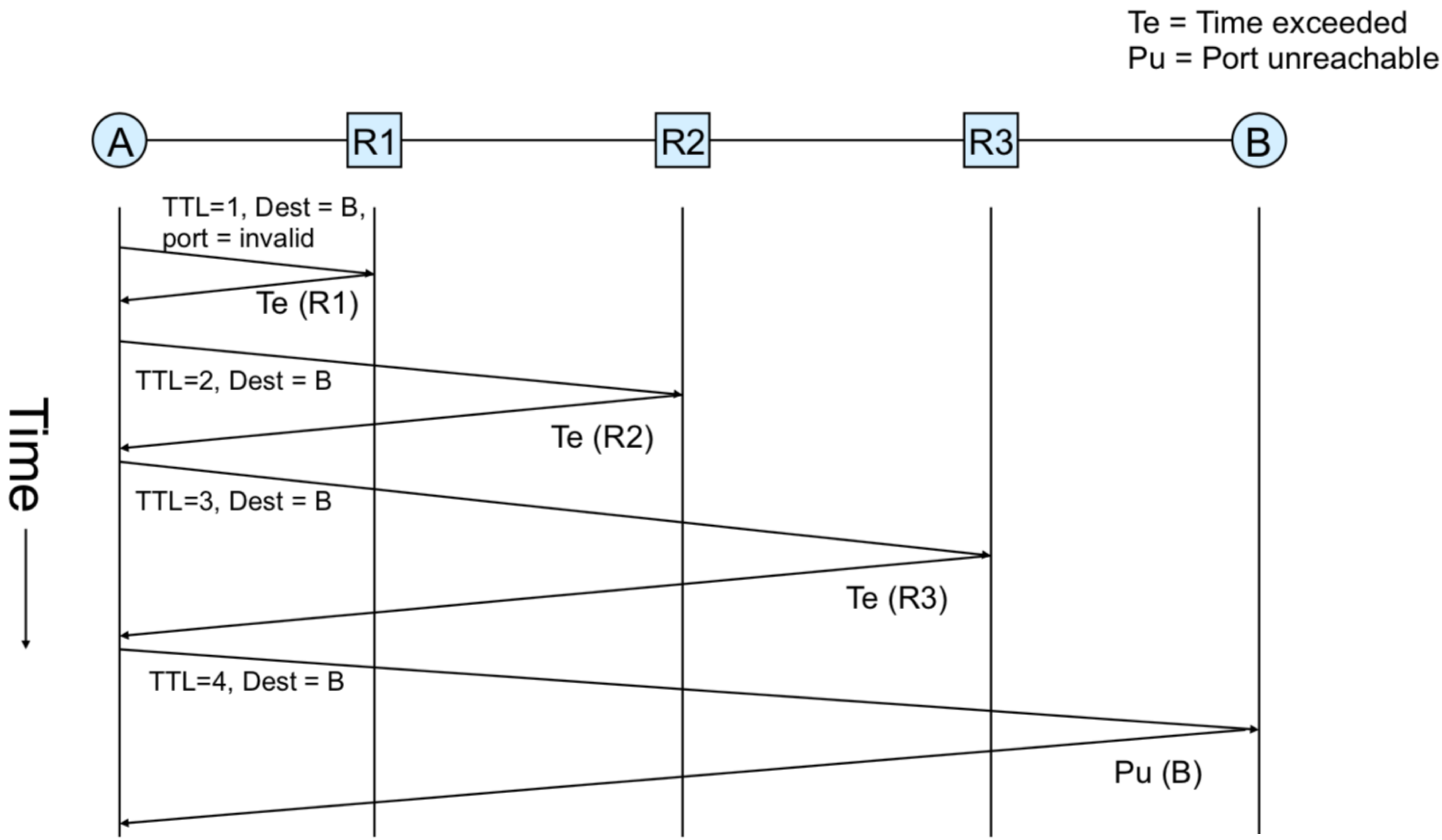
Traceroute and ICMP

stopping criteria:

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

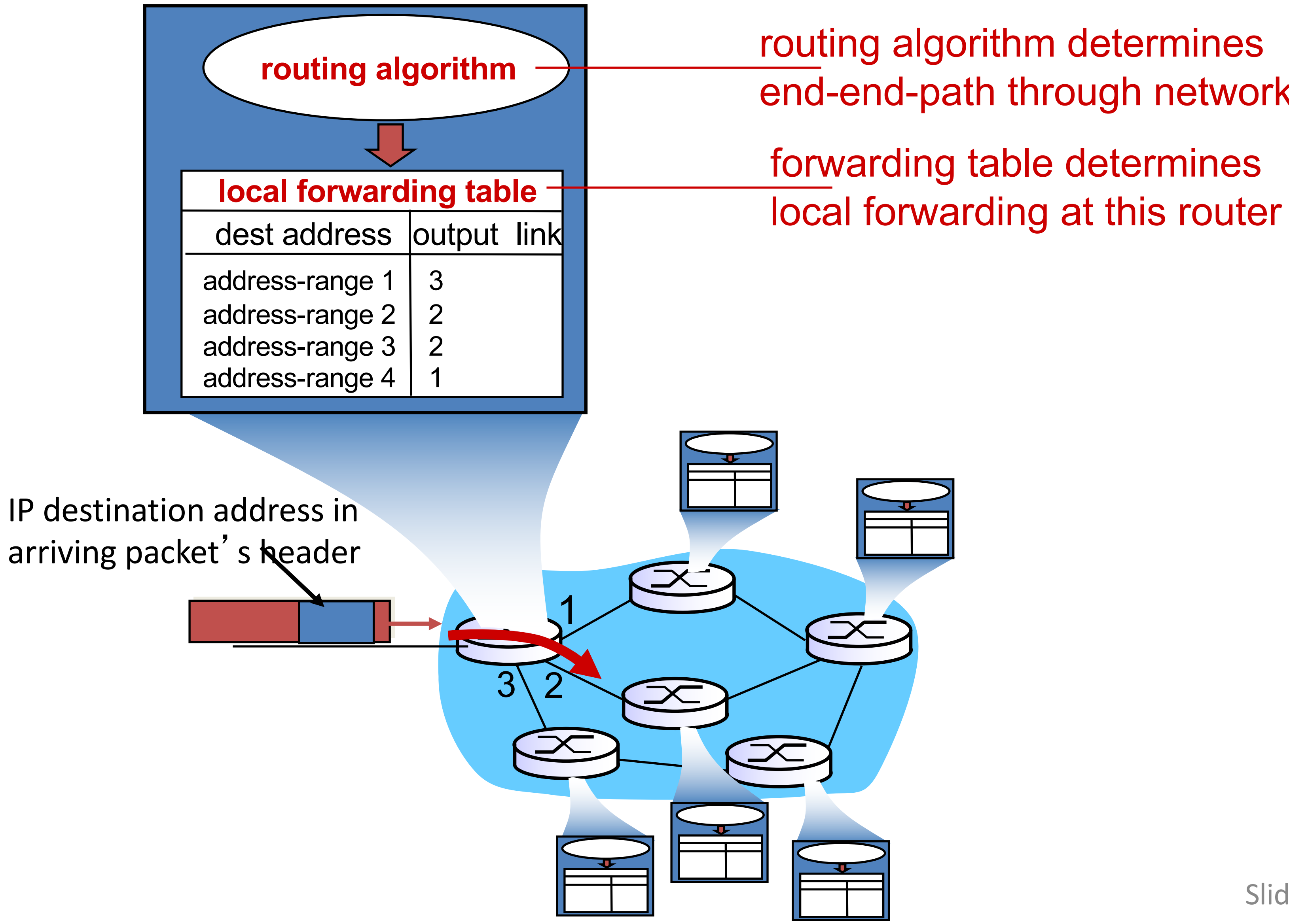


Traceroute Demo

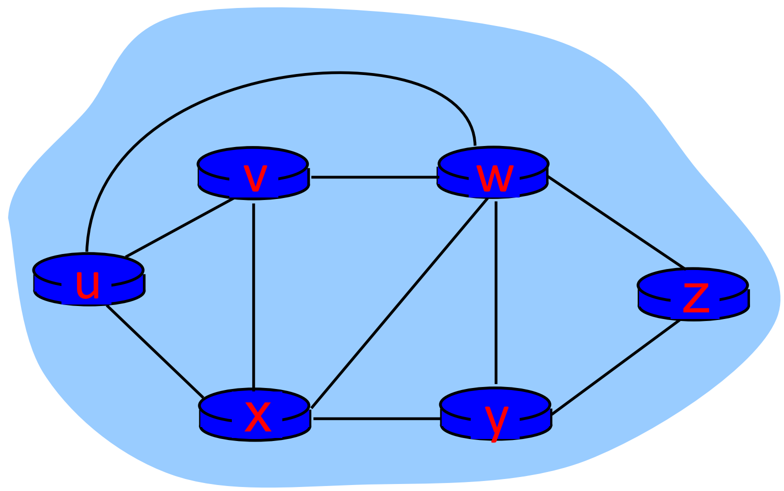


```
6  Episode.IV (206.214.251.1)  68.642 ms  67.307 ms  67.005 ms
 7  A.NEW.HOPE (206.214.251.6)  65.986 ms  68.502 ms  68.708 ms
 8  It.is.a.period.of.civil.war (206.214.251.9)  67.067 ms  70.139 ms  66.52
 9  Rebel.spaceships (206.214.251.14)  70.214 ms  70.192 ms  71.622 ms
10  striking.from.a.hidden.base (206.214.251.17)  71.427 ms  74.206 ms
11  have.won.their.first.victory (206.214.251.22)  71.665 ms  70.434 ms  7
12  against.the.evil.Galactic.Empire (206.214.251.25)  69.218 ms  70.621
13  During.the.battle (206.214.251.30)  69.059 ms  68.931 ms  69.981 ms
14  Rebel.spies.managed (206.214.251.33)  77.247 ms  72.757 ms  77.61
15  to.steal.secret.plans (206.214.251.38)  71.224 ms  71.164 ms  69.543
16  to.the.Empires.ultimate.weapon (206.214.251.41)  68.744 ms  68.824
17  the.DEATH.STAR (206.214.251.46)  72.316 ms  74.551 ms  66.354 ms
18  an.armored.space.station (206.214.251.49)  69.413 ms  70.334 ms  6
19  with.enough.power.to (206.214.251.54)  66.182 ms  66.627 ms  71.23
20  destroy.an.entire.planet (206.214.251.57)  71.926 ms  71.266 ms  70.
21  Pursued.by.the.Empires (206.214.251.62)  67.298 ms  65.956 ms  66.
22  sinister.agents (206.214.251.65)  65.020 ms  67.806 ms  70.508 ms
23  Princess.Leia.races.home (206.214.251.70)  68.894 ms  71.147 ms  71
24  aboard.her.starship (206.214.251.73)  72.130 ms  71.093 ms  74.026
25  custodian.of.the.stolen.plans (206.214.251.78)  68.568 ms  67.939 ms
26  that.can.save.her (206.214.251.81)  67.063 ms  69.874 ms  68.889 m
27  people.and.restore (206.214.251.86)  70.395 ms  70.144 ms
28  freedom.to.the.galaxy (206.214.251.89)  66.098 ms  65.432 ms
29  0-------------------0 (206.214.251.94)  75.931 ms  74.159 ms  80.012
30  0------------------0 (206.214.251.97)  73.026 ms  73.403 ms  73.256
31  0-----------------0 (206.214.251.102)  83.602 ms  82.079 ms  70.743
32  0----------------0 (206.214.251.105)  70.459 ms  69.403 ms  68.782 m
33  0---------------0 (206.214.251.110)  68.516 ms  72.472 ms  71.811 ms
34  0--------------0 (206.214.251.113)  69.056 ms  65.981 ms  68.202 ms
35  0-------------0 (206.214.251.118)  66.790 ms  71.556 ms  74.292 ms
36  0------------0 (206.214.251.121)  68.286 ms  71.042 ms  71.587 ms
37  0-----------0 (206.214.251.126)  72.702 ms  71.785 ms  72.442 ms
38  0----------0 (206.214.251.129)  78.143 ms  74.411 ms  72.828 ms
39  0---------0 (206.214.251.134)  69.692 ms  66.187 ms  67.369 ms
40  0--------0 (206.214.251.137)  69.184 ms  70.678 ms  67.445 ms
41  0-------0 (206.214.251.142)  70.383 ms  68.220 ms  67.543 ms
42  0------0 (206.214.251.145)  67.593 ms  72.970 ms  73.220 ms
43  0-----0 (206.214.251.150)  70.964 ms  69.082 ms  70.831 ms
44  0----0 (206.214.251.153)  73.856 ms  71.848 ms  70.311 ms
45  0---0 (206.214.251.158)  71.517 ms  69.204 ms  69.538 ms
46  0--0 (206.214.251.161)  68.076 ms  68.179 ms  67.620 ms
47  0-0 (206.214.251.166)  68.738 ms  70.518 ms  68.757 ms
48  00 (206.214.251.169)  68.281 ms  70.225 ms  74.811 ms
49  I (206.214.251.174)  70.203 ms  71.668 ms  71.672 ms
50  By.Ryan.Werber (206.214.251.177)  68.900 ms  71.461 ms  72.297 ms
51  When.CCIEs.Get.Bored (206.214.251.182)  75.816 ms  73.957 ms  71.333 ms
52  read.more.at.beaglenetworks.net (206.214.251.185)  70.254 ms  73.799 ms
```

Interplay between routing, forwarding



Graph Abstraction

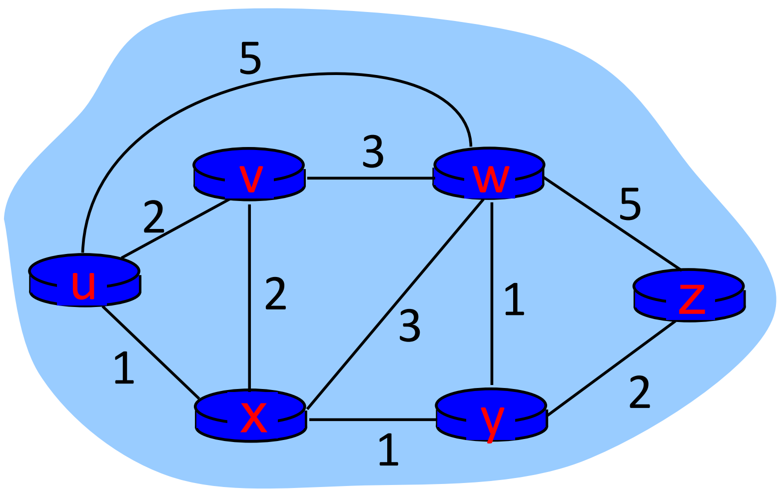


graph: G = (N,E)

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

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

Link Cost



c(x,x') = cost of link (x,x')
e.g., c(w,z) = 5

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

Key question: what is the least-cost path between u and z ?
Routing algorithm: algorithm that finds that least cost path

How should link costs be determined?

A. They should all be equal.

B. They should be a function of link capacity.

C. They should take current traffic characteristics into account (congestion, delay, etc.).

D. They should be manually determined by network administrators.

E. They should be determined in some other way.

Link Cost

- Typically simple: all links are equal
- Least-cost paths => shortest paths (hop count)
- Network operators add policy exceptions
  - Lower operational costs
  - Peering agreements
  - Security concerns

Routing Challenges

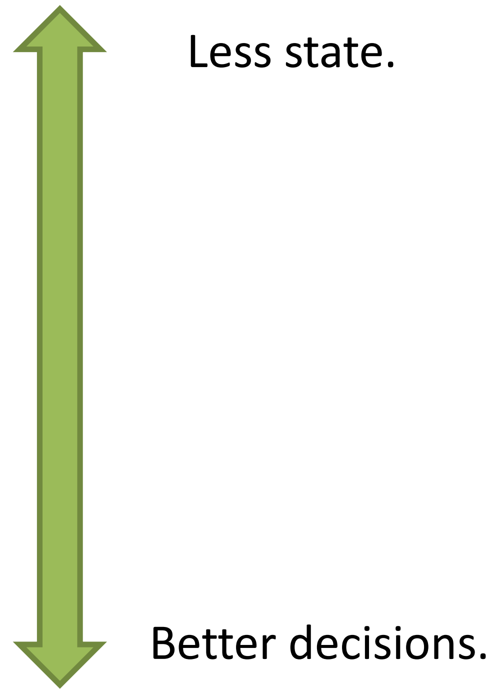
- How to choose best path?
  - Defining “best” can be slippery
- How to scale to millions of users?
  - Minimize control messages and routing table size
- How to adapt quickly to failures or changes?
  - Node and link failures, plus message loss

How much information should a router know about the network?

A. The next hop and cost of forwarding to its neighbor(s).

B. The next hop and cost of forwarding to any destination.

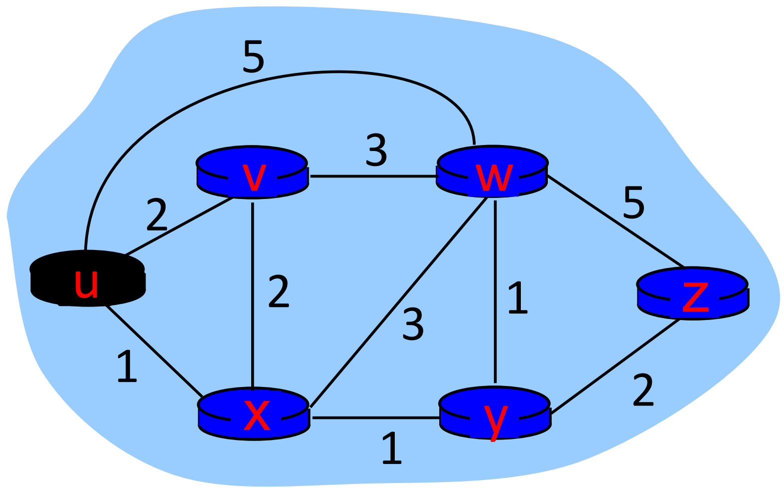
C. The status and cost of every link in the network.



D. Some other amount of information.

Routing Table?

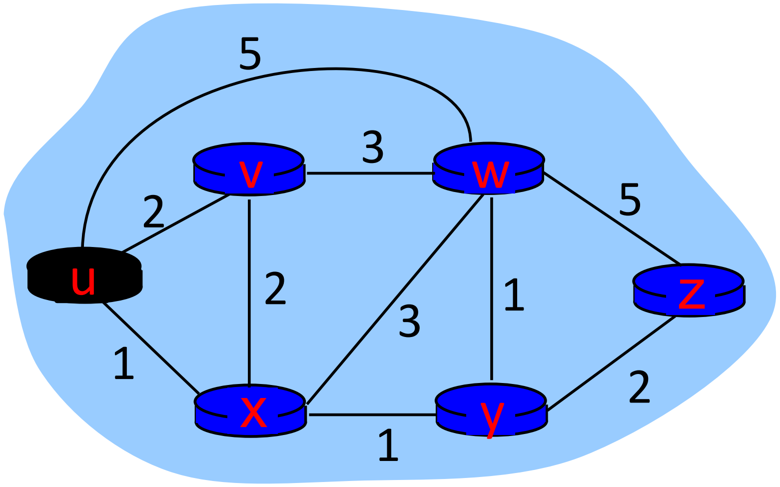
Dest	Next Hop
V	V
X	X
W	X
Y	X
Z	X



- *At a minimum*, the routing table at U needs to know the next hop for each possible destination.

Routing Table

Dest	Next Hop	Cost (Path)
V	V	2
X	X	1
W	X	4
Y	X	2
Z	X	4



- *At a minimum*, the routing table at U needs to know the next hop for each possible destination.
- Probably want more info (e.g., path cost, maybe path itself)
- <u>This is a key difference between routing & forwarding!</u>

Routing Algorithm Classes

Link State (Global)

- Routers maintain cost of each link in the network.
- Connectivity/cost changes flooded to all routers.
- Converges quickly (less inconsistency, looping, etc.).
- Limited network sizes.

Distance Vector (Decentralized)

- Routers maintain next hop & cost of each destination.
- Connectivity/cost changes iteratively propagate from neighbor to neighbor.
- Requires multiple rounds to converge.
- Scales to large networks.

Link-state Routing

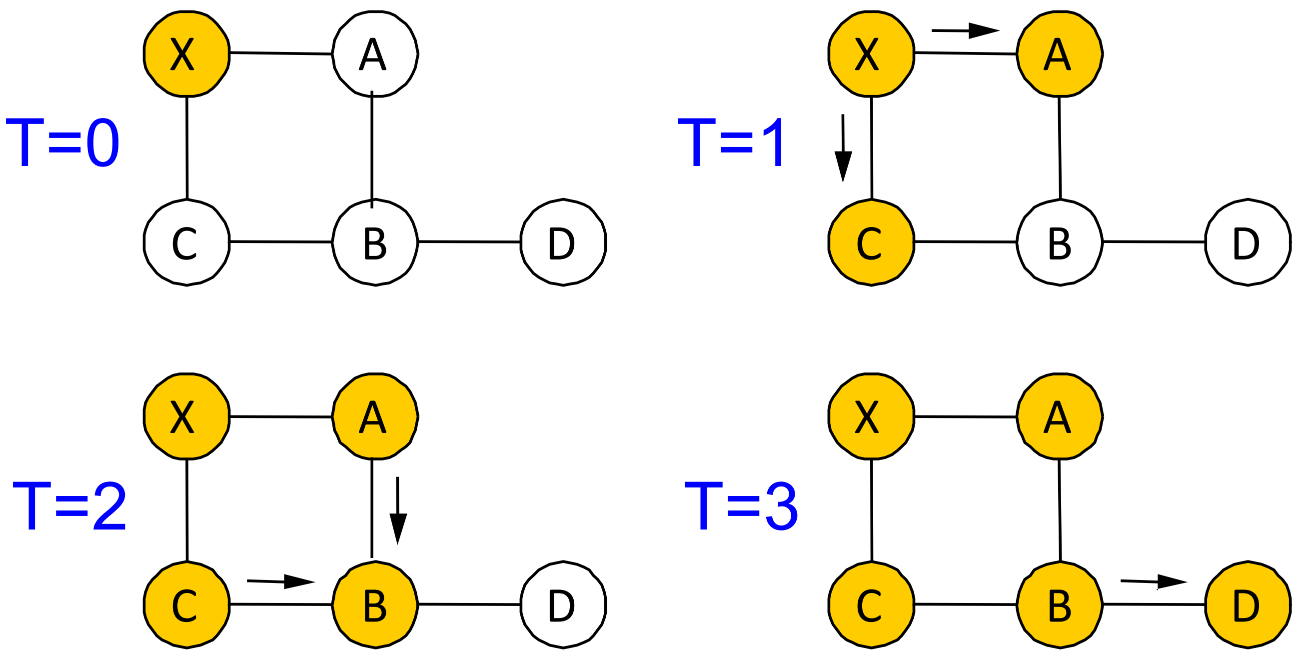
- Two phases
  - Reliable flooding
    - Tell all routers what you know about your links
    - Typically in response to event: link failure/recovery/cost
  - Path calculation (Dijkstra's algorithm)
    - Each router computes best path over complete network
- Motivation
  - Global information allows optimal routing
  - Straightforward to implement and verify

Flooding LSAs

- Routers transmit Link State Advertisement (LSA) on links
  - A neighboring router forwards out all links except incoming
  - Keep a copy locally; don’t forward previously-seen LSAs
- Challenges
  - Packet loss
  - Out-of-order arrival
- Solutions
  - Acknowledgments and retransmissions
  - Sequence numbers
  - Time-to-live for each packet

Flooding Example

- LSA generated by X at T=0



Dijkstra's Algorithm

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

Nodes we've determined lowest-cost path for already.

Best known cost for reaching node v.

Dijkstra’s Algorithm

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

Only know best route to self so far.

For every other router, set it’s known distance to link cost if it’s a neighbor. Otherwise, set it to infinity.

Dijkstra's Algorithm

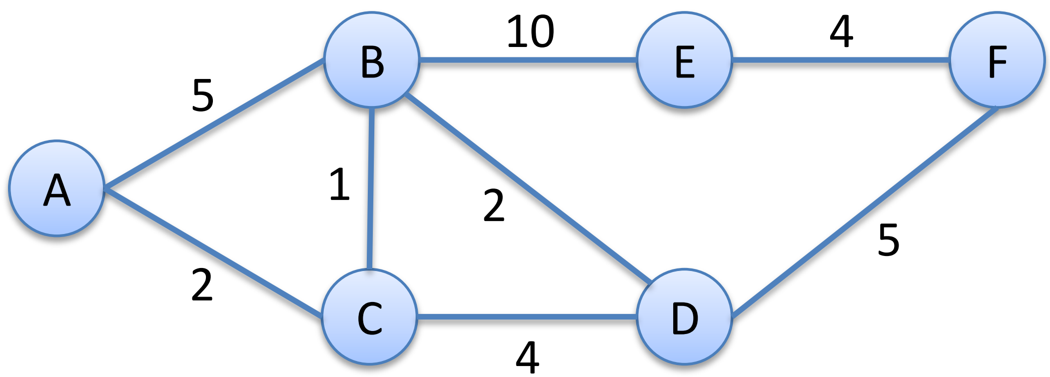
```
Initialization:
  N' = {u}
  for all nodes v
    if v adjacent to u
      then D(v) = c(u,v)
    else D(v) = ∞

Loop
  find w not in N' such that D(w) is a 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) )
  /* new cost to v is either old cost to v or known
   shortest path cost to w plus cost from w to v */
until all nodes in N'
```

Pick the node (w) that isn't already in N' with the shortest distance (least cost path) and add it to N'.

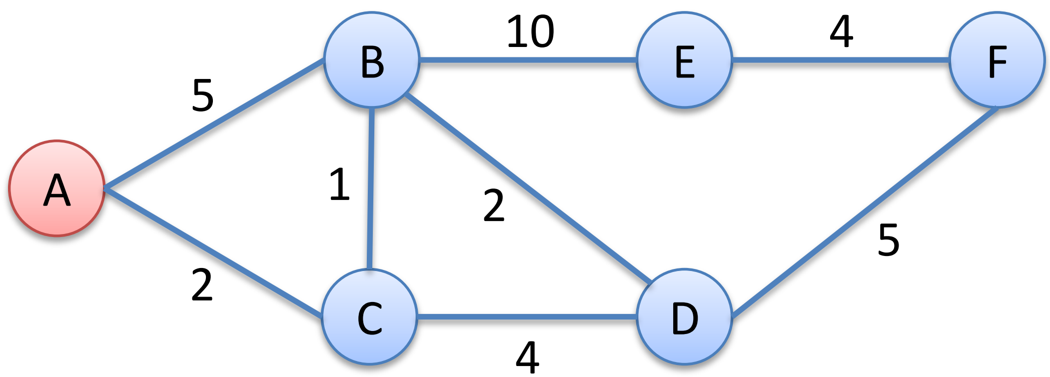
Check all possible destinations from w. If going through w gives a lower cost to destination v, update D(v).

Dijkstra's Algorithm Example



- Goal: From the perspective of node A:
  - Determine shortest path to every destination
- Other perspectives:
  - Look up "Dijkstra's Algorithm" on YouTube

Dijkstra's Algorithm – Step 0



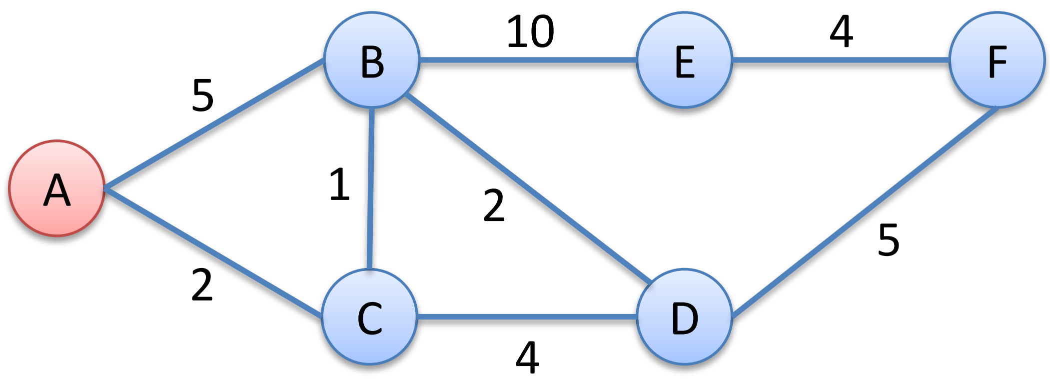
Previous Step

Dest	Path	Cost D(v)
A		
B		
C		
D		
E		
F		

This Step

Dest	Path	Cost D(v)
✓ A	A	0
B	B	5
C	C	2
D	?	∞
E	?	∞
F	?	∞

Dijkstra's Algorithm – Step 1



Previous Step

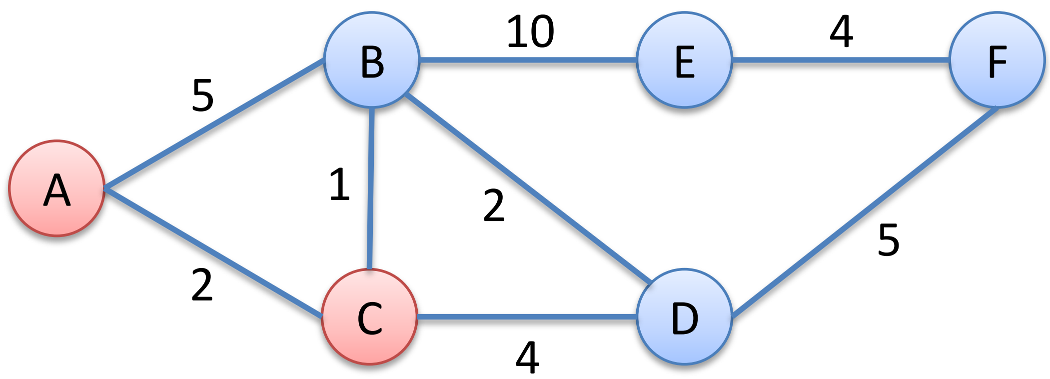
Dest	Path	Cost D(v)
A	A	0
B	B	5
C	C	2
D	?	∞
E	?	∞
F	?	∞

Pick Min

This Step

Dest	Path	Cost D(v)
A	A	0
B		
C		
D		
E		
F		

Dijkstra's Algorithm – Step 1



Can we find lower cost to any other node by going through C?

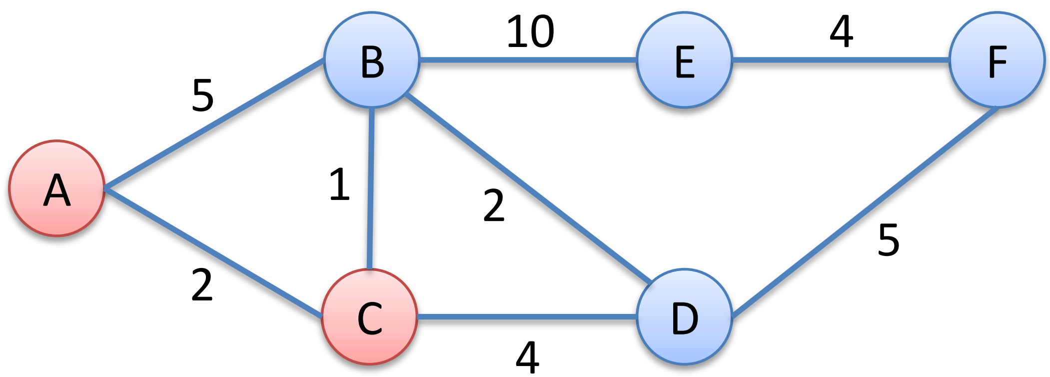
Previous Step

Dest	Path	Cost D(v)
✓ A	A	0
B	B	5
C	C	2
D	?	∞
E	?	∞
F	?	∞

This Step

Dest	Path	Cost D(v)
✓ A	A	0
B		
✓ C	C	2
D		
E		
F		

Dijkstra's Algorithm – Step 1



Consider path to B:

D(B)
or
D(C) + cost(C, B)

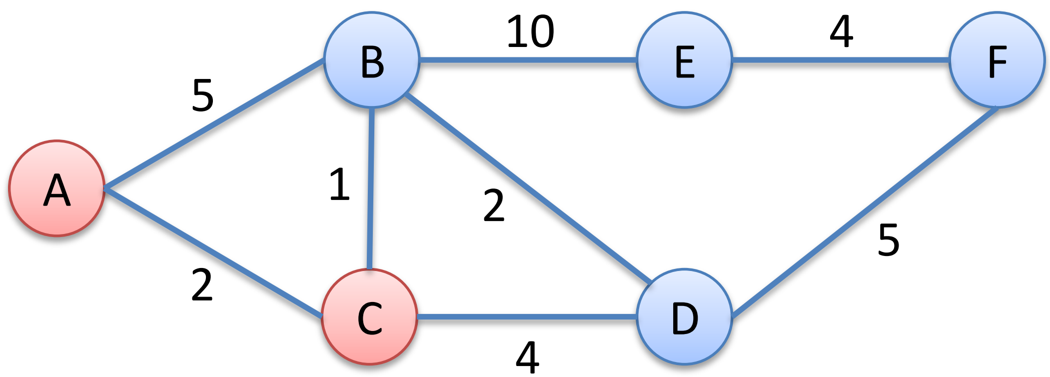
Previous Step

Dest	Path	Cost D(v)
✓ A	A	0
B	B	5
C	C	2
D	?	∞
E	?	∞
F	?	∞

This Step

Dest	Path	Cost D(v)
✓ A	A	0
B		
✓ C	C	2
D		
E		
F		

Dijkstra's Algorithm – Step 1



Consider path to B:

D(B) = 5
or
D(C) + cost(C, B)
2 + 1 = 3

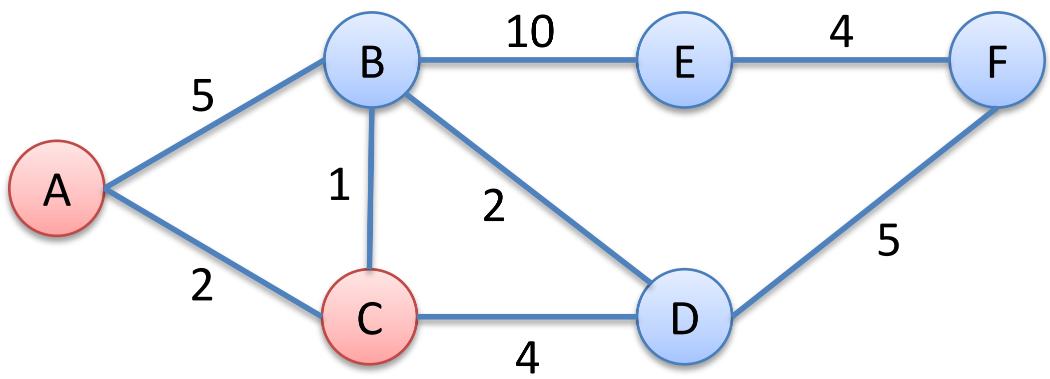
Previous Step

Dest	Path	Cost D(v)
A	A	0
B	B	5
C	C	2
D	?	∞
E	?	∞
F	?	∞

This Step

Dest	Path	Cost D(v)
A	A	0
B	C, B	3
C	C	2
D		
E		
F		

Dijkstra's Algorithm – Step 1



Consider path to D:

D(D) = ∞
or
D(C) + cost(C, D)
2 + 4 = 6

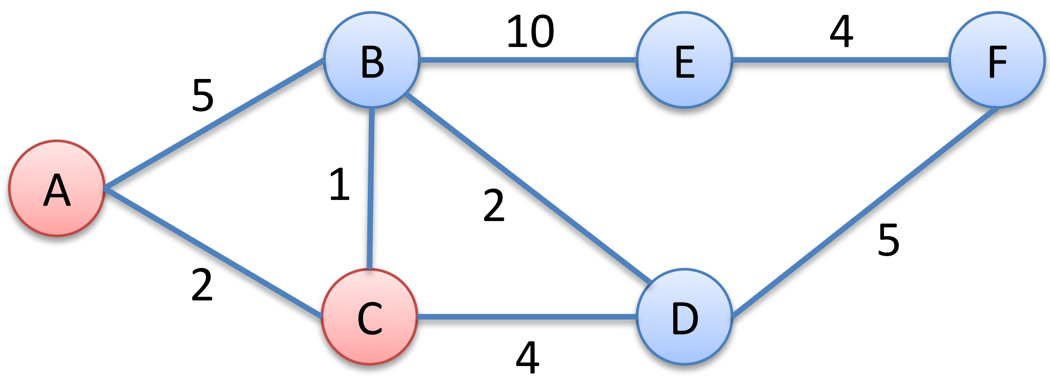
Previous Step

	Dest	Path	Cost D(v)
✓	A	A	0
	B	B	5
	C	C	2
	D	?	∞
	E	?	∞
	F	?	∞

This Step

	Dest	Path	Cost D(v)
✓	A	A	0
	B	C, B	3
✓	C	C	2
	D	C, D	6
	E		
	F		

Dijkstra's Algorithm – Step 1



Still no information about E or F.

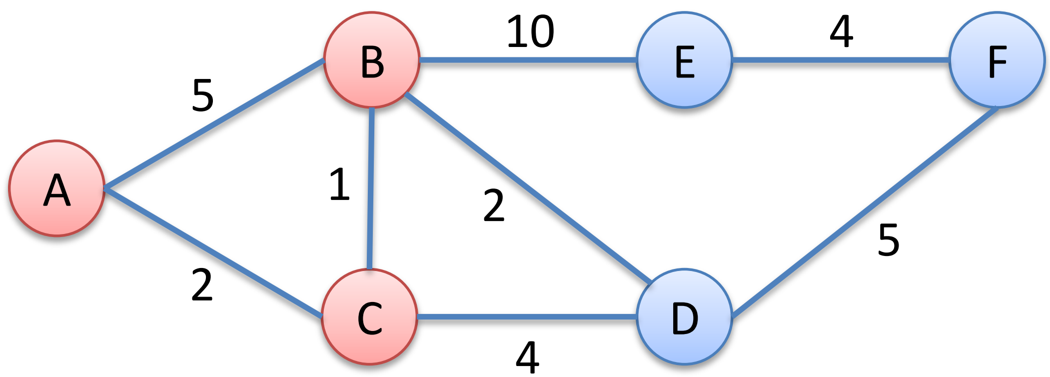
Previous Step

Dest	Path	Cost D(v)
✓ A	A	0
B	B	5
C	C	2
D	?	∞
E	?	∞
F	?	∞

This Step

Dest	Path	Cost D(v)
✓ A	A	0
B	C, B	3
✓ C	C	2
D	C, D	6
E	?	∞
F	?	∞

Dijkstra's Algorithm – Step 2



Choose B.

Previous Step

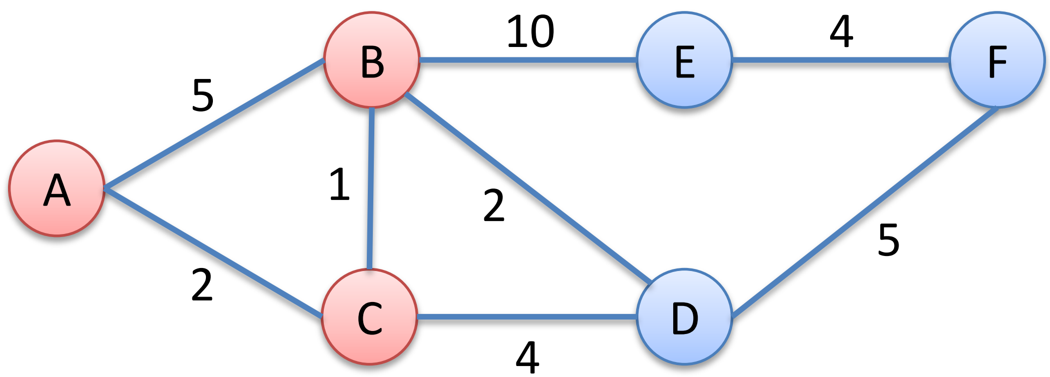
Dest	Path	Cost D(v)
✓ A	A	0
B	C, B	3
✓ C	C	2
D	C, D	6
E	?	∞
F	?	∞

Pick Min

This Step

Dest	Path	Cost D(v)
✓ A	A	0
✓ B	C, B	3
✓ C	C	2
D		
E		
F		

Dijkstra's Algorithm – Step 2



Consider path to D:

D(D) = 6
or
D(B) + cost(B, D)
3 + 2 = 5

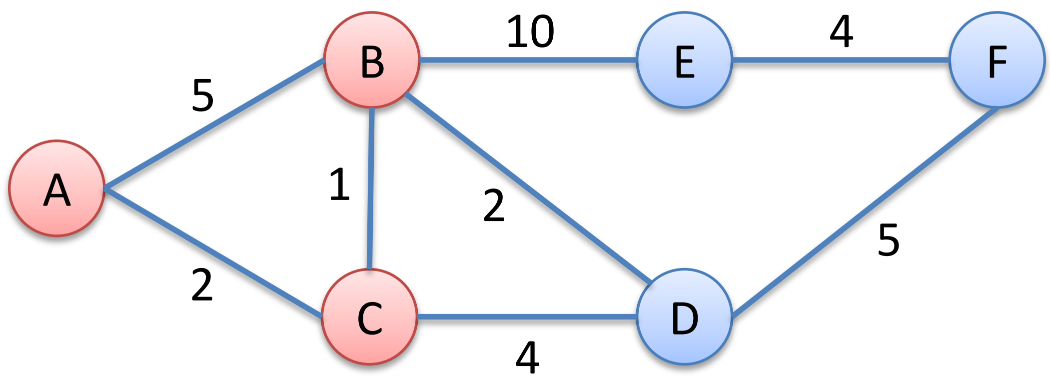
Previous Step

Dest	Path	Cost D(v)
✓ A	A	0
B	C, B	3
✓ C	C	2
D	C, D	6
E	?	∞
F	?	∞

This Step

Dest	Path	Cost D(v)
✓ A	A	0
✓ B	C, B	3
✓ C	C	2
D	C, B, D	5
E		
F		

Dijkstra's Algorithm – Step 2



Consider path to E:

D(E) = ∞
or
D(B) + cost(B, E)
3 + 10 = 13

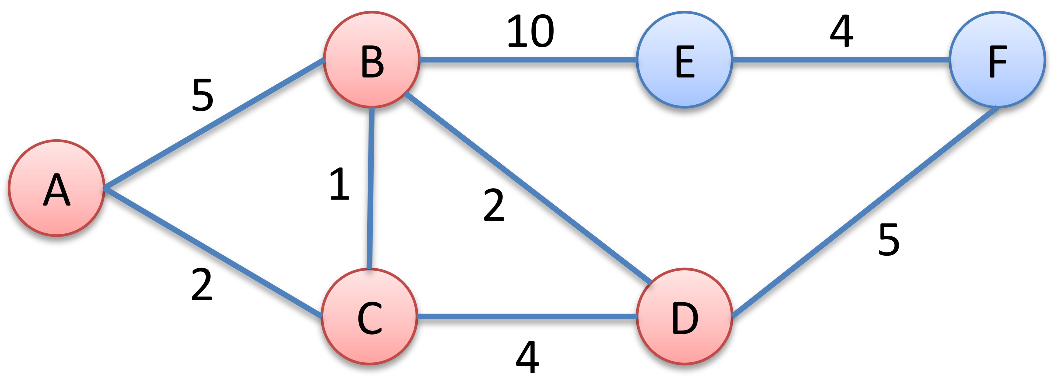
Previous Step

	Dest	Path	Cost D(v)
✓	A	A	0
	B	C, B	3
✓	C	C	2
	D	C, D	6
	E	?	∞
	F	?	∞

This Step

	Dest	Path	Cost D(v)
✓	A	A	0
✓	B	C, B	3
✓	C	C	2
	D	C, B, D	5
	E	C, B, E	13
	F	?	∞

Dijkstra's Algorithm – Step 3



Choose D.

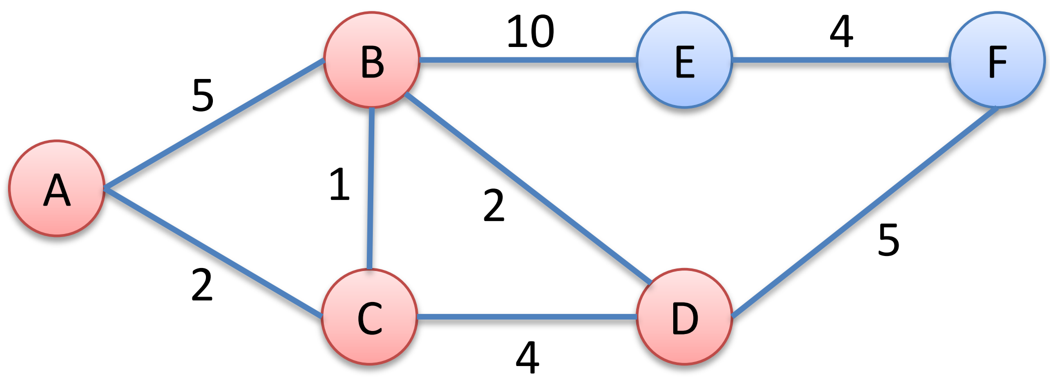
Previous Step

Dest	Path	Cost D(v)
✓ A	A	0
✓ B	C, B	3
✓ C	C	2
D	C, B, D	5
E	C, B, E	13
F	?	∞

This Step

Dest	Path	Cost D(v)
✓ A	A	0
✓ B	C, B	3
✓ C	C	2
✓ D	C, B, D	5
E		
F		

Dijkstra's Algorithm – Step 3



No change for E.

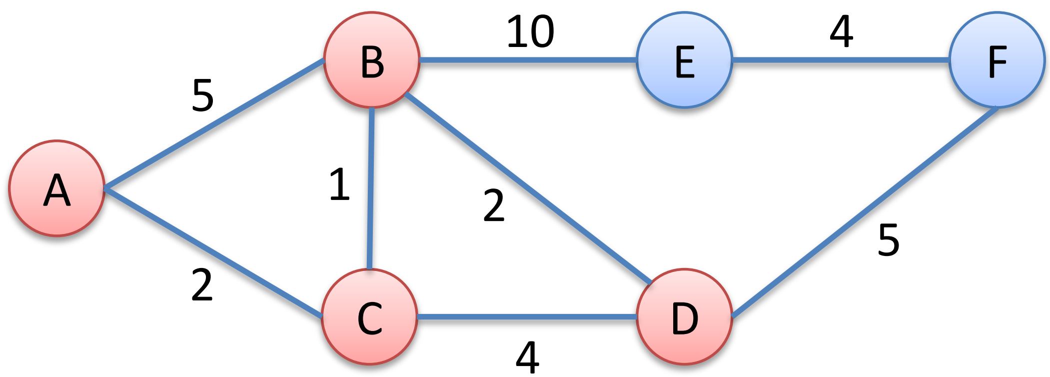
Previous Step

Dest	Path	Cost D(v)
✓ A	A	0
✓ B	C, B	3
✓ C	C	2
D	C, B, D	5
E	C, B, E	13
F	?	∞

This Step

Dest	Path	Cost D(v)
✓ A	A	0
✓ B	C, B	3
✓ C	C	2
✓ D	C, B, D	5
E	C, B, E	13
F		

Dijkstra's Algorithm – Step 3



Consider path to F:

D(F) = ∞
or
D(D) + cost(D, F)
5 + 5 = 10

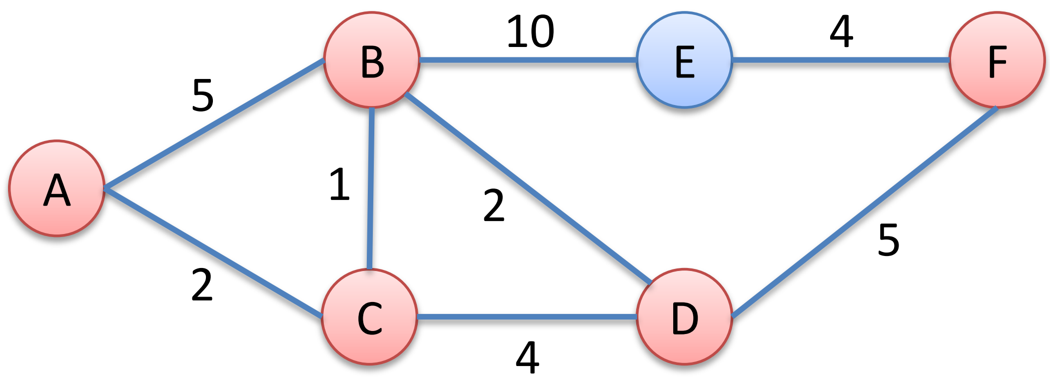
Previous Step

Dest	Path	Cost D(v)
✓ A	A	0
✓ B	C, B	3
✓ C	C	2
D	C, B, D	5
E	C, B, E	13
F	?	∞

This Step

Dest	Path	Cost D(v)
✓ A	A	0
✓ B	C, B	3
✓ C	C	2
✓ D	C, B, D	5
E	C, B, E	13
F	C, B, D, F	10

Dijkstra's Algorithm – Step 4



Choose F.

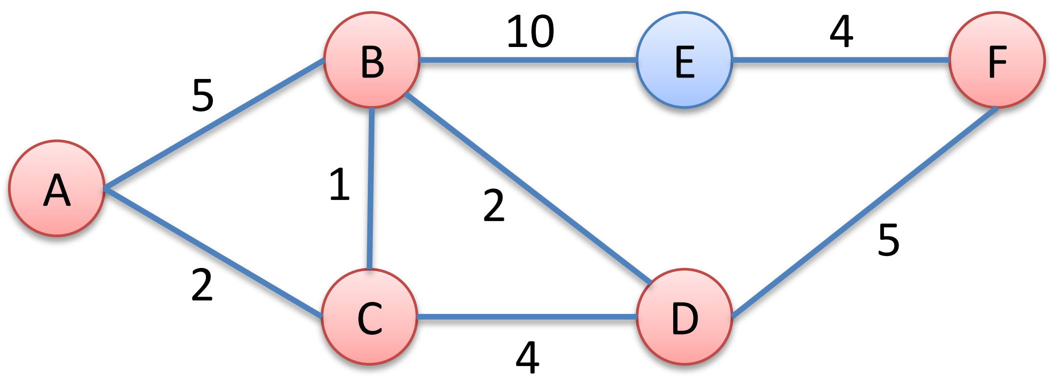
Previous Step

	Dest	Path	Cost D(v)
✓	A	A	0
✓	B	C, B	3
✓	C	C	2
✓	D	C, B, D	5
	E	C, B, E	13
	F	C, B, D, F	10

This Step

	Dest	Path	Cost D(v)
✓	A	A	0
✓	B	C, B	3
✓	C	C	2
✓	D	C, B, D	5
✓	F	C, B, D, F	10

Dijkstra's Algorithm – Step 4



Consider path to E:

D(E) = 13
or
D(F) + cost(F, E)
10 + 4 = 14

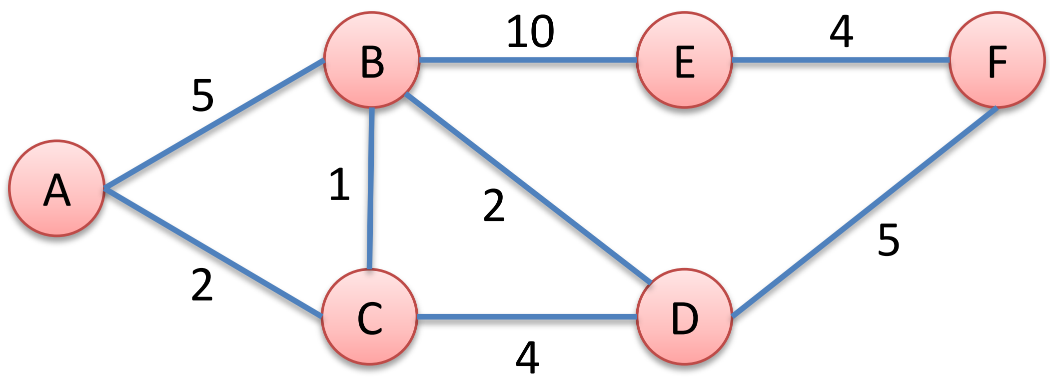
Previous Step

	Dest	Path	Cost D(v)
✓	A	A	0
✓	B	C, B	3
✓	C	C	2
✓	D	C, B, D	5
	E	C, B, E	13
	F	C, B, D, F	10

This Step

	Dest	Path	Cost D(v)
✓	A	A	0
✓	B	C, B	3
✓	C	C	2
✓	D	C, B, D	5
	E	C, B, E	13
✓	F	C, B, D, F	10

Dijkstra's Algorithm – Step 5



Choose E.

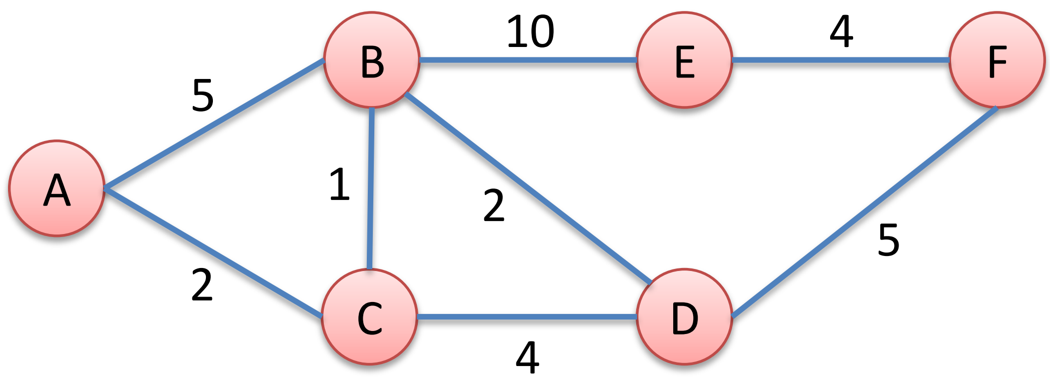
Previous Step

	Dest	Path	Cost D(v)
✓	A	A	0
✓	B	C, B	3
✓	C	C	2
✓	D	C, B, D	5
	E	C, B, E	13
✓	F	C, B, D, F	10

This Step

	Dest	Path	Cost D(v)
✓	A	A	0
✓	B	C, B	3
✓	C	C	2
✓	D	C, B, D	5
✓	E	C, B, E	13
✓	F	C, B, D, F	10

Dijkstra's Algorithm – Done!



Lot more state in routing table!

Final Answer

Dest	Path	Cost D(v)
A	A	0
B	C, B	3
C	C	2
D	C, B, D	5
E	C, B, E	13
F	C, B, D, F	10

Populate Forwarding Table

Forwarding Table

Dest	Forward To
B	C
C	C
D	C
E	C
F	C

Dijkstra's Algorithm – Complexity

- With N nodes and E edges…
- As previously described it's $O(N^2)$
  - At each step, there are N nodes to choose next
  - Total of N steps (each node must be chosen)
- Fastest known is $O(N \log N + E)$
  - Uses a min-heap

Link State - Summary

- + Fast convergence (reacts to events quickly)
- + Small window of inconsistency

- − Large number of messages sent on events
- − Large routing tables as network size grows

Routing Algorithm Classes

Link State (Global)

- Routers maintain cost of each link in the network.
- Connectivity/cost changes flooded to all routers.
- Converges quickly (less inconsistency, looping, etc.).
- Limited network sizes.

Distance Vector (Decentralized)

- Routers maintain next hop & cost of each destination.
- Connectivity/cost changes iteratively propagate from neighbor to neighbor.
- Requires multiple rounds to converge.
- Scales to large networks.

Bellman-Ford Equation

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

Distance Vectors

- Let $D_x(y)$ = vector of least cost from x to y
- Node x:
  - Knows cost to each neighbor v: c(x,v)
  - Maintains its neighbors' distance vectors.
    For each neighbor v, x maintains:
    $\mathbf{D}_v = [D_v(y): y \in N]$
- **As opposed to link state:**
  - **Only keeps state for yourself and direct neighbors**

Distance Vector Algorithm

- Periodically, each node sends its own distance vector to neighbors
- Upon receiving new DV from neighbor, update its local DV using B-F equation:

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

- Under typical conditions, $D_x(y)$ will converge to the least cost $d_x(y)$

Distance Vector Algorithm

Iterative, asynchronous:

Iteration when:

- Local link cost change
- DV update from neighbor
- Periodic timer

Distributed:

- Each node knows only a portion of global link info

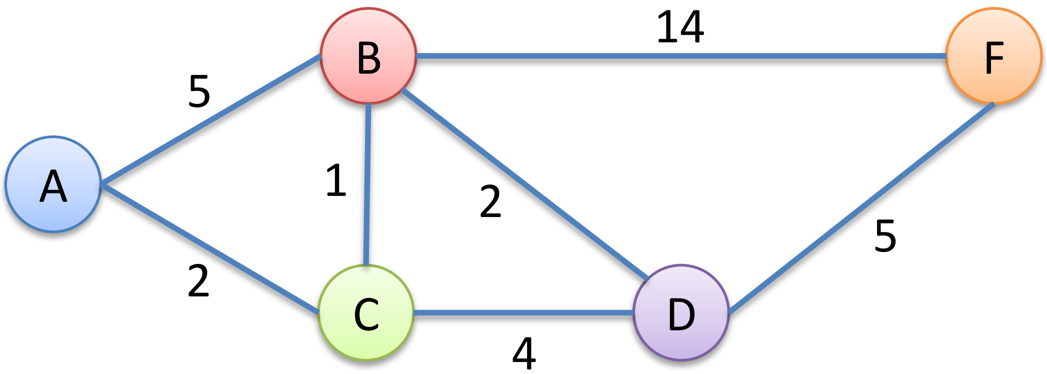
each node:

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

recompute estimates

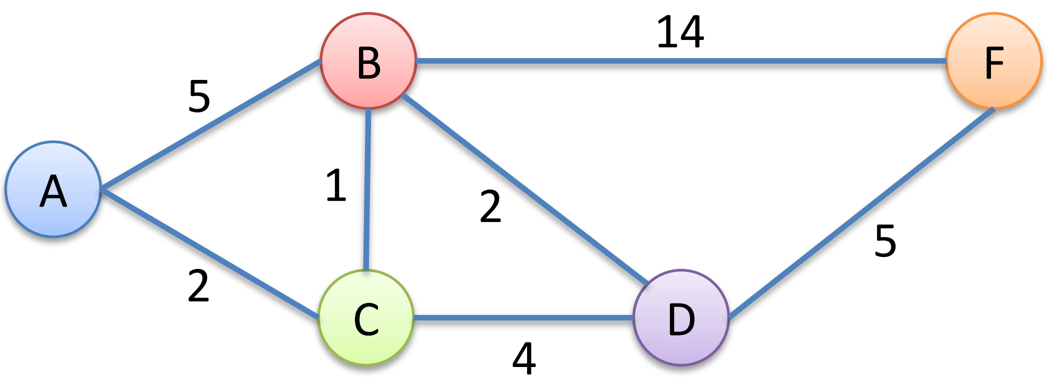
if DV to any dest has changed, *notify* neighbors

Distance Vector Example



- Same network as Dijkstra's example, without node E.
- What I'll show you next is routing table (of distance vectors) at each router.

Distance Vector – Round 0



Routers populate their forwarding table by taking the row minimum.

Router F

Via→ ↓ To	B	D
A		
B	14	
C		
D		5

Router A

Via→ ↓ To	B	C
B	5	
C		2
D		
F		

Router B

Via→ ↓ To	A	C	D	F
A	5			
C		1		
D			2	
F				14

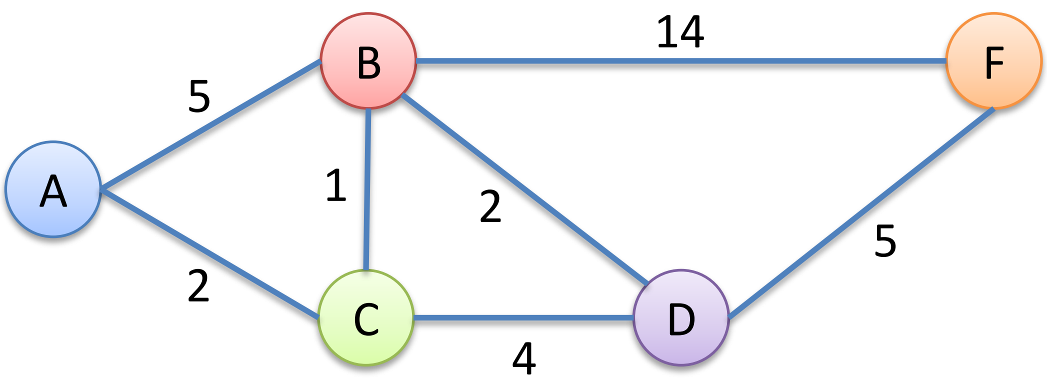
Router C

Via→ ↓ To	A	B	D
A	2		
B		1	
D			4
F			

Router D

Via→ ↓ To	B	C	F
A			
B	2		
C		4	
F			5

Distance Vector – Round 0



Router exchange their local vectors with direct neighbors.
We'll assume they all exchange at once (synchronous). (Not realistic)

Router F

Via→ ↓ To	B	D
A		
B	14	
C		
D		5

Router A

Via→ ↓ To	B	C
B	5	
C		2
D		
F		

Router B

Via→ ↓ To	A	C	D	F
A	5			
C		1		
D			2	
F				14

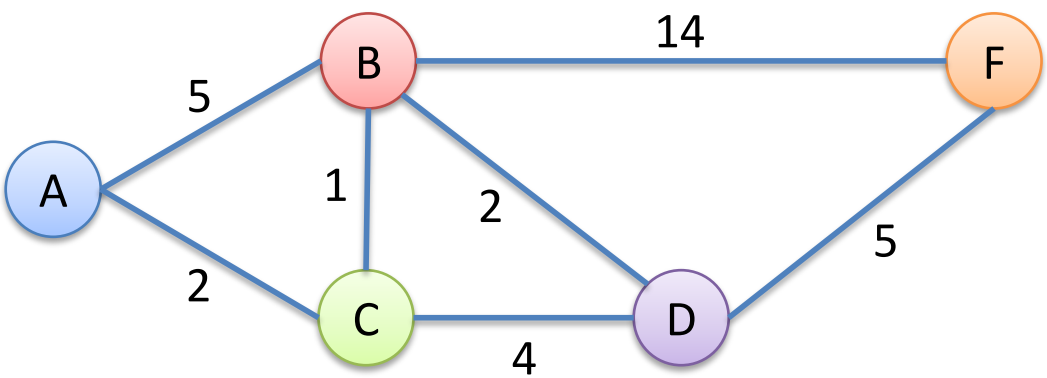
Router C

Via→ ↓ To	A	B	D
A	2		
B		1	
D			4
F			

Router D

Via→ ↓ To	B	C	F
A			
B	2		
C		4	
F			5

Distance Vector – Round 1



A will send to neighbors (B & C):
I can get to B in 5 and C in 2.

Router F

Via→ ↓ To	B	D
A		
B	14	
C		
D		5

Router A

Via→ ↓ To	B	C
B	5	
C		2
D		
F		

Router B

Via→ ↓ To	A	C	D	F
A	5			
C	7	1		
D			2	
F				14

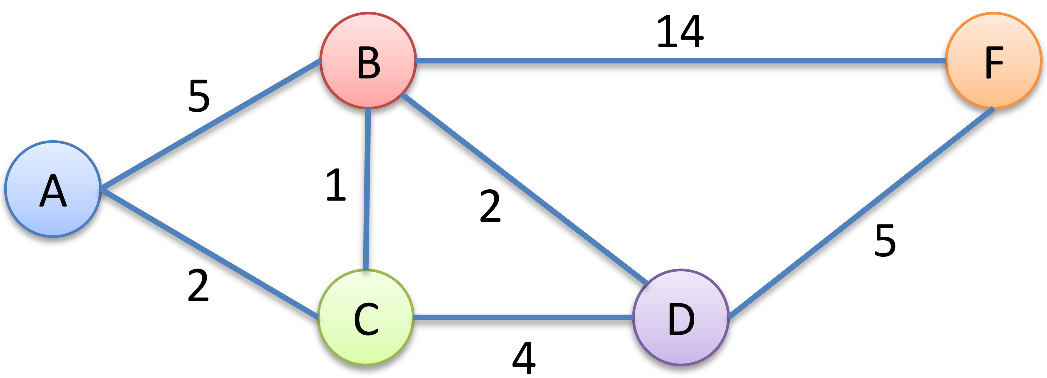
Router C

Via→ ↓ To	A	B	D
A	2		
B	7	1	
D			4
F			

Router D

Via→ ↓ To	B	C	F
A			
B	2		
C		4	
F			5

Distance Vector – Round 1



B will send to neighbors (A, C, D, F):
I can get to A in 5, C in 1, D in 2, and F in 14.

Router F

Via→ ↓ To	B	D
A	19	
B	14	
C	15	
D	16	5

Router A

Via→ ↓ To	B	C
B	5	
C	6	2
D	7	
F	19	

Router B

Via→ ↓ To	A	C	D	F
A	5			
C	7	1		
D			2	
F				14

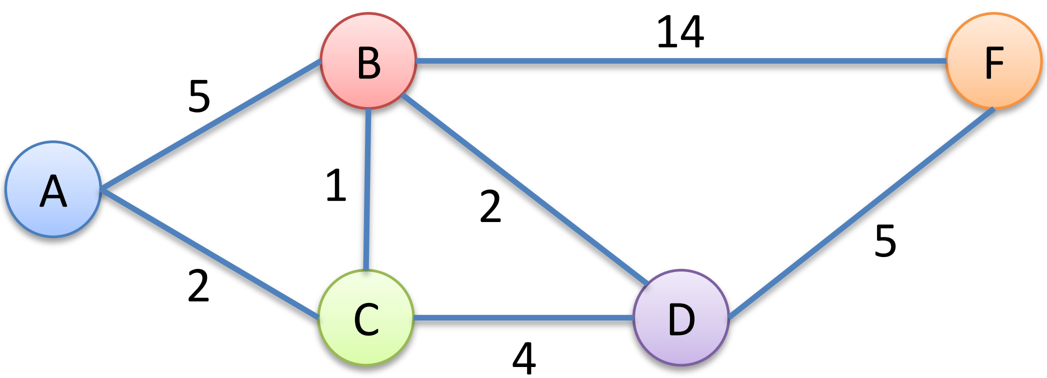
Router C

Via→ ↓ To	A	B	D
A	2	6	
B	7	1	
D		3	4
F		15	

Router D

Via→ ↓ To	B	C	F
A	7		
B	2		
C	3	4	
F	16		5

Distance Vector – Round 1



C will send to neighbors (A, B, D):
I can get to A in 2, B in 1, and D in 4.

Router F

Via→ ↓ To	B	D
A	19	
B	14	
C	15	
D	16	5

Router A

Via→ ↓ To	B	C
B	5	3
C	6	2
D	7	6
F	19	

Router B

Via→ ↓ To	A	C	D	F
A	5	3		
C	7	1		
D		5	2	
F				14

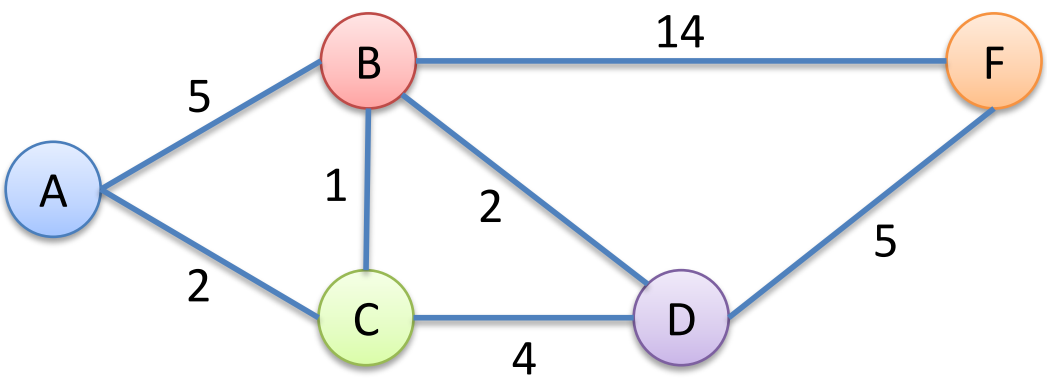
Router C

Via→ ↓ To	A	B	D
A	2	6	
B	7	1	
D		3	4
F		15	

Router D

Via→ ↓ To	B	C	F
A	7	6	
B	2	5	
C	3	4	
F	16		5

Distance Vector – Round 1



D will send to neighbors (B, C, F):
I can get to B in 2, C in 4, and F in 5.

Router F

Via→ ↓ To	B	D
A	19	
B	14	7
C	15	9
D	16	5

Router A

Via→ ↓ To	B	C
B	5	3
C	6	2
D	7	6
F	19	

Router B

Via→ ↓ To	A	C	D	F
A	5	3		
C	7	1	6	
D		5	2	
F			7	14

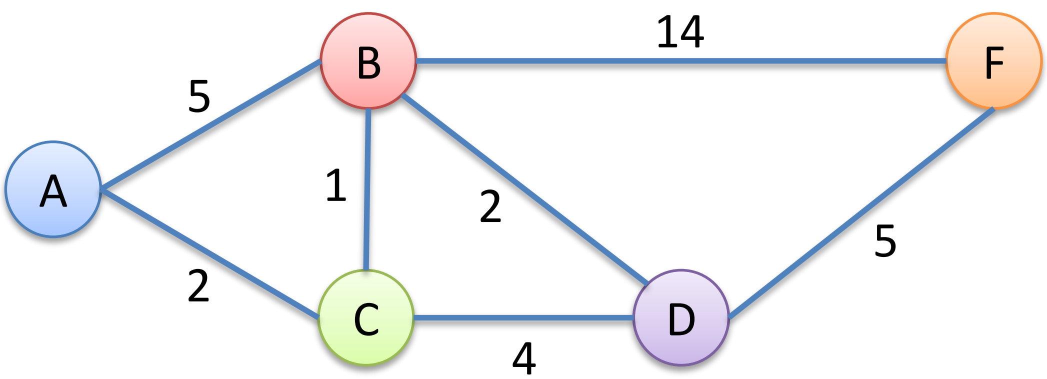
Router C

Via→ ↓ To	A	B	D
A	2	6	
B	7	1	6
D		3	4
F		15	9

Router D

Via→ ↓ To	B	C	F
A	7	6	
B	2	5	
C	3	4	
F	16		5

Distance Vector – Round 1



F will send to neighbors (B, D):
I can get to B in 14, D in 5.

Router F

Via→ ↓ To	B	D
A	19	
B	14	7
C	15	9
D	16	5

Router A

Via→ ↓ To	B	C
B	5	3
C	6	2
D	7	6
F	19	

Router B

Via→ ↓ To	A	C	D	F
A	5	3		
C	7	1	6	
D		5	2	19
F			7	14

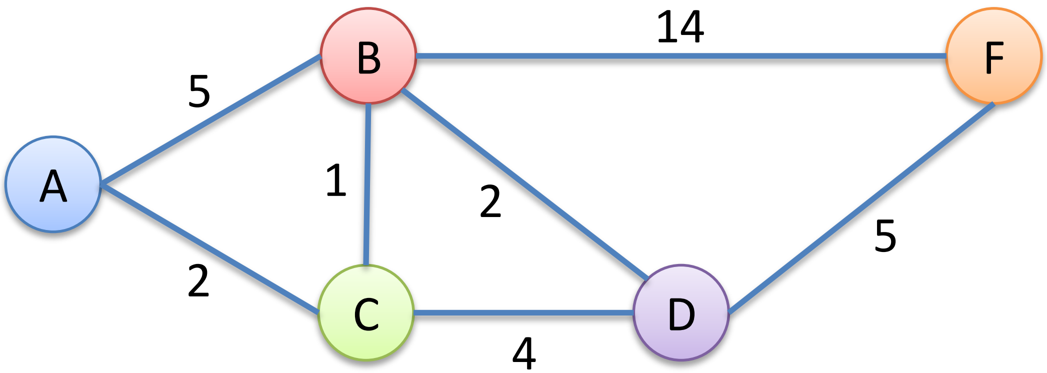
Router C

Via→ ↓ To	A	B	D
A	2	6	
B	7	1	6
D		3	4
F		15	9

Router D

Via→ ↓ To	B	C	F
A	7	6	
B	2	5	19
C	3	4	
F	16		5

At the end of round 1, how many routers need to update their forwarding tables?



A – 1, B – 2, C – 3, D – 4, E – 5

Router F

Via→ ↓ To	B	D
A	19	
B	14	7
C	15	9
D	16	5

Router A

Via→ ↓ To	B	C
B	5	3
C	6	2
D	7	6
F	19	

Router B

Via→ ↓ To	A	C	D	F
A	5	3		
C	7	1	6	
D		5	2	19
F			7	14

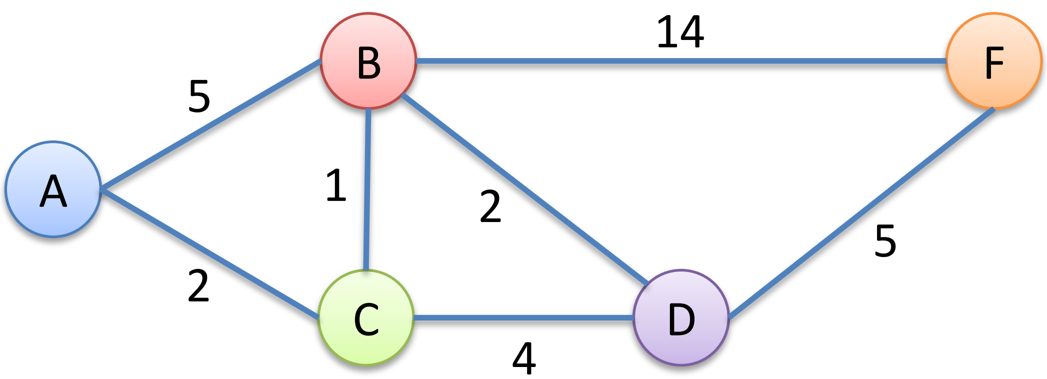
Router C

Via→ ↓ To	A	B	D
A	2	6	
B	7	1	6
D		3	4
F		15	9

Router D

Via→ ↓ To	B	C	F
A	7	6	
B	2	5	19
C	3	4	
F	16		5

Distance Vector – Round 2



Each router advertises the best cost it has to each destination.
Nothing new to learn from A or F, so we'll skip their announcements.

Router F

Via→ ↓ To	B	D
A	19	
B	14	7
C	15	9
D	16	5

Router A

Via→ ↓ To	B	C
B	5	3
C	6	2
D	7	6
F	19	

Router B

Via→ ↓ To	A	C	D	F
A	5	3		
C	7	1	6	
D		5	2	19
F			7	14

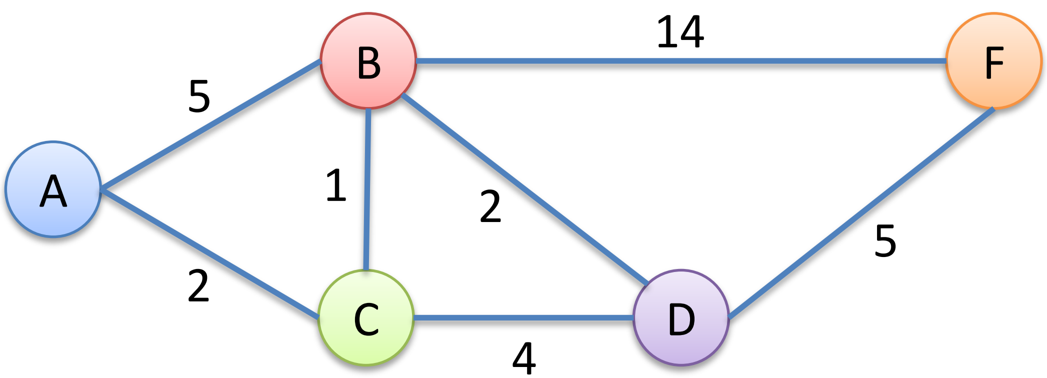
Router C

Via→ ↓ To	A	B	D
A	2	6	
B	7	1	6
D		3	4
F		15	9

Router D

Via→ ↓ To	B	C	F
A	7	6	
B	2	5	19
C	3	4	
F	16		5

Distance Vector – Round 2



B will send to neighbors (A, C, D, F):
I can get to A in 3, C in 1, D in 2, and F in 7.

Router F

Via→ ↓ To	B	D
A	17	
B	14	7
C	15	9
D	16	5

Router A

Via→ ↓ To	B	C
B	5	3
C	6	2
D	7	6
F	12	

Router B

Via→ ↓ To	A	C	D	F
A	5	3		
C	7	1	6	
D		5	2	19
F			7	14

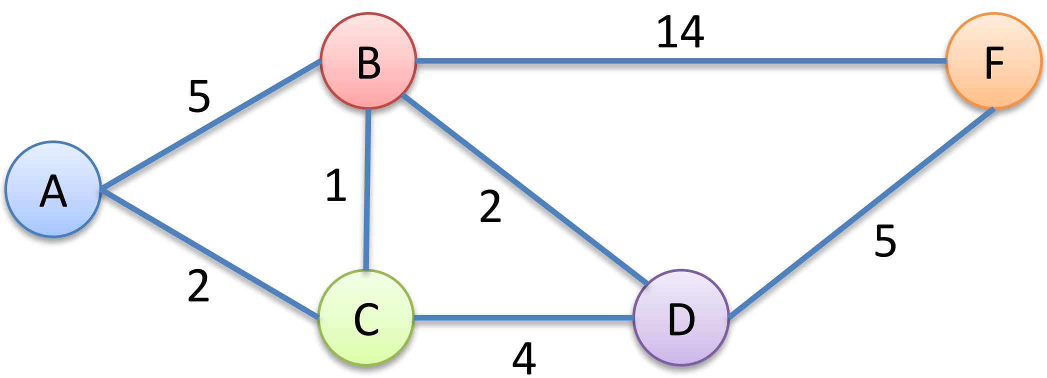
Router C

Via→ ↓ To	A	B	D
A	2	4?	
B	7	1	6
D		3	4
F		8	9

Router D

Via→ ↓ To	B	C	F
A	5	6	
B	2	5	19
C	3	4	
F	9?		5

Distance Vector – Round 2



Router F

Via→ ↓ To	B	D
A	17	
B	14	7
C	15	9
D	16	5

C will send to neighbors (A, B, D):
I can get to A in 2, B in 1, D in 3, and F in 9.

Router A

Via→ ↓ To	B	C
B	5	3
C	6	2
D	7	5
F	12	11

Router B

Via→ ↓ To	A	C	D	F
A	5	3		
C	7	1	6	
D		4?	2	19
F		10	7	14

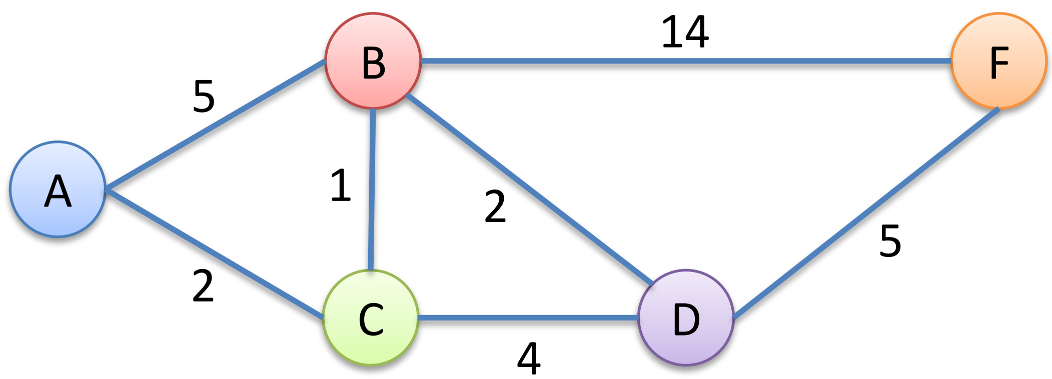
Router C

Via→ ↓ To	A	B	D
A	2	4?	
B	7	1	6
D		3	4
F		8	9

Router D

Via→ ↓ To	B	C	F
A	5	6	
B	2	5	19
C	3	4	
F	9?	13?	5

Distance Vector – Round 2



This process repeats for a while…

Router F

Via→ ↓ To	B	D
A	17	
B	14	7
C	15	9
D	16	5

Router A

Via→ ↓ To	B	C
B	5	3
C	6	2
D	7	5
F	12	11

Router B

Via→ ↓ To	A	C	D	F
A	5	3		
C	7	1	6	
D		4?	2	19
F		10	7	14

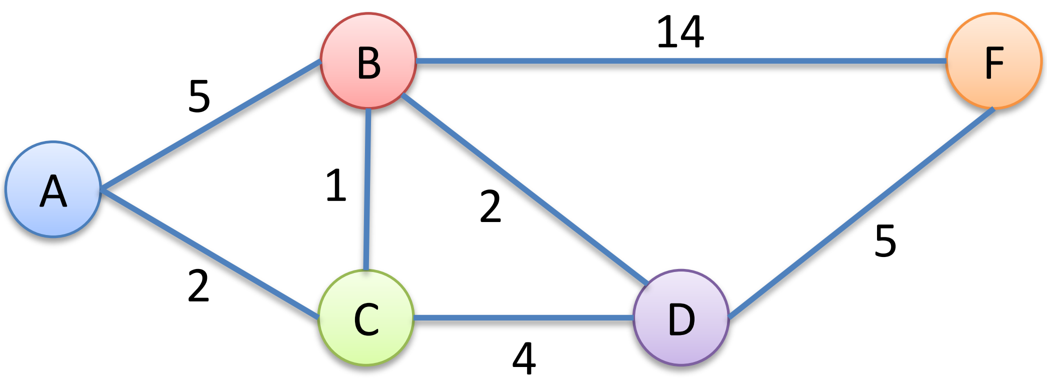
Router C

Via→ ↓ To	A	B	D
A	2	4?	
B	7	1	6
D		3	4
F		8	9

Router D

Via→ ↓ To	B	C	F
A	5	6	
B	2	5	19
C	3	4	
F	9?	13?	5

Distance Vector – Convergence



Eventually, we reach a converged state.

Router F

Via→ ↓ To	B	D
A	17	10
B	14	7
C	15	8
D	16	5

Router A

Via→ ↓ To	B	C
B	5	3
C	6	2
D	7	5
F	12	10

Router B

Via→ ↓ To	A	C	D	F
A	5	3	7	24
C	7	1	4	22
D	10	4	2	19
F	15	9	7	14

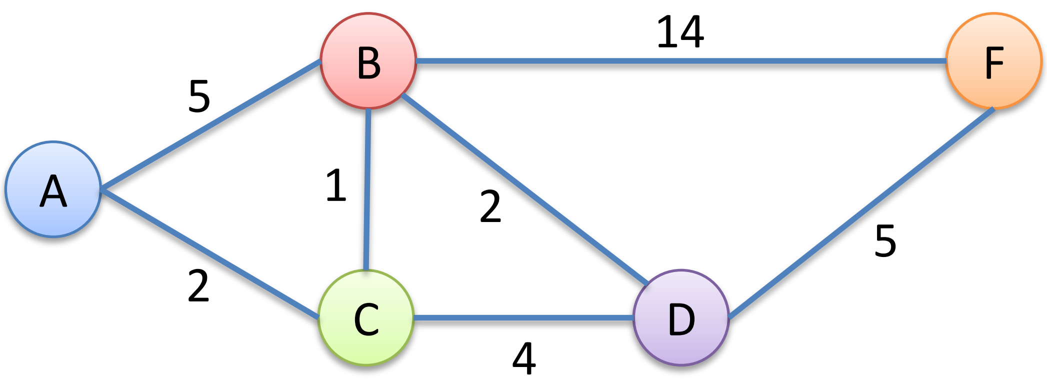
Router C

Via→ ↓ To	A	B	D
A	2	4	9
B	7	1	6
D	7	3	4
F	12	8	9

Router D

Via→ ↓ To	B	C	F
A	5	6	15
B	2	5	12
C	3	4	13
F	9	12	5

Distance Vector – Convergence



Router F

Via→ ↓ To	B	D
A	17	10
B	14	7
C	15	8
D	16	5

Final forwarding tables:

Router A

Via→ ↓ To	B	C
B	5	3
C	6	2
D	7	5
F	12	10

Router B

Via→ ↓ To	A	C	D	F
A	5	3	7	24
C	7	1	4	22
D	10	4	2	19
F	15	9	7	14

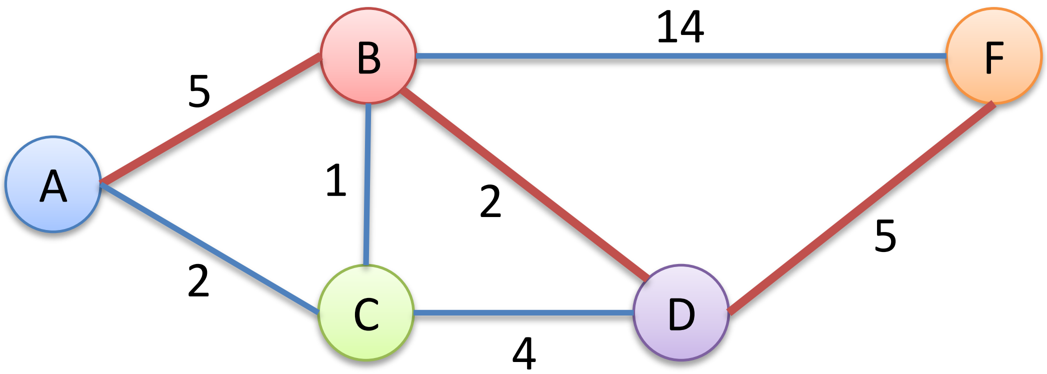
Router C

Via→ ↓ To	A	B	D
A	2	4	9
B	7	1	6
D	7	3	4
F	12	8	9

Router D

Via→ ↓ To	B	C	F
A	5	6	15
B	2	5	12
C	3	4	13
F	9	12	5

Of the links in red below, for how many would a failure cause a loop?



A – 0, B – 1, C – 2, D – 3

Consider the failures independently (not all at the same time).

Router F

Via→ ↓ To	B	D
A	17	10
B	14	7
C	15	8
D	16	5

Router A

Via→ ↓ To	B	C
B	5	3
C	6	2
D	7	5
F	12	10

Router B

Via→ ↓ To	A	C	D	F
A	5	3	7	24
C	7	1	4	22
D	10	4	2	19
F	15	9	7	14

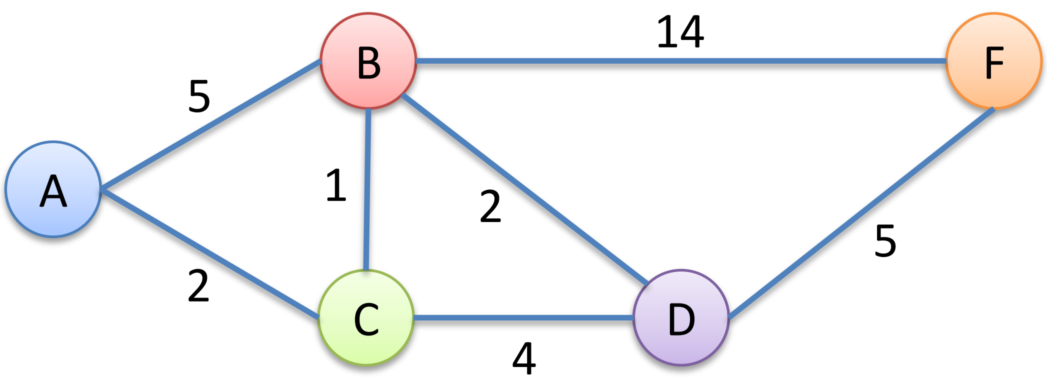
Router C

Via→ ↓ To	A	B	D
A	2	4	9
B	7	1	6
D	7	3	4
F	12	8	9

Router D

Via→ ↓ To	B	C	F
A	5	6	15
B	2	5	12
C	3	4	13
F	9	12	5

Rewind: Distance Vector – Round 2



B will send to neighbors (A, C, D, F):
I can get to A in 3, C in 1, D in 2, and F in 7.

Router F

Via→ ↓ To	B	D
A	17	
B	14	7
C	15	9
D	16	5

Router A

Via→ ↓ To	B	C
B	5	3
C	6	2
D	7	6
F	12	

Router B

Via→ ↓ To	A	C	D	F
A	5	3		
C	7	1	6	
D		5	2	19
F			7	14

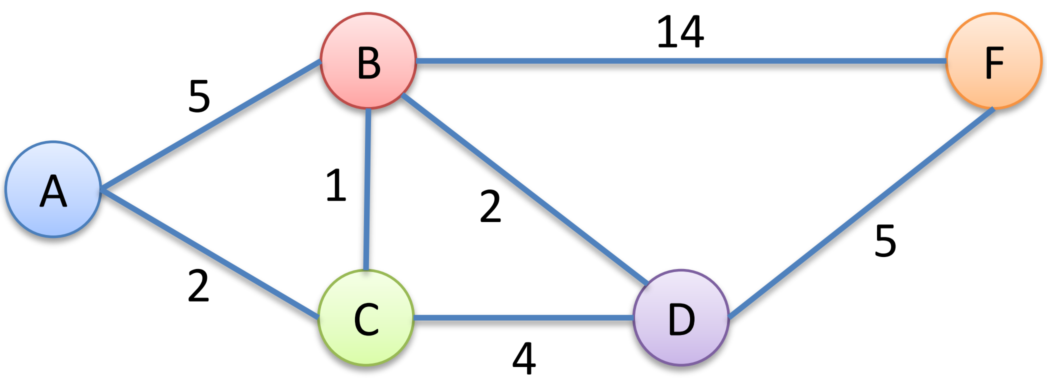
Router C

Via→ ↓ To	A	B	D
A	2	4?	
B	7	1	6
D		3	4
F		8	9

Router D

Via→ ↓ To	B	C	F
A	5	6	
B	2	5	19
C	3	4	
F	9?		5

Rewind: Distance Vector – Round 2



Router F

Via→ ↓ To	B	D
A	17	
B	14	7
C	15	9
D	16	5

Poisoned reverse: Don't advertise a lower value to a neighbor if you go through that neighbor to get there!

No! ∞

Router A

Via→ ↓ To	B	C
B	5	3
C	6	2
D	7	6
F	12	

Router B

Via→ ↓ To	A	C	D	F
A	5	3		
C	7	1	6	
D		5	2	19
F			7	14

Router C

Via→ ↓ To	A	B	D
A	2	~~4?~~	
B	7	1	6
D		3	4
F		8	9

Router D

Via→ ↓ To	B	C	F
A	5	6	
B	2	5	19
C	3	4	
F	9?		5

Loop-prevention

- Route poisoning helps prevent loops, but doesn't guarantee loop free.
- Other mechanisms help too
- <u>There will always be a window of vulnerability</u>

Summary

Link State

+ Fast convergence (reacts to events quickly)
+ Small window of inconsistency

- Large number of messages sent on events
- Large routing tables as network size grows

Distance Vector

+ Distributed (small tables)
+ No flooding (fewer messages)

– Slower convergence
– Larger window of inconsistency