COMP 3331/9331: Computer Networks and Applications

Week 10

Network Layer: Control Plane (Routing)

Chapter 5: Section 5.1 – 5.2

Network layer control plane

Goals: understand principles behind network control plane

- traditional interior routing algorithms

Network layer, control plane: outline

5.1 introduction

5.2 routing protocols

- link state
- distance vector

Network-layer functions

Recall: two network-layer functions:

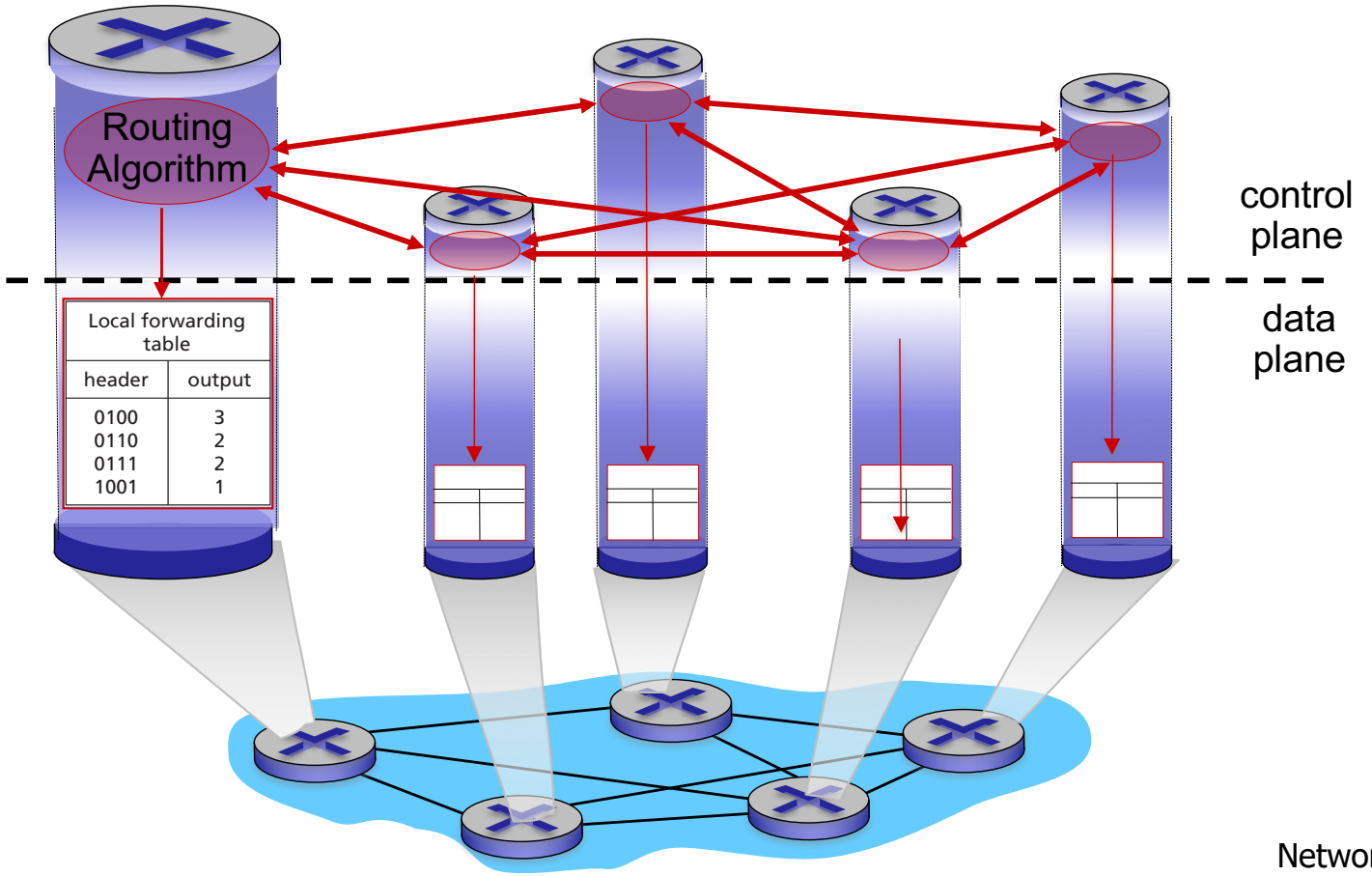
- *forwarding:* move packets from router's input to appropriate router output — *data plane*
- *routing:* determine route taken by packets from source to destination — *control plane*

Two approaches to structuring network control plane:

- per-router control (traditional)
- logically centralized control (software defined networking)

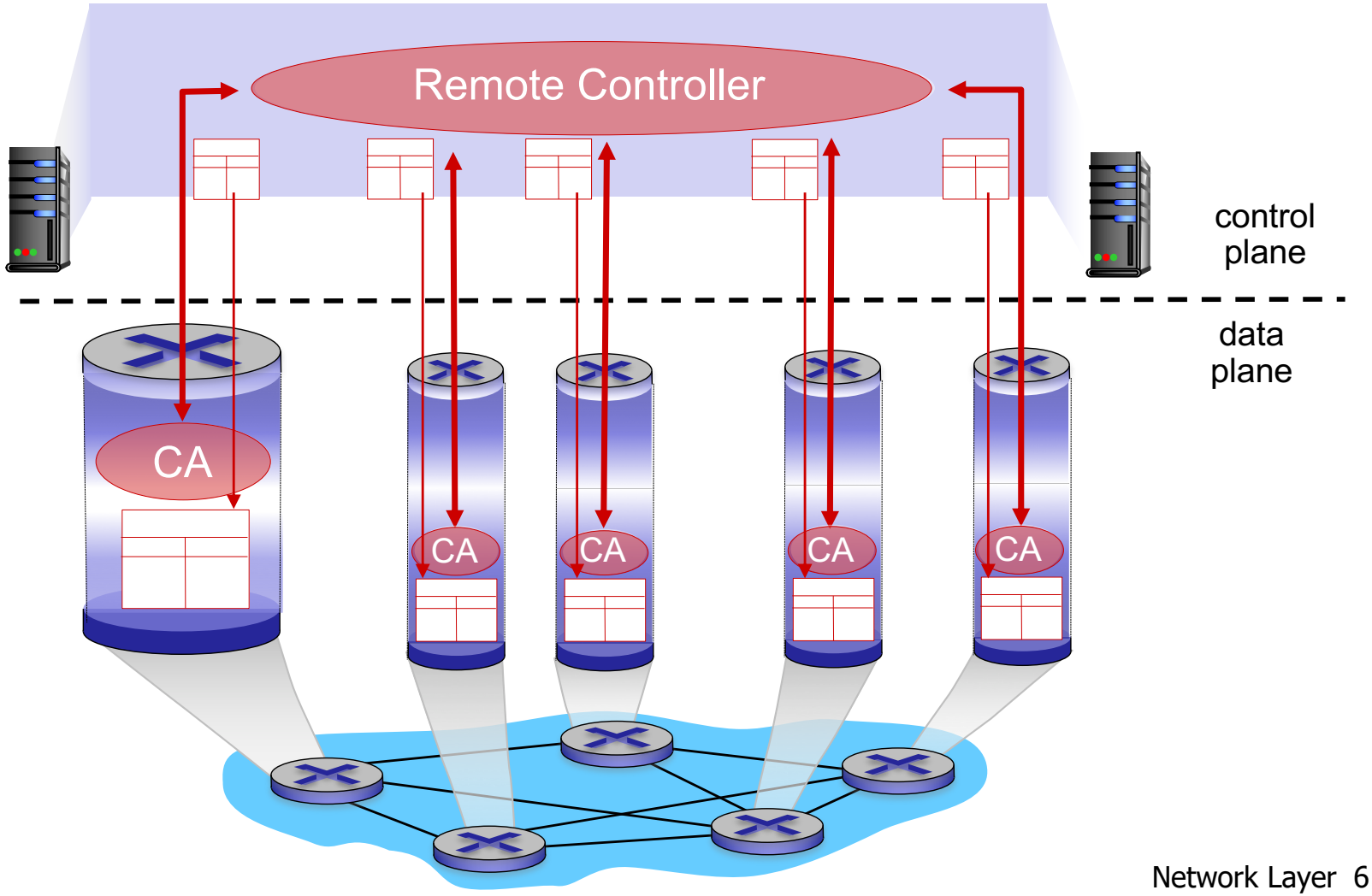
Per-router control plane

Individual routing algorithm components *in each and every router* interact with each other in control plane to compute forwarding tables



Logically centralized control plane

A distinct (typically remote) controller interacts with local control agents (CAs) in routers to compute forwarding tables



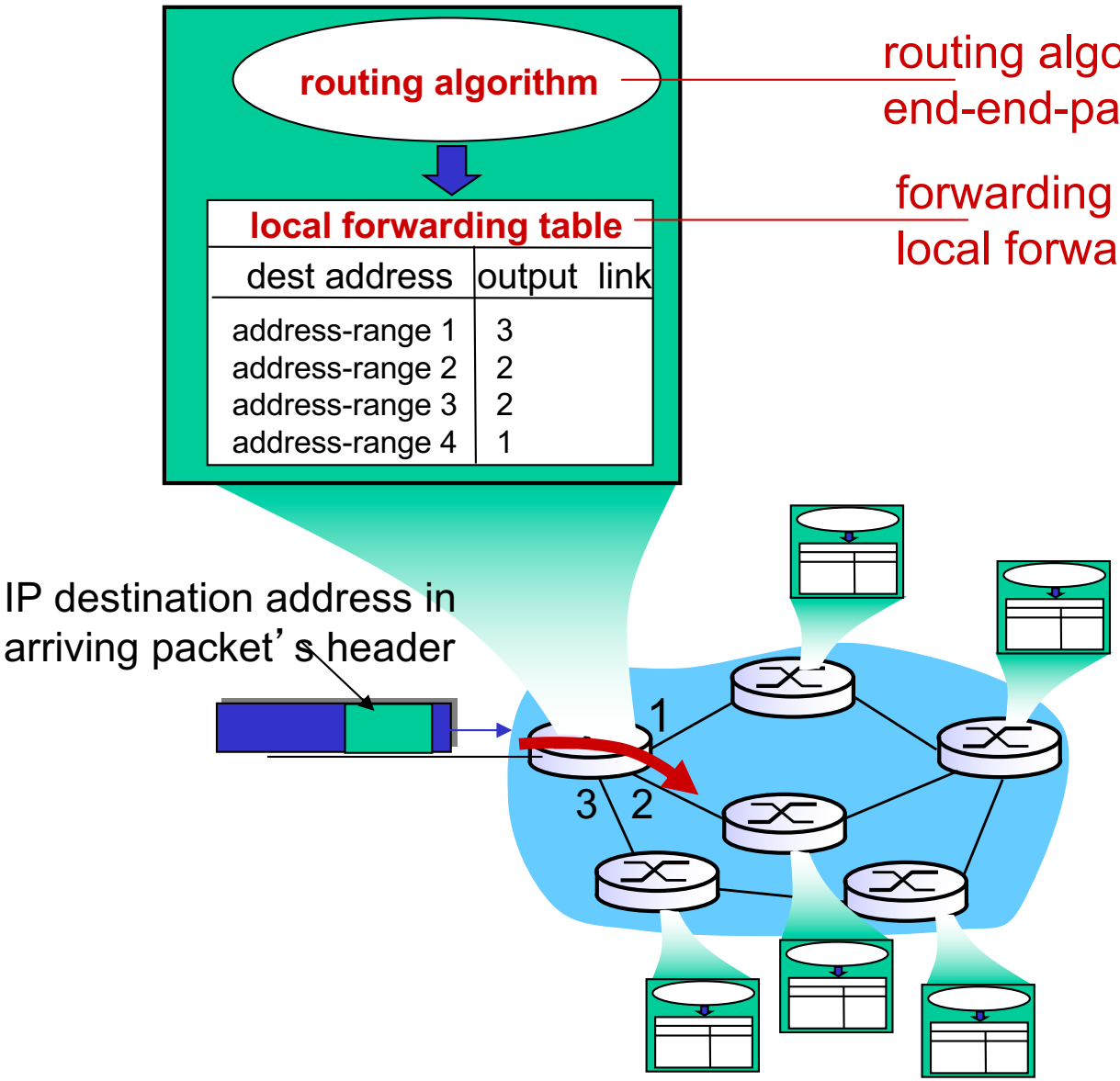
Network layer, control plane: outline

5.1 introduction

5.2 routing protocols

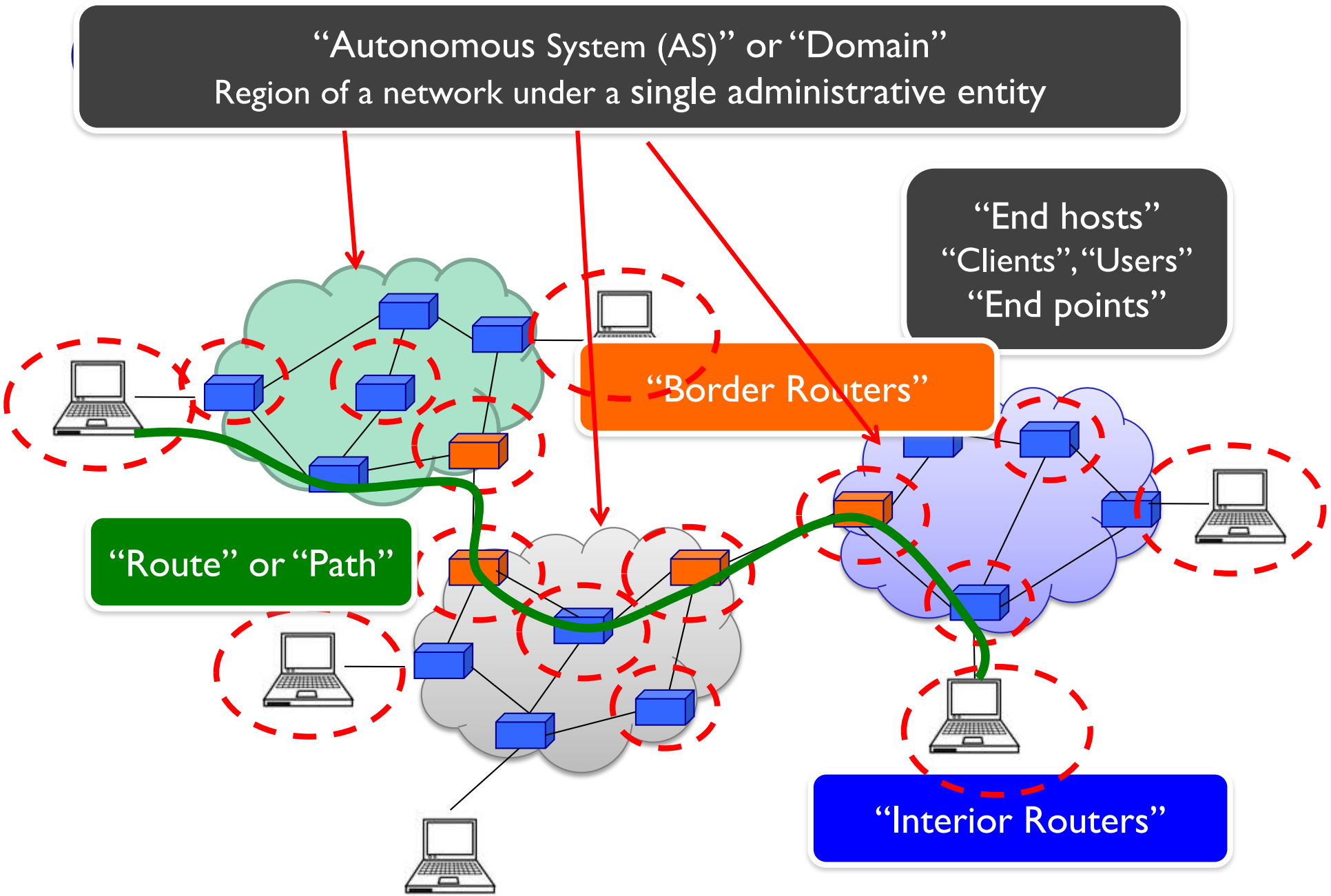
- link state
- distance vector

Interplay between routing, forwarding



routing algorithm determines
end-end-path through network

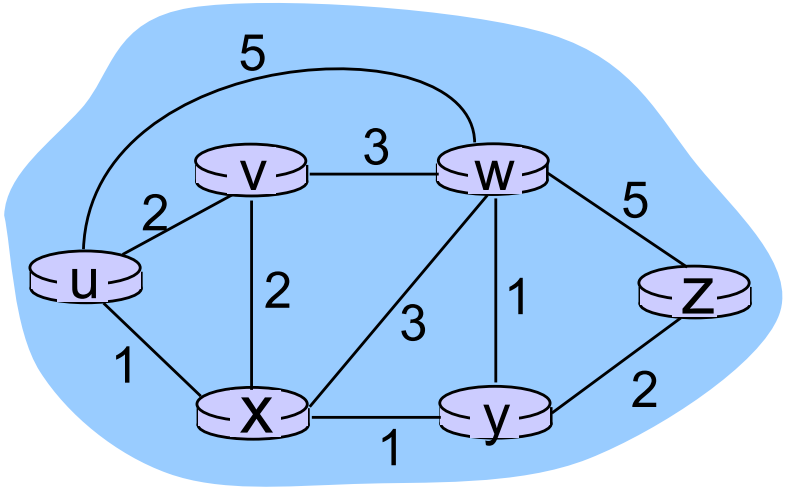
forwarding table determines
local forwarding at this router


"Autonomous System (AS)" or "Domain"
Region of a network under a single administrative entity
"End hosts"
"Clients", "Users"
"End points"
"Border Routers"
"Route" or "Path"
"Interior Routers"

Internet Routing

- Internet Routing works at two levels
- Each AS runs an intra-domain routing protocol that establishes routes within its domain [covered in this course]
  - (AS -- region of network under a single administrative entity)
  - Link State, e.g., Open Shortest Path First (OSPF)
  - Distance Vector, e.g., Routing Information Protocol (RIP)
- ASes participate in an inter-domain routing protocol that establishes routes between domains [not covered]
  - Path Vector, e.g., Border Gateway Protocol (BGP)

Graph abstraction



graph: G = (N,E)

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

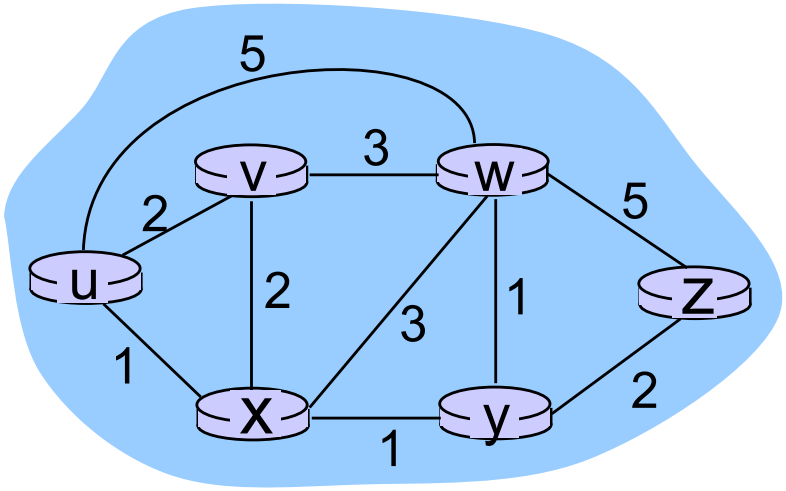
E = set of links ={ (u,v), (u,x), (v,x), (v,w), (x,w), (x,y), (w,y), (w,z), (y,z) }

aside: graph abstraction is useful in other network contexts, e.g., P2P, where *N* is set of peers and *E* is set of TCP connections

Addressing (abstraction)

- Assume each host has a unique ID (address)
- No particular structure to those IDs
- Earlier in course we talked about real IP addressing

Graph abstraction: costs



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

Quiz: 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 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 (Decentralised)

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

Network layer, control plane: outline

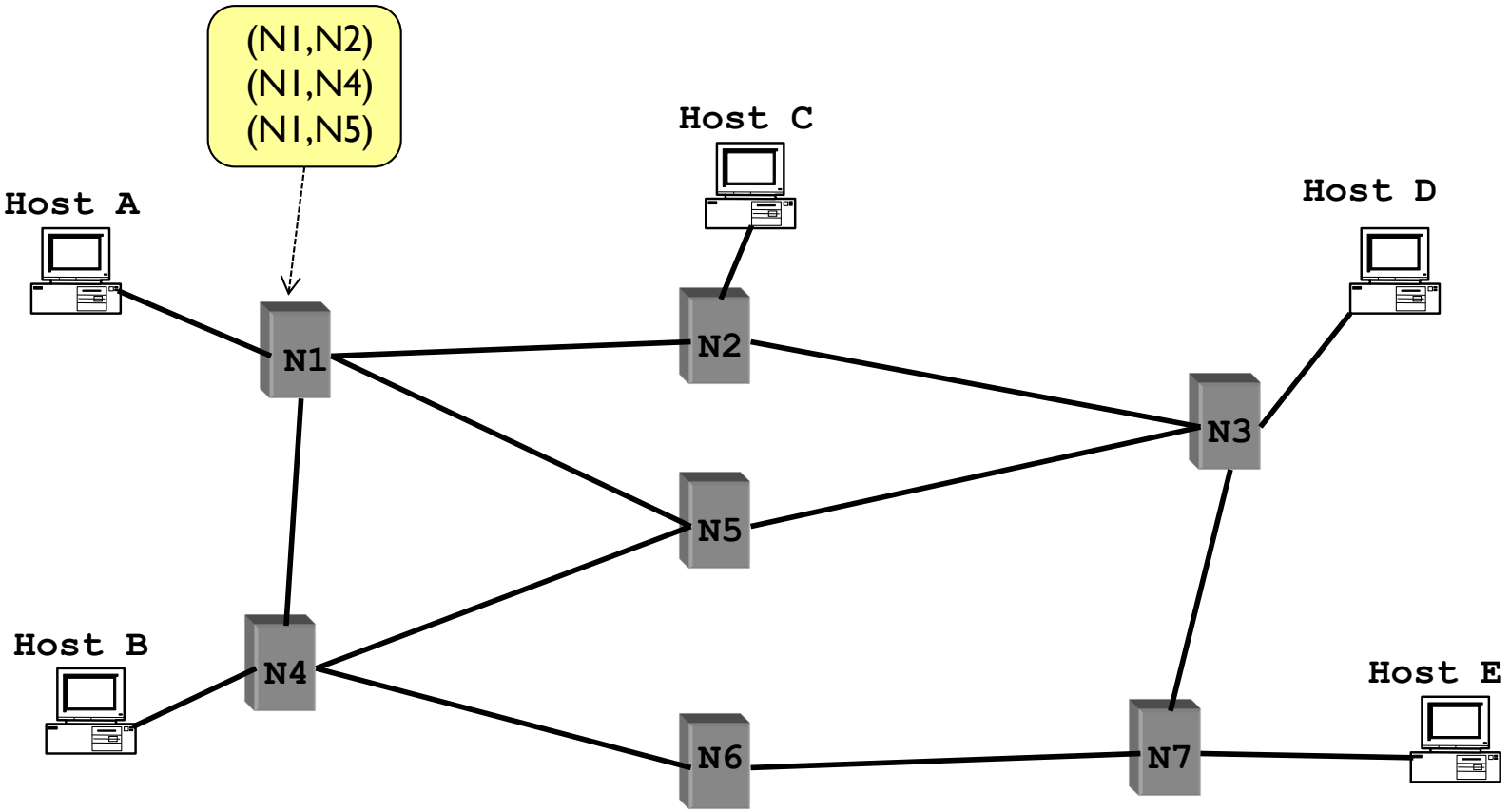
5.1 introduction

5.2 routing protocols

- link state
- distance vector

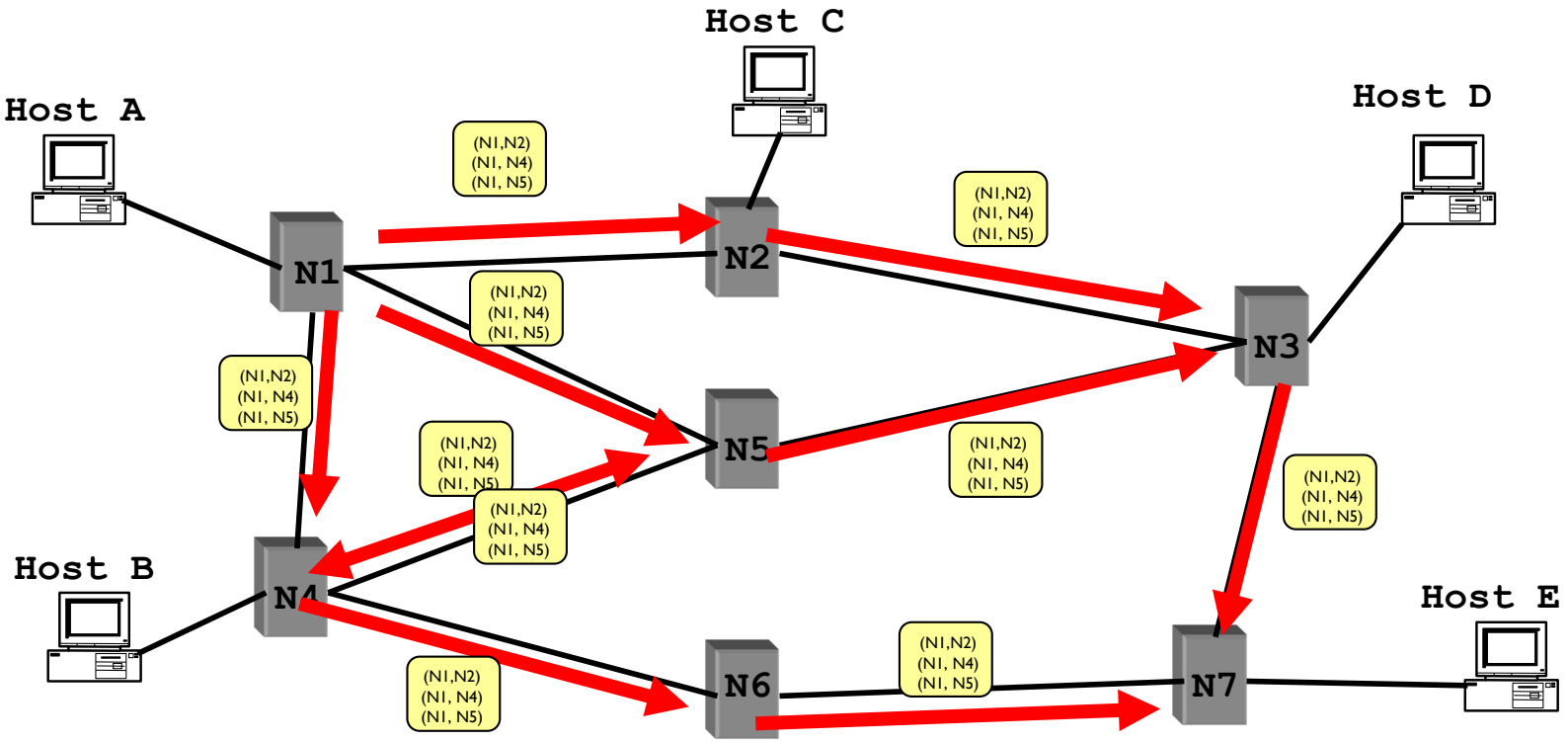
Link State Routing

- Each node maintains its local "link state" (LS)
  - i.e., a list of its directly attached links and their costs



Link State Routing

- Each node maintains its local "link state" (LS)
- Each node floods its local link state
  - on receiving a new LS message, a router forwards the message to all its neighbors other than the one it received the message from

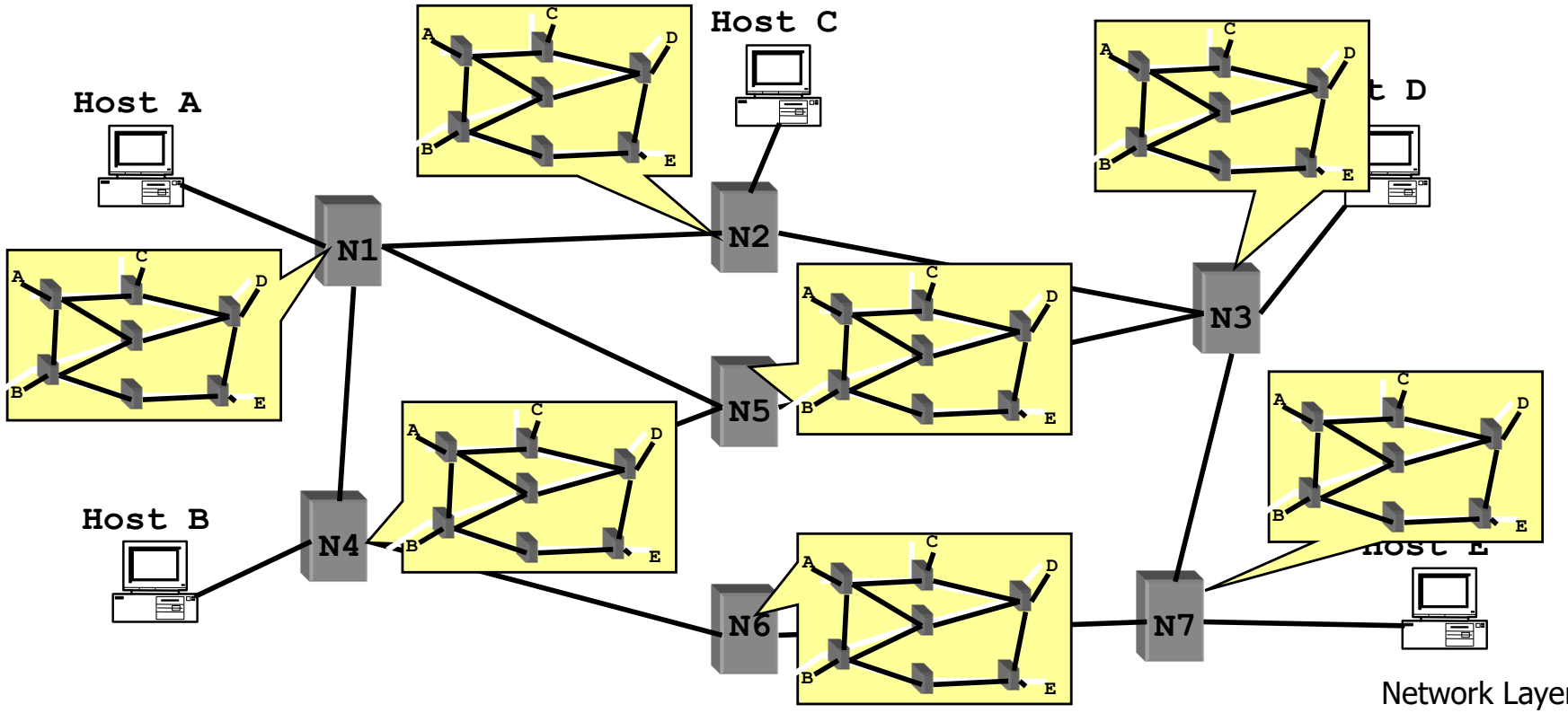


Flooding LSAs

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

Link State Routing

- Each node maintains its local “link state” (LS)
- Each node floods its local link state
- Eventually, each node learns the entire network topology
  - Can use Dijkstra’s to compute the shortest paths between nodes



A Link-State Routing Algorithm

Dijkstra's algorithm

- net topology, link costs known to all nodes
  - accomplished via "link state broadcast"
  - all nodes have same info
- computes least cost paths from one node ("source") to all other nodes
  - gives *forwarding table* for that node
- iterative: after k iterations, know least cost path to k dest.'s

notation:

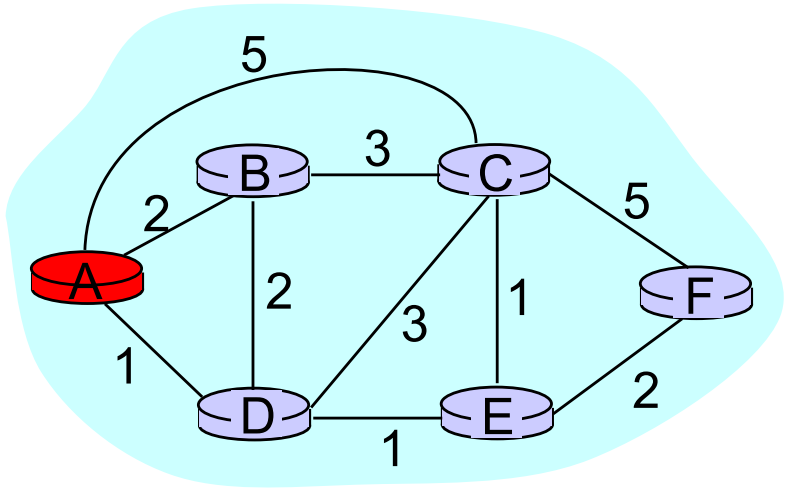
- c(x,y): link cost from node x to y; = ∞ if not direct neighbors
- D(v): current value of cost of path from source to dest. v
- p(v): predecessor node along path from source to v
- N': set of nodes whose least cost path definitively known

Dijsktra's Algorithm

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

Example: Dijkstra's Algorithm

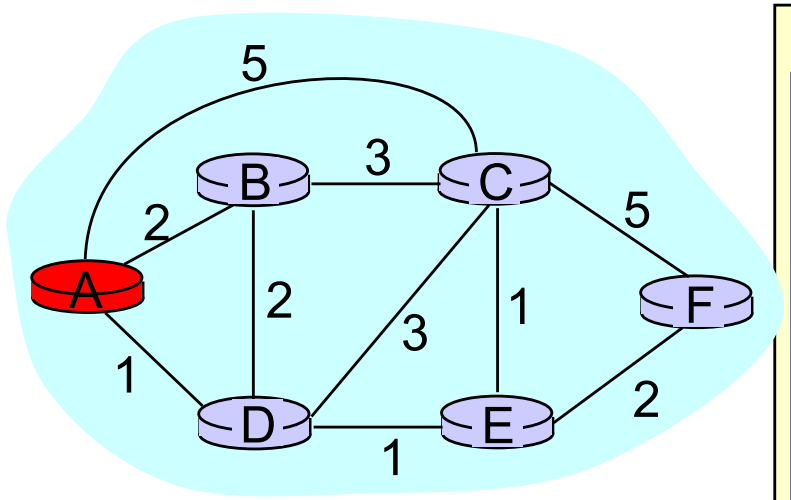
Step	Set N'	D(B),p(B)	D(C),p(C)	D(D),p(D)	D(E),p(E)	D(F),p(F)
0	A	2,A	5,A	1,A	∞	∞
1						
2						
3						
4						
5						




```
1  Initialization:
2    N' = {A};
3    for all nodes v
4      if v adjacent to A
5        then D(v) = c(A,v);
6        else D(v) = ∞;
…
```

Example: Dijkstra's Algorithm

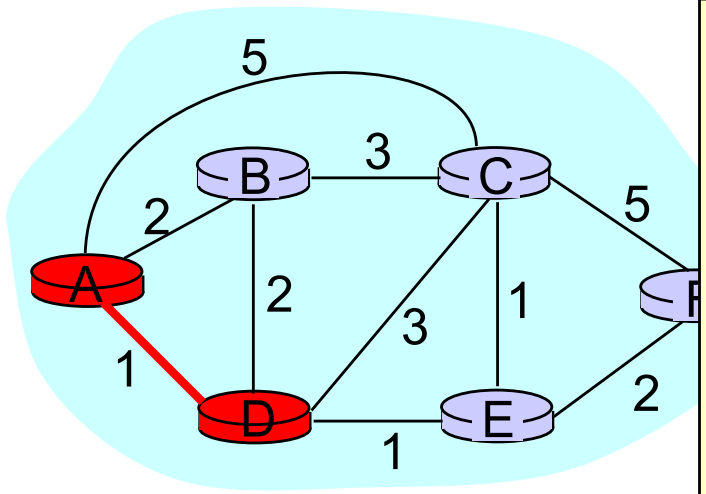
Step	Set N'	D(B),p(B)	D(C),p(C)	D(D),p(D)	D(E),p(E)	D(F),p(F)
0	A	2,A	5,A	1,A	∞	∞
1						
2						
3						
4						
5						




```
...
8   Loop
9     find w not in N' s.t. D(w) is a minimum;
10    add w to N';
11    update D(v) for all v adjacent
        to w and not in N':
12    If D(w) + c(w,v) < D(v) then
13      D(v) = D(w) + c(w,v); p(v) = w;
14    until all nodes in N';
```

Example: Dijkstra's Algorithm

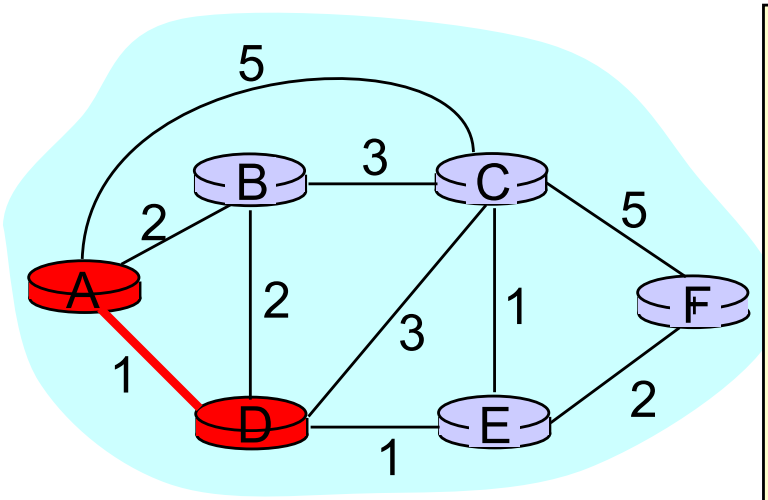
Step	Set N'	D(B),p(B)	D(C),p(C)	D(D),p(D)	D(E),p(E)	D(F),p(F)
0	A	2,A	5,A	1,A	∞	∞
1	AD					
2						
3						
4						
5						




```
...
8   Loop
9     find w not in N' s.t. D(w) is a minimum;
10    add w to N';
11    update D(v) for all v adjacent
        to w and not in N':
12    If D(w) + c(w,v) < D(v) then
13      D(v) = D(w) + c(w,v); p(v) = w;
14    until all nodes in N';
```

Example: Dijkstra's Algorithm

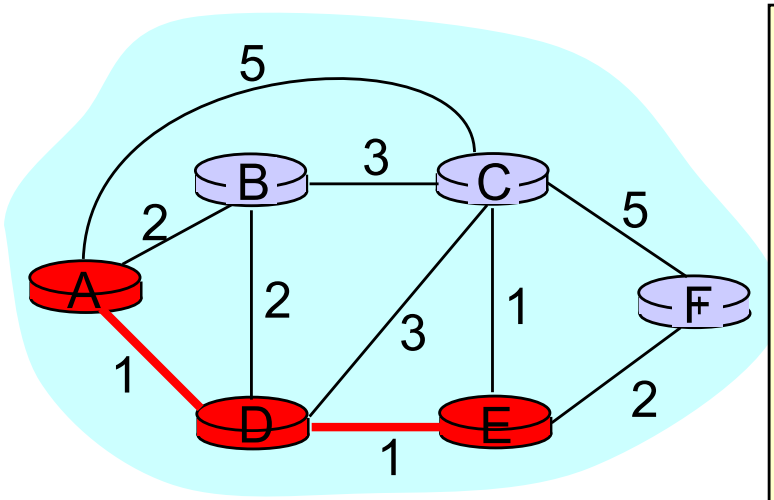
Step	Set N'	D(B),p(B)	D(C),p(C)	D(D),p(D)	D(E),p(E)	D(F),p(F)
0	A	2,A	5,A	1,A	∞	∞
1	AD		4,D		2,D	
2						
3						
4						
5						




```
…
8   Loop
9     find w not in N' s.t. D(w) is a minimum;
10    add w to N';
11    update D(v) for all v adjacent
        to w and not in N':
12    If D(w) + c(w,v) < D(v) then
13      D(v) = D(w) + c(w,v); p(v) = w;
14  until all nodes in N';
```

Example: Dijkstra's Algorithm

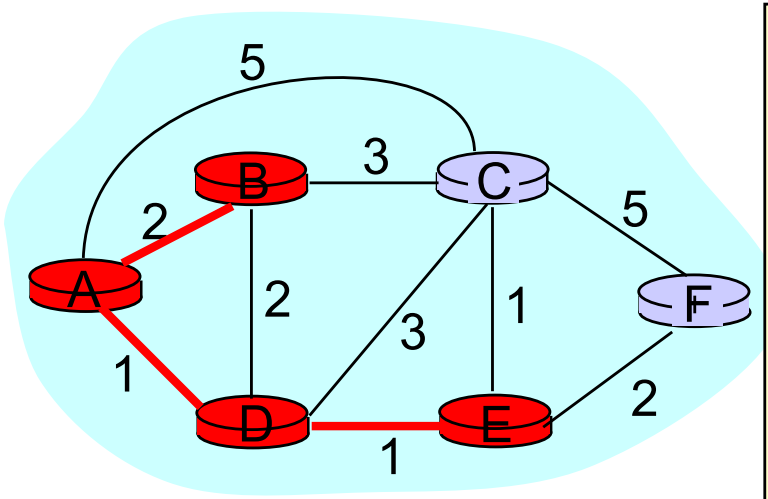
Step	Set N'	D(B),p(B)	D(C),p(C)	D(D),p(D)	D(E),p(E)	D(F),p(F)
0	A	2,A	5,A	1,A	∞	∞
1	AD		4,D		2,D	
→ 2	ADE		3,E			4,E
3						
4						
5						




```
...
8   Loop
9     find w not in N' s.t. D(w) is a minimum;
10    add w to N';
11    update D(v) for all v adjacent
        to w and not in N':
12    If D(w) + c(w,v) < D(v) then
13      D(v) = D(w) + c(w,v); p(v) = w;
14    until all nodes in N';
```

Example: Dijkstra's Algorithm

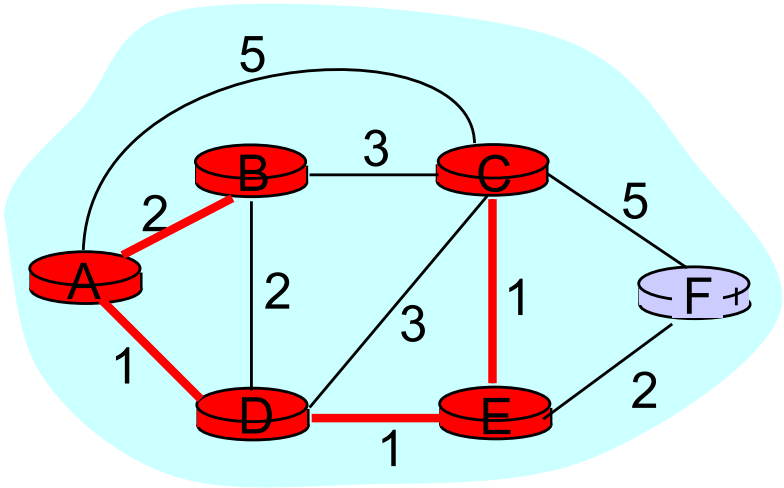
Step	Set N'	D(B),p(B)	D(C),p(C)	D(D),p(D)	D(E),p(E)	D(F),p(F)
0	A	2,A	5,A	1,A	∞	∞
1	AD		4,D		2,D	
2	ADE		3,E			4,E
→ 3	ADEB					
4						
5						




```
...
8   Loop
9     find w not in N' s.t. D(w) is a minimum;
10    add w to N';
11    update D(v) for all v adjacent
        to w and not in N':
12    If D(w) + c(w,v) < D(v) then
13      D(v) = D(w) + c(w,v); p(v) = w;
14    until all nodes in N';
```

Example: Dijkstra's Algorithm

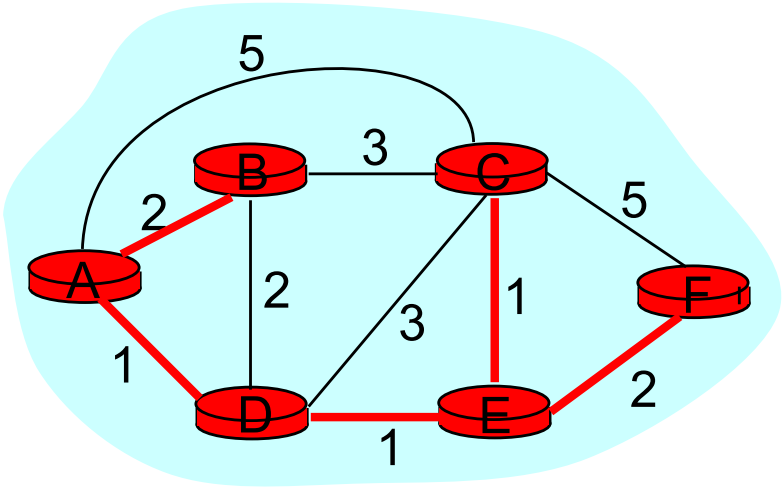
Step	Set N'	D(B),p(B)	D(C),p(C)	D(D),p(D)	D(E),p(E)	D(F),p(F)
0	A	2,A	5,A	1,A	∞	∞
1	AD		4,D		2,D	
2	ADE		3,E			4,E
3	ADEB					
4	ADEBC					
5						




```
…
8   Loop
9     find w not in N' s.t. D(w) is a minimum;
10    add w to N';
11    update D(v) for all v adjacent
        to w and not in N':
12    If D(w) + c(w,v) < D(v) then
13      D(v) = D(w) + c(w,v); p(v) = w;
14    until all nodes in N';
```

Example: Dijkstra's Algorithm

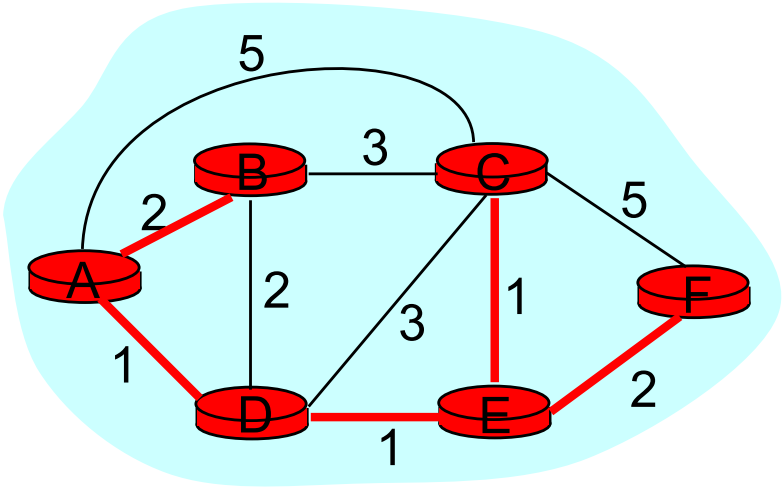
Step	Set N'	D(B),p(B)	D(C),p(C)	D(D),p(D)	D(E),p(E)	D(F),p(F)
0	A	2,A	5,A	1,A	∞	∞
1	AD		4,D		2,D	
2	ADE		3,E			4,E
3	ADEB					
4	ADEBC					
5	ADEBCF					




```
...
8   Loop
9     find w not in N' s.t. D(w) is a minimum;
10    add w to N';
11    update D(v) for all v adjacent
        to w and not in N':
12    If D(w) + c(w,v) < D(v) then
13      D(v) = D(w) + c(w,v); p(v) = w;
14    until all nodes in N';
```

Example: Dijkstra's Algorithm

Step	Set 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		4,D		2,D	
2	ADE		3,E			4,E
3	ADEB					
4	ADEBC					
5	ADEBCF					

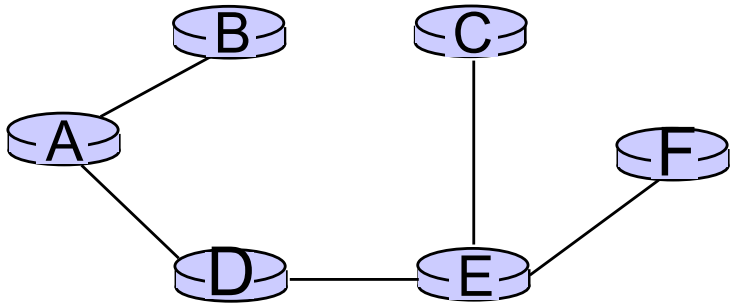


To determine path A → C (say), work backward from C via p(v)

The Forwarding Table

- Running Dijkstra at node A gives the shortest path from A to all destinations
- We then construct the *forwarding table*

resulting shortest-path tree from A:



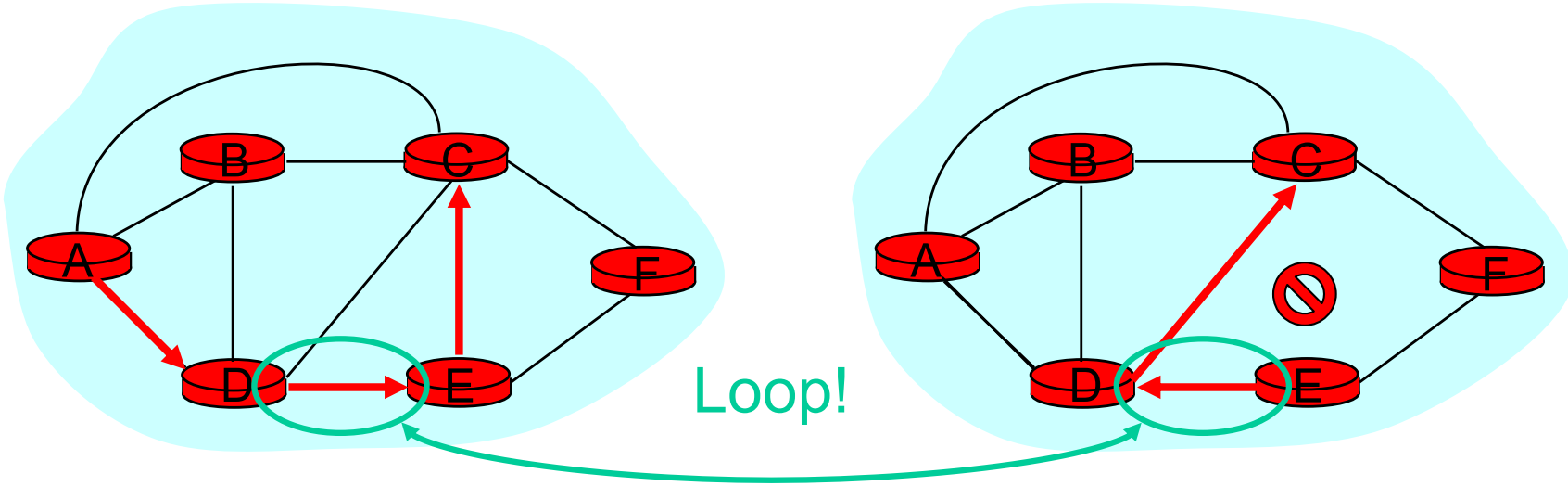
Destination	Link
B	(A,B)
C	(A,D)
D	(A,D)
E	(A,D)
F	(A,D)

Issue #1: Scalability

- How many messages needed to flood link state messages?
  - O(N x E), where N is #nodes; E is #edges in graph
- Processing complexity for Dijkstra's algorithm?
  - $O(N^2)$, because we check all nodes w not in S at each iteration and we have O(N) iterations
  - more efficient implementations: O(N log(N) +E) using min-heap
- How many entries in the LS topology database? O(E)
- How many entries in the forwarding table? O(N)

Issue#2: Transient Disruptions

- Inconsistent link-state database
  - Some routers know about failure before others
  - The shortest paths are no longer consistent
  - Can cause transient forwarding loops



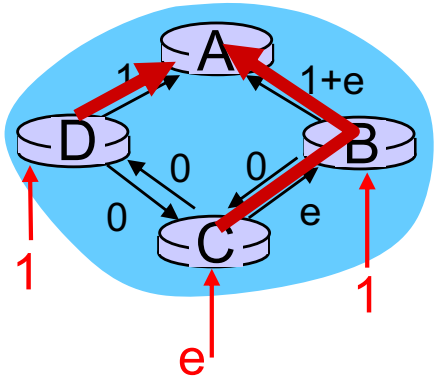
A and D think that this is the path to C

E thinks that this is the path to C

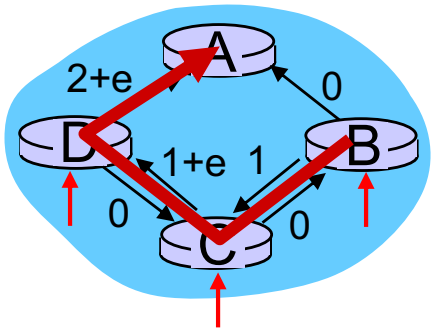
Oscillations

oscillations possible:

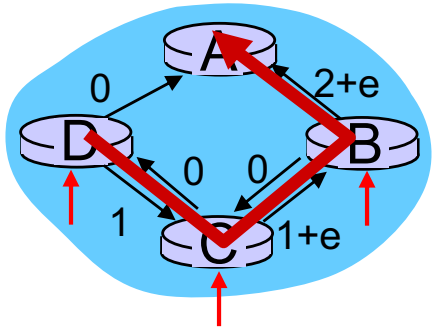
- e.g., suppose link cost equals amount of carried traffic:



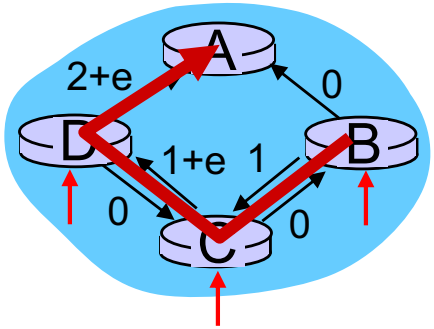
initially



given these costs,
find new routing….
resulting in new costs



given these costs,
find new routing….
resulting in new costs



given these costs,
find new routing….
resulting in new costs

Network layer, control plane: outline

5.1 introduction

5.2 routing protocols

- link state
- distance vector (in 2nd set of lecture notes)