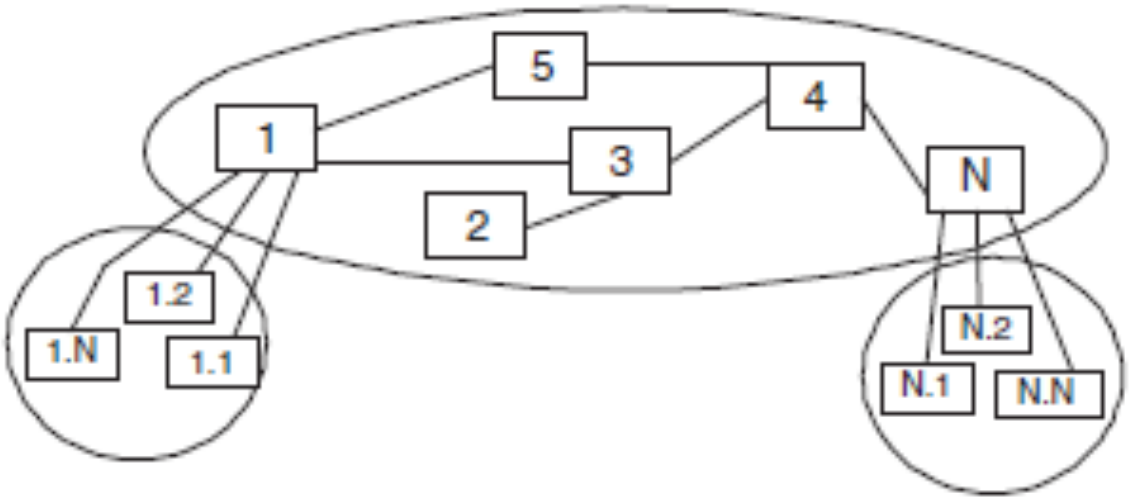
# ECE 158A: Lecture 5

Fall 2015

# Routing (1)

# Location-Based Addressing

- Recall from Lecture 1 that routers maintain routing tables to forward packets based on their IP addresses
- To allow scalability, IP addresses are assigned based on location, similarly to phone numbers [area,zone,number]

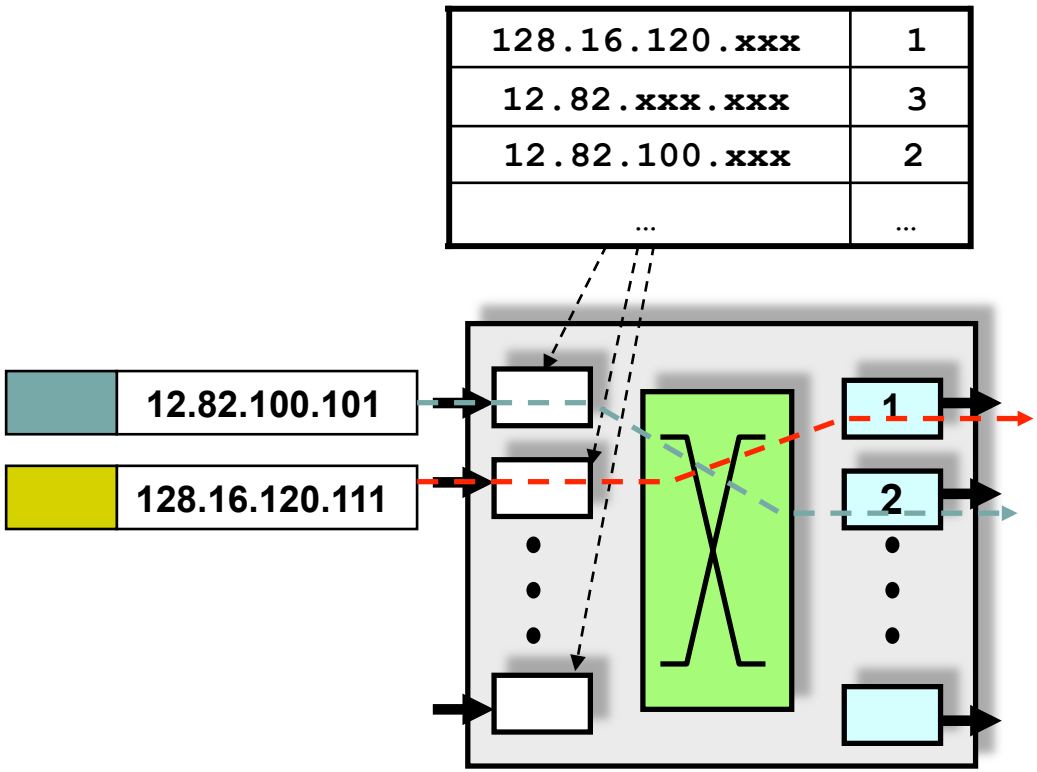


# Location-Based Addressing

- Location-based addressing helps reducing the number of entries in routing tables:
  - IP addresses with the same prefix can share the same entry in a routing table
- A routing table maps prefixes of input IP addresses to output ports
- Given a destination IP address, the router finds the longest matching prefix in the routing table

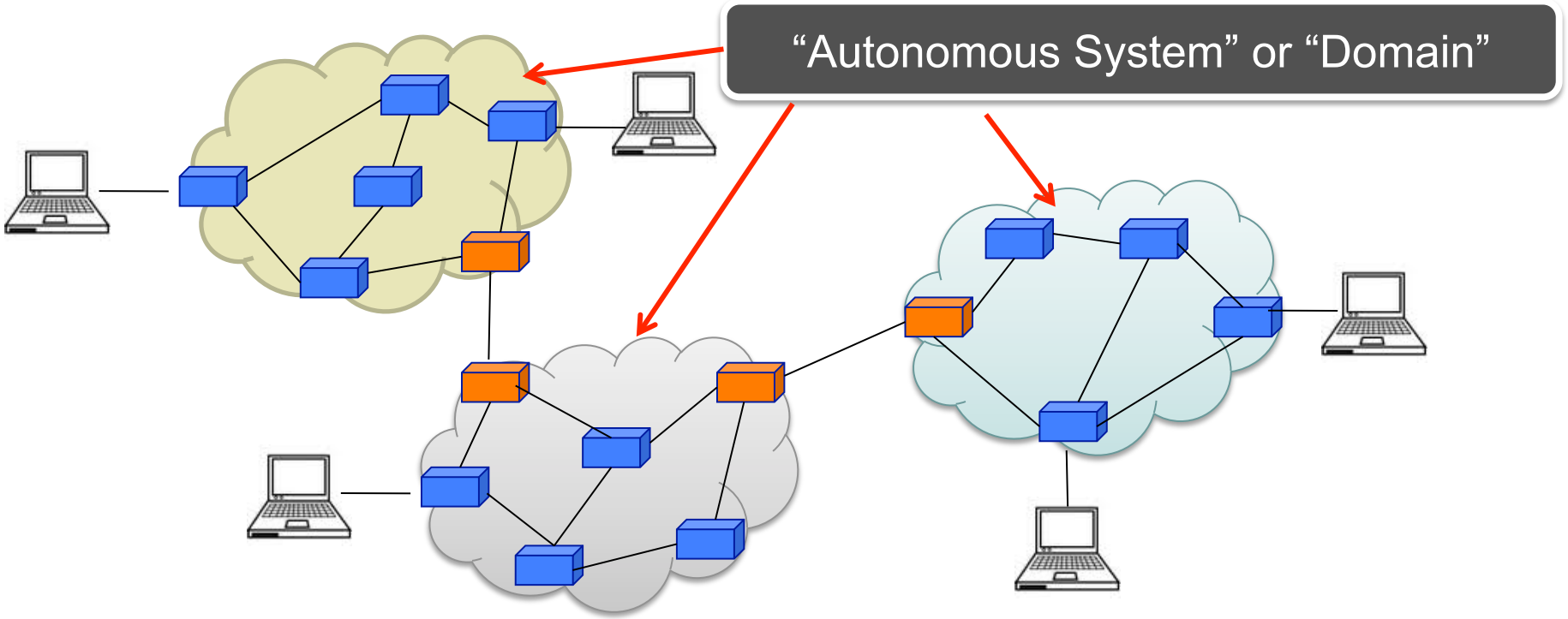
# Example

- Packet with destination address 12.82.100.101 is sent to output port 2, as 12.82.100.xxx is the longest prefix matching packet's destination address



# Autonomous Systems (AS)

- The Internet is grouped into about 40,000 Autonomous Systems (AS) or “domains”
  - Groups of hosts/routers under a single administrative entity
- Each AS is assigned a unique identifier
  - 16 bit AS Number (ASN)



# Two-Level Routing

- Routing is performed at two levels:
  - Routing within a domain is called intra-domain routing
  - Routing across domains is called inter-domain routing
- Reasons:
  - Routing within a domain requires microscopic knowledge of local network topology
  - Domains are managed by different companies/institutions
  - Intra-domain routing driven by optimal performance
  - Inter-domain driven by economic trade-offs

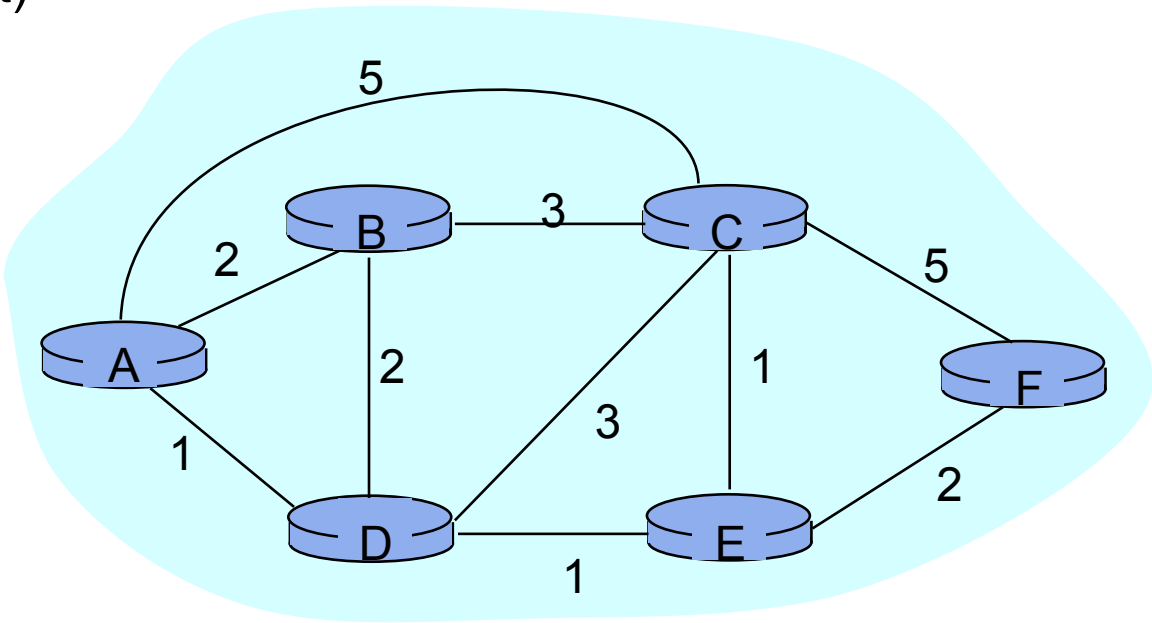
# Routing Protocols

- Internet routing protocols are responsible for constructing and updating the routing tables at the routers
- Routing protocols use shortest path algorithms to establish “good” paths between nodes. Two classical algorithms:
  - Link State: Based on Dijkstra’s Shortest Path Algorithm
  - Distance Vector: Based on Bellman-Ford Algorithm

- Routing Protocols used in practice:
  - Intra-domain routing: Open Shortest Path First (OSPF) (link state), Routing Information Protocol (RIP) (distance vector)
  - Inter-Domain routing: Border Gateway Protocol (BGP)

# Network Representation as a Graph

- Routing algorithms model the network as a graph
  - Routers are graph vertices and links are edges
  - Edges have an associated "cost" (e.g., distance, queue delay, cost)




- Goal: Find a least-cost path from any two vertices

# Link State vs Distance Vector

- Link State (LS):
  - Each node learns the complete network map (global information)
  - Each node computes shortest paths independently and in parallel

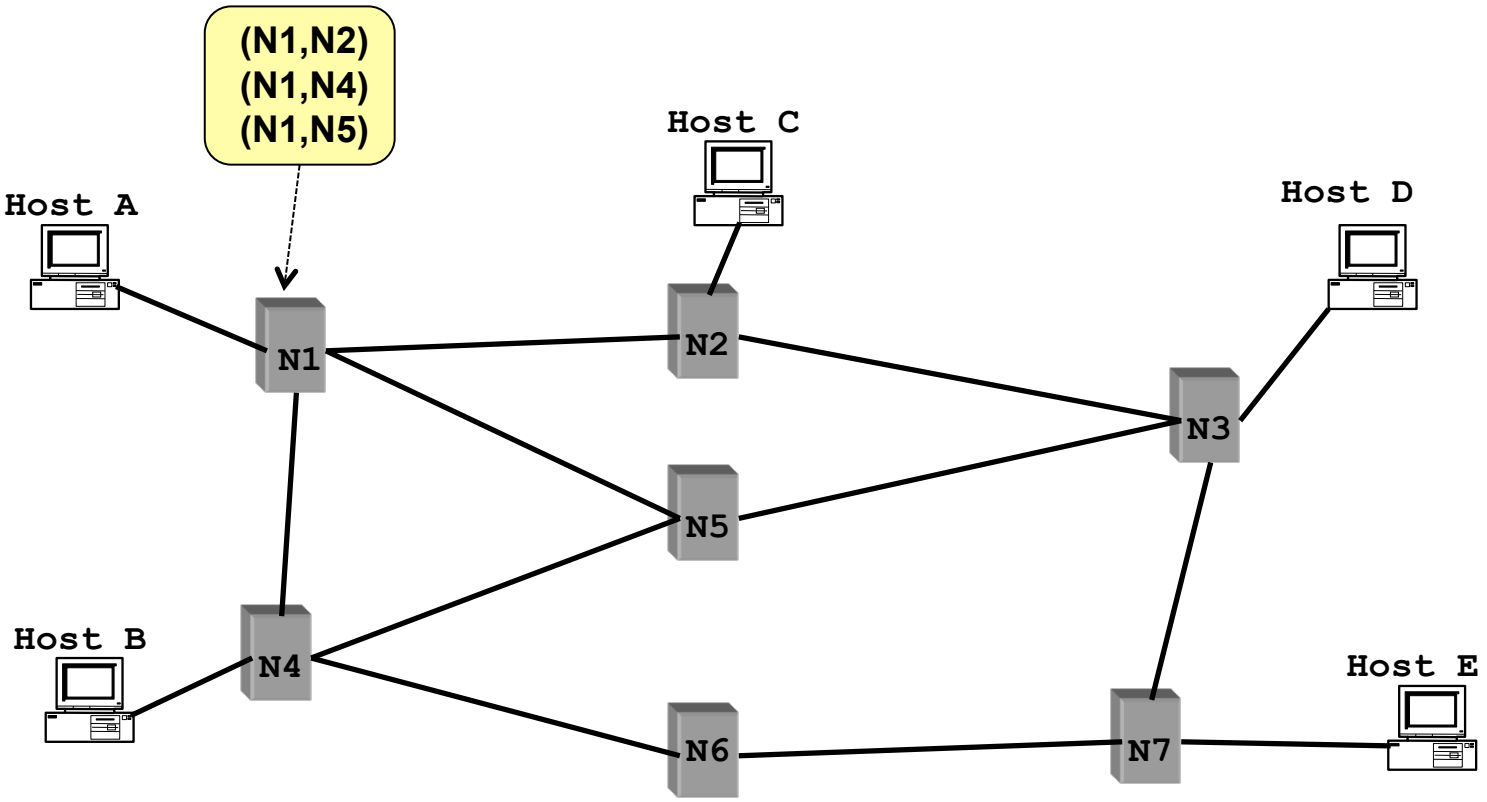
- Distance Vector (DV):
  - No node has the complete picture (local information)
  - Nodes cooperate to compute shortest paths in a distributed manner

- LS uses global information, while DV is asynchronous, and distributed.
- LS has higher messaging overhead and higher processing complexity, but is less vulnerable to looping

# Link-State

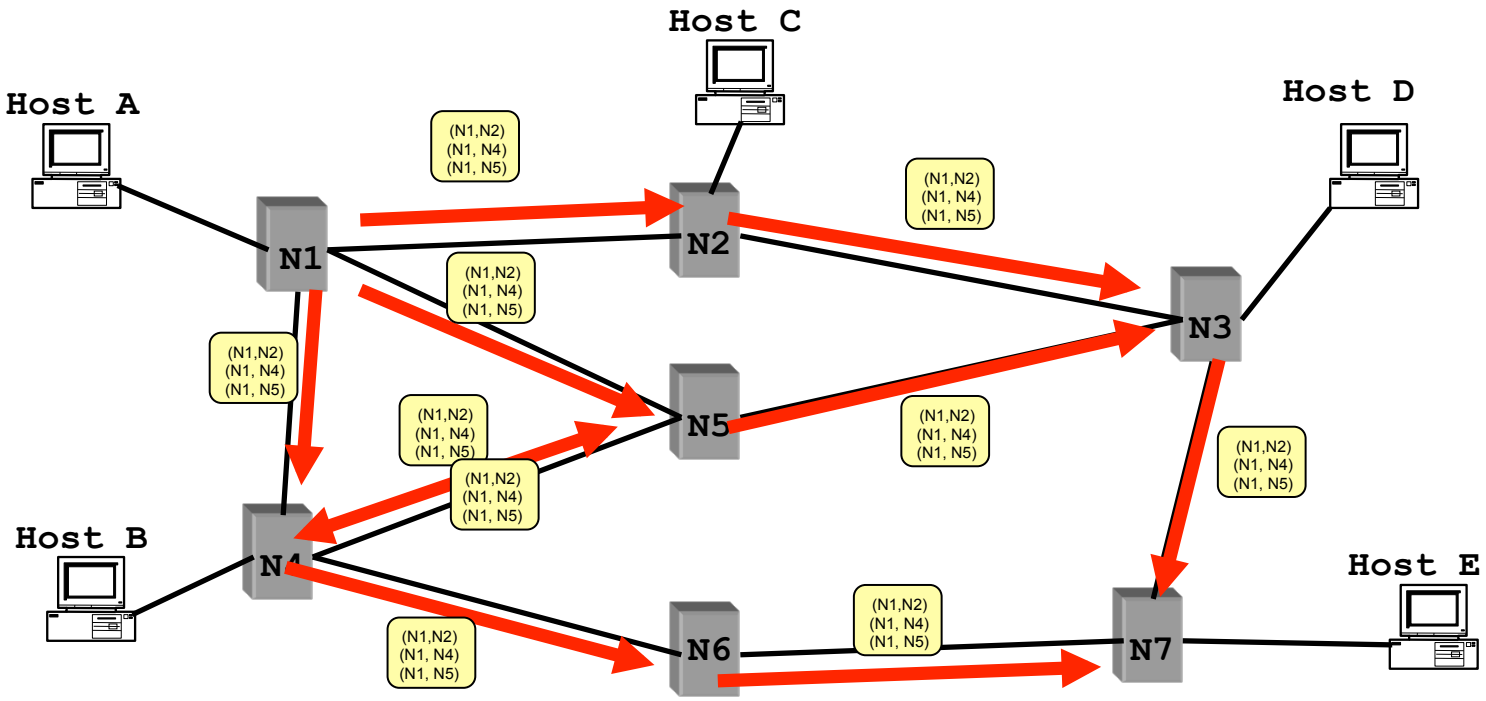
# Link State Routing

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



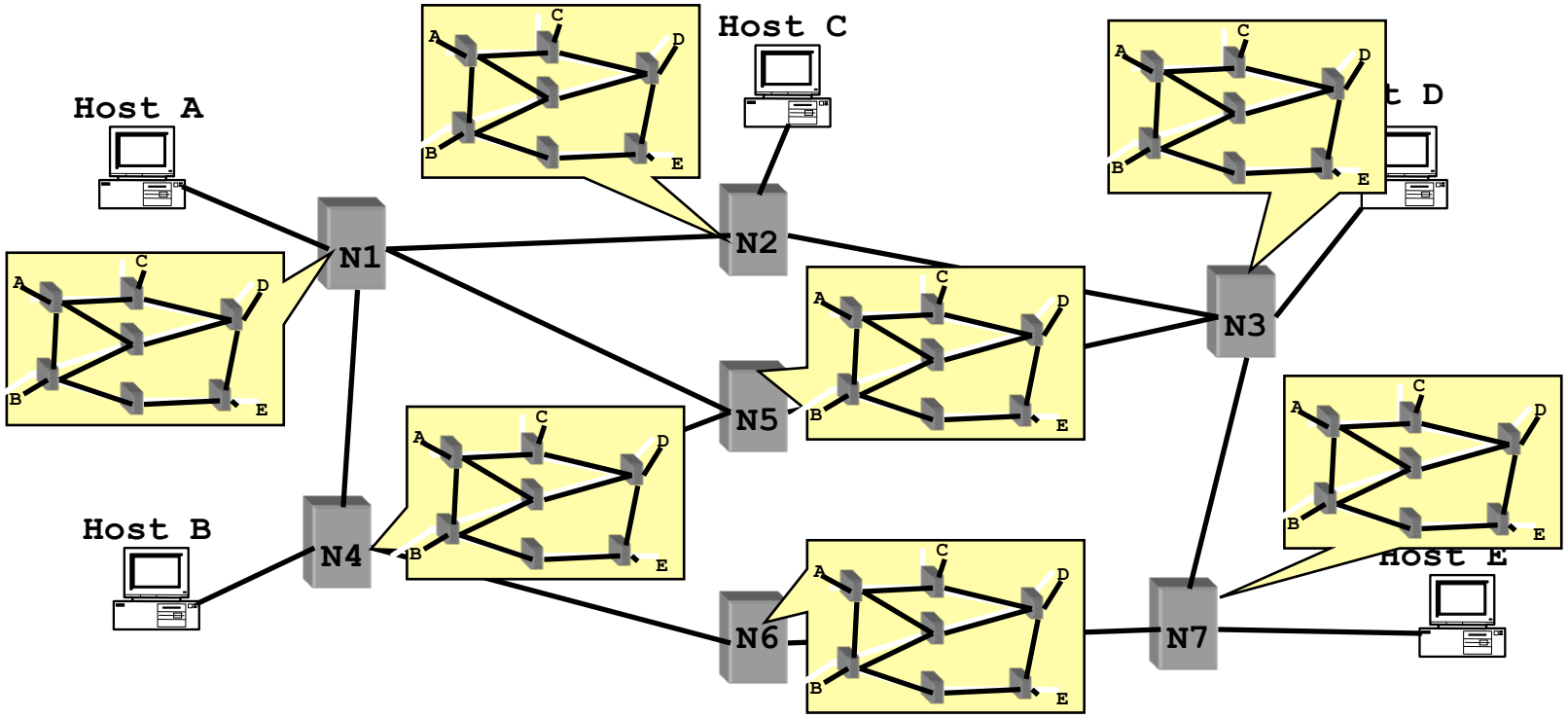
# Link State Routing

- 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



# Link State Routing

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



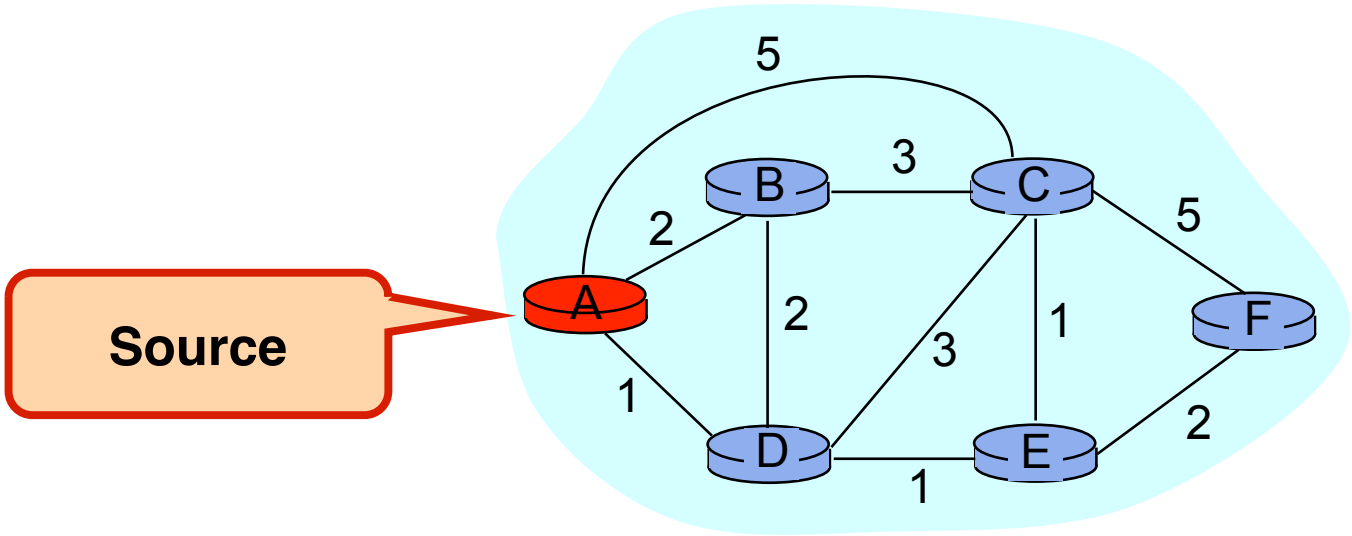
# Dijkstra's Shortest Path Algorithm

- INPUT:
  - Network topology (graph), with link costs
- OUTPUT:
  - Least cost paths from one node to all other nodes
- Iterative: after *k* iterations, a node knows the least cost path to its *k* closest neighbors

https://en.wikipedia.org/wiki/Edsger_W._Dijkstra

# Notation

- c(i,j): cost ≥ 0 from node i to j (∞ if i and j are not neighbors)
- D(v): total cost of the current least-cost path from source to destination v
- p(v): v's predecessor along path from source to v
- S: set of nodes whose least cost path is known



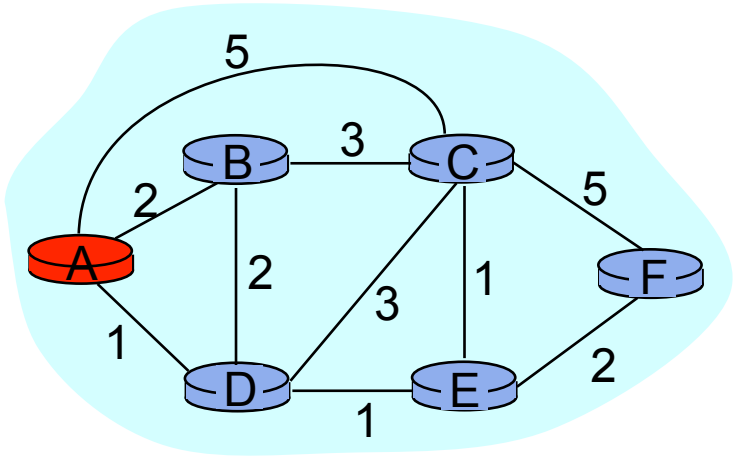
# Dijkstra's Algorithm

- c(i,j): link cost from node *i* to *j*
- D(v): current cost source → *v*
- p(v): *v*'s predecessor along path from source to *v*
- S: set of nodes whose least cost path definitively known

```
1  Initialization:
2    S = {A};
3    for all nodes v
4      if v adjacent to A
5        then D(v) = c(A,v);
6        else D(v) = ∞ ;
7
8  Loop
9      find w not in S such that D(w) is a minimum;
10     add w to S;
11     update D(v) for all v adjacent to w and not in S:
12       if  D(w) + c(w,v) < D(v) then
             // w gives us a shorter path to v than we've found so far
13           D(v) = D(w) + c(w,v); p(v) = w;
14 until all nodes in S;
```

# Example: Dijkstra's Algorithm

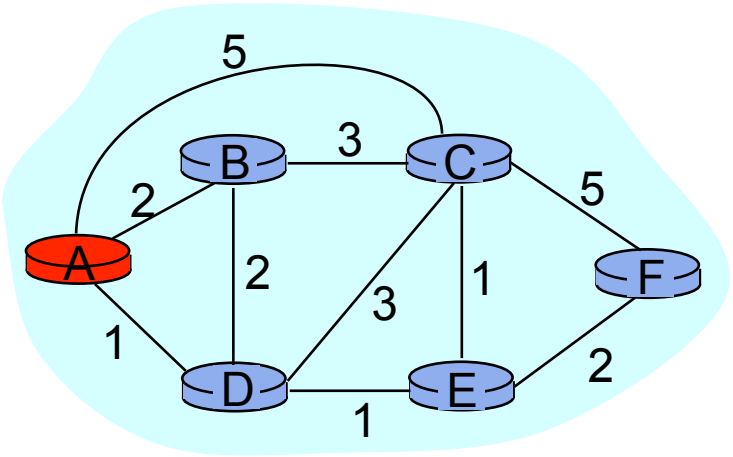
| Step | set S | D(B),p(B) | D(C),p(C) | D(D),p(D) | D(E),p(E) | D(F),p(F) |
|---|---|---|---|---|---|---|
| 0 | A | 2,A | 5,A | 1,A | ∞ | ∞ |
| 1 | | | | | | |
| 2 | | | | | | |
| 3 | | | | | | |
| 4 | | | | | | |
| 5 | | | | | | |




```
1  Initialization:
2    S = {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

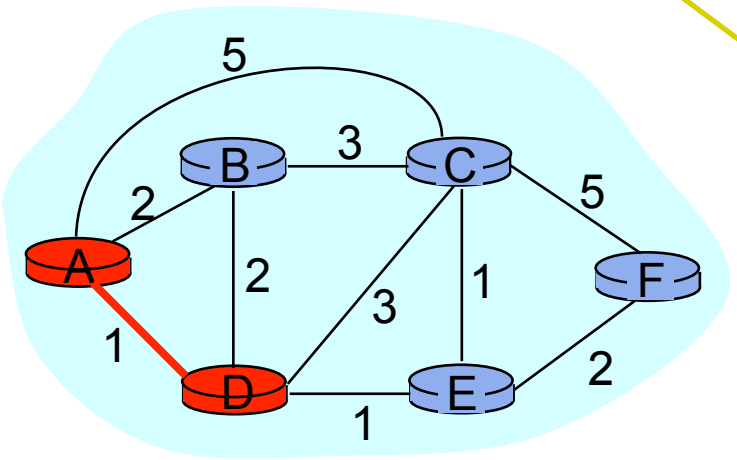
| Step | set S | D(B),p(B) | D(C),p(C) | D(D),p(D) | D(E),p(E) | D(F),p(F) |
|---|---|---|---|---|---|---|
| 0 | A | 2,A | 5,A | 1,A | ∞ | ∞ |
| 1 | | | | | | |
| 2 | | | | | | |
| 3 | | | | | | |
| 4 | | | | | | |
| 5 | | | | | | |




```
...
8   Loop
9     find w not in S s.t. D(w) is a minimum;
10    add w to S;
11    update D(v) for all v adjacent
        to w and not in S:
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 S;
```

# Example: Dijkstra's Algorithm

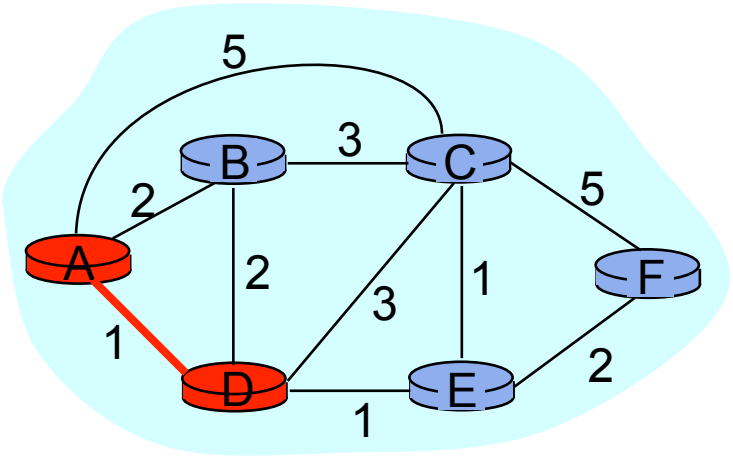
| Step | set S | D(B),p(B) | D(C),p(C) | D(D),p(D) | D(E),p(E) | D(F),p(F) |
|---|---|---|---|---|---|---|
| 0 | A | 2,A | 5,A | 1,A | ∞ | ∞ |
| 1 | AD | | | | | |
| 2 | | | | | | |
| 3 | | | | | | |
| 4 | | | | | | |
| 5 | | | | | | |




```
...
8   Loop
9     find w not in S s.t. D(w) is a minimum;
10    add w to S;
11    update D(v) for all v adjacent
        to w and not in S:
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 S;
```

# Example: Dijkstra's Algorithm

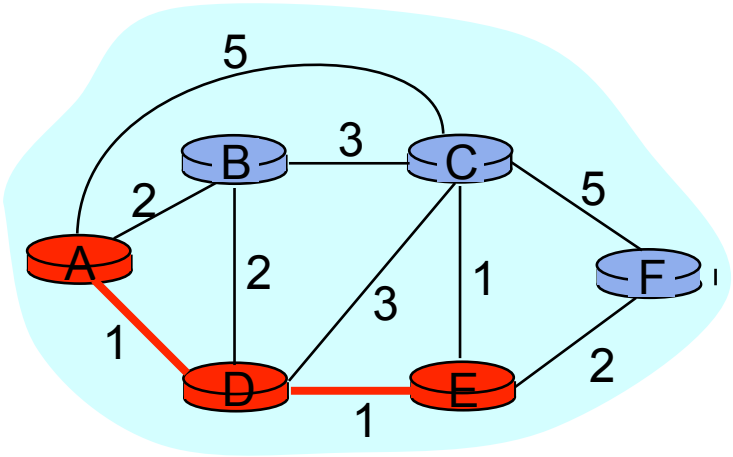
| Step | set S | D(B),p(B) | D(C),p© | D(D),p(D) | D(E),p(E) | D(F),p(F) |
|---|---|---|---|---|---|---|
| 0 | A | 2,A | 5,A | 1,A | ∞ | ∞ |
| 1 | AD | | 4,D | | 2,D | |
| 2 | | | | | | |
| 3 | | | | | | |
| 4 | | | | | | |
| 5 | | | | | | |




```
…
8   Loop
9     find w not in S s.t. D(w) is a minimum;
10    add w to S;
11    update D(v) for all v adjacent
        to w and not in S:
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 S;
```

# Example: Dijkstra's Algorithm

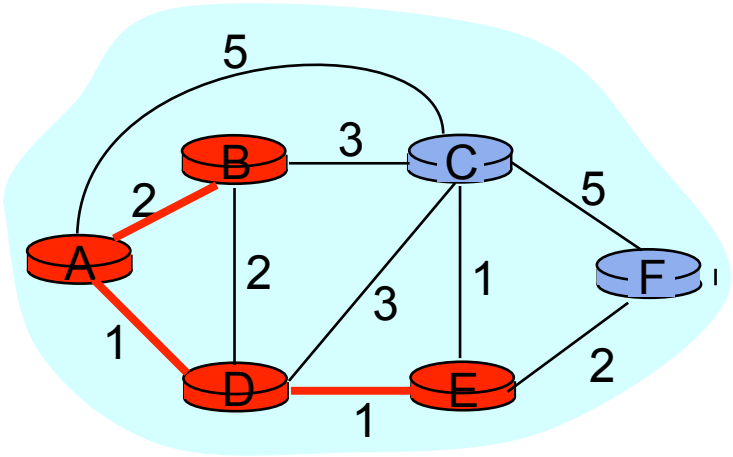
| Step | set S | D(B),p(B) | D(C),p(C) | D(D),p(D) | D(E),p(E) | D(F),p(F) |
|---|---|---|---|---|---|---|
| 0 | A | 2,A | 5,A | 1,A | ∞ | ∞ |
| 1 | AD | | 4,D | | 2,D | |
| 2 | ADE | | 3,E | | | 4,E |
| 3 | | | | | | |
| 4 | | | | | | |
| 5 | | | | | | |




```
…
8   Loop
9     find w not in S s.t. D(w) is a minimum;
10    add w to S;
11    update D(v) for all v adjacent
        to w and not in S:
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 S;
```

# Example: Dijkstra's Algorithm

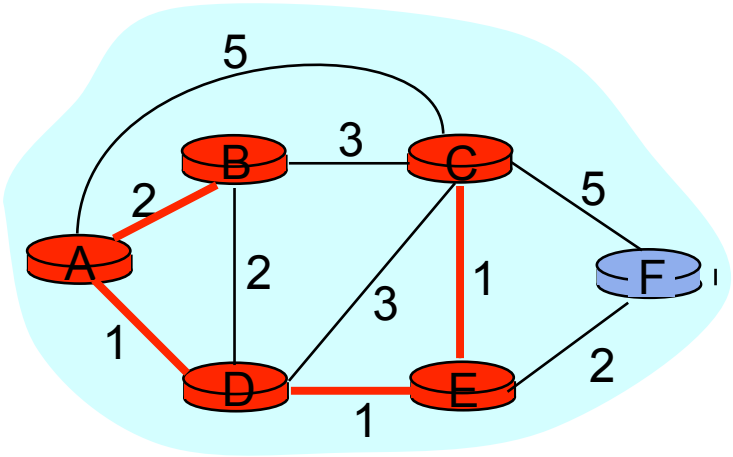
| Step | set S | D(B),p(B) | D(C),p(C) | D(D),p(D) | D(E),p(E) | D(F),p(F) |
|---|---|---|---|---|---|---|
| 0 | A | 2,A | 5,A | 1,A | ∞ | ∞ |
| 1 | AD | | 4,D | | 2,D | |
| 2 | ADE | | 3,E | | | 4,E |
| 3 | ADEB | | | | | |
| 4 | | | | | | |
| 5 | | | | | | |




```
…
8   Loop
9     find w not in S s.t. D(w) is a minimum;
10    add w to S;
11    update D(v) for all v adjacent
        to w and not in S:
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 S;
```

# Example: Dijkstra's Algorithm

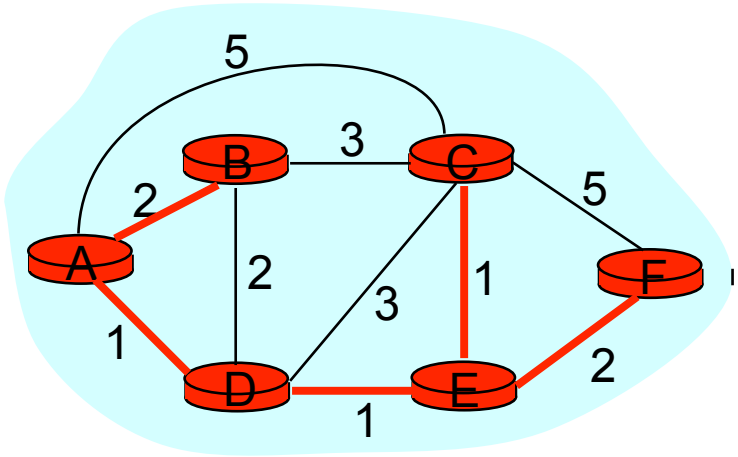
| Step | set S | D(B),p(B) | D(C),p(C) | D(D),p(D) | D(E),p(E) | D(F),p(F) |
|---|---|---|---|---|---|---|
| 0 | A | 2,A | 5,A | 1,A | ∞ | ∞ |
| 1 | AD | | 4,D | | 2,D | |
| 2 | ADE | | 3,E | | | 4,E |
| 3 | ADEB | | | | | |
| 4 | ADEBC | | | | | |
| 5 | | | | | | |




```
…
8   Loop
9     find w not in S s.t. D(w) is a minimum;
10    add w to S;
11    update D(v) for all v adjacent
        to w and not in S:
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 S;
```

# Example: Dijkstra's Algorithm

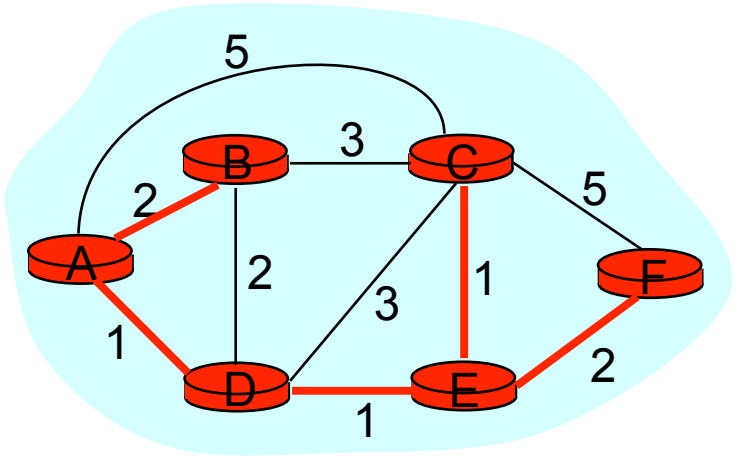
| Step | set S | D(B),p(B) | D(C),p(C) | D(D),p(D) | D(E),p(E) | D(F),p(F) |
|---|---|---|---|---|---|---|
| 0 | A | 2,A | 5,A | 1,A | ∞ | ∞ |
| 1 | AD | | 4,D | | 2,D | |
| 2 | ADE | | 3,E | | | 4,E |
| 3 | ADEB | | | | | |
| 4 | ADEBC | | | | | |
| 5 | ADEBCF | | | | | |




```
…
8   Loop
9     find w not in S s.t. D(w) is a minimum;
10    add w to S;
11    update D(v) for all v adjacent
        to w and not in S:
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 S;
```

# Example: Dijkstra's Algorithm

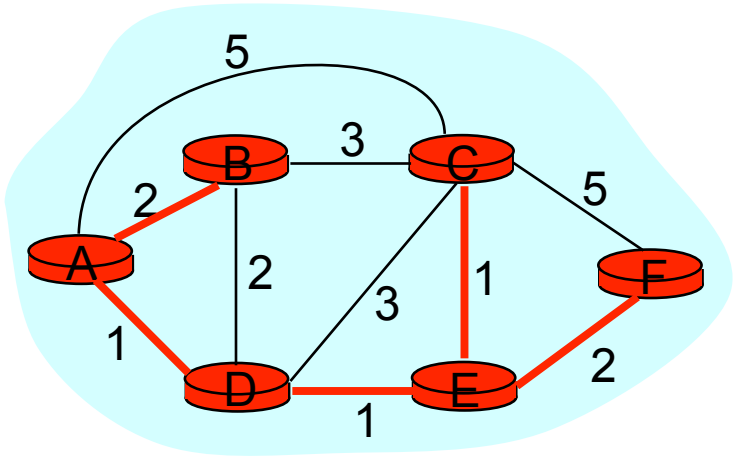
| Step | set S | D(B),p(B) | D(C),p(C) | D(D),p(D) | D(E),p(E) | D(F),p(F) |
|---|---|---|---|---|---|---|
| 0 | A | 2,A | 5,A | 1,A | ∞ | ∞ |
| 1 | AD | | 4,D | | 2,D | |
| 2 | ADE | | 3,E | | | 4,E |
| 3 | ADEB | | | | | |
| 4 | ADEBC | | | | | |
| 5 | ADEBCF | | | | | |




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

# The Routing Table

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



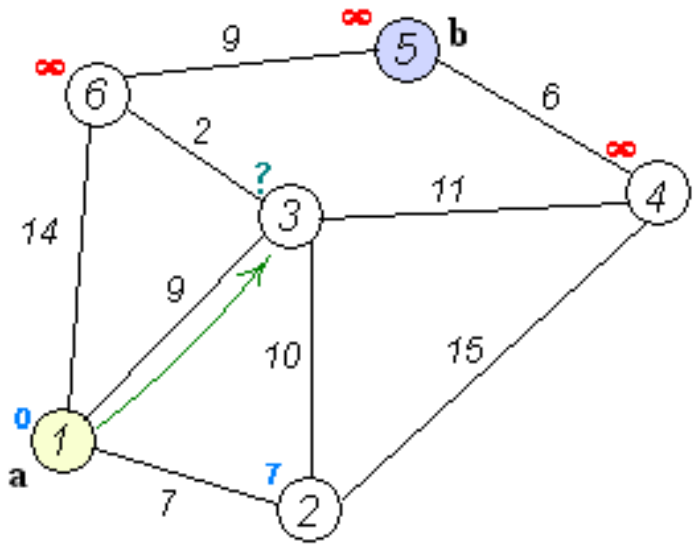

| Destination | Link |
|---|---|
| B | (A,B) |
| C | (A,D) |
| D | (A,D) |
| E | (A,D) |
| F | (A,D) |

# Algorithmic Complexity

- How many messages needed to flood link state messages?
  - O(|N||E|), where |N| is no. nodes; |E| is no. 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|))

# Dijkstra in action



# Dijkstra in action