COSC264
Introduction to Computer Networks and the Internet

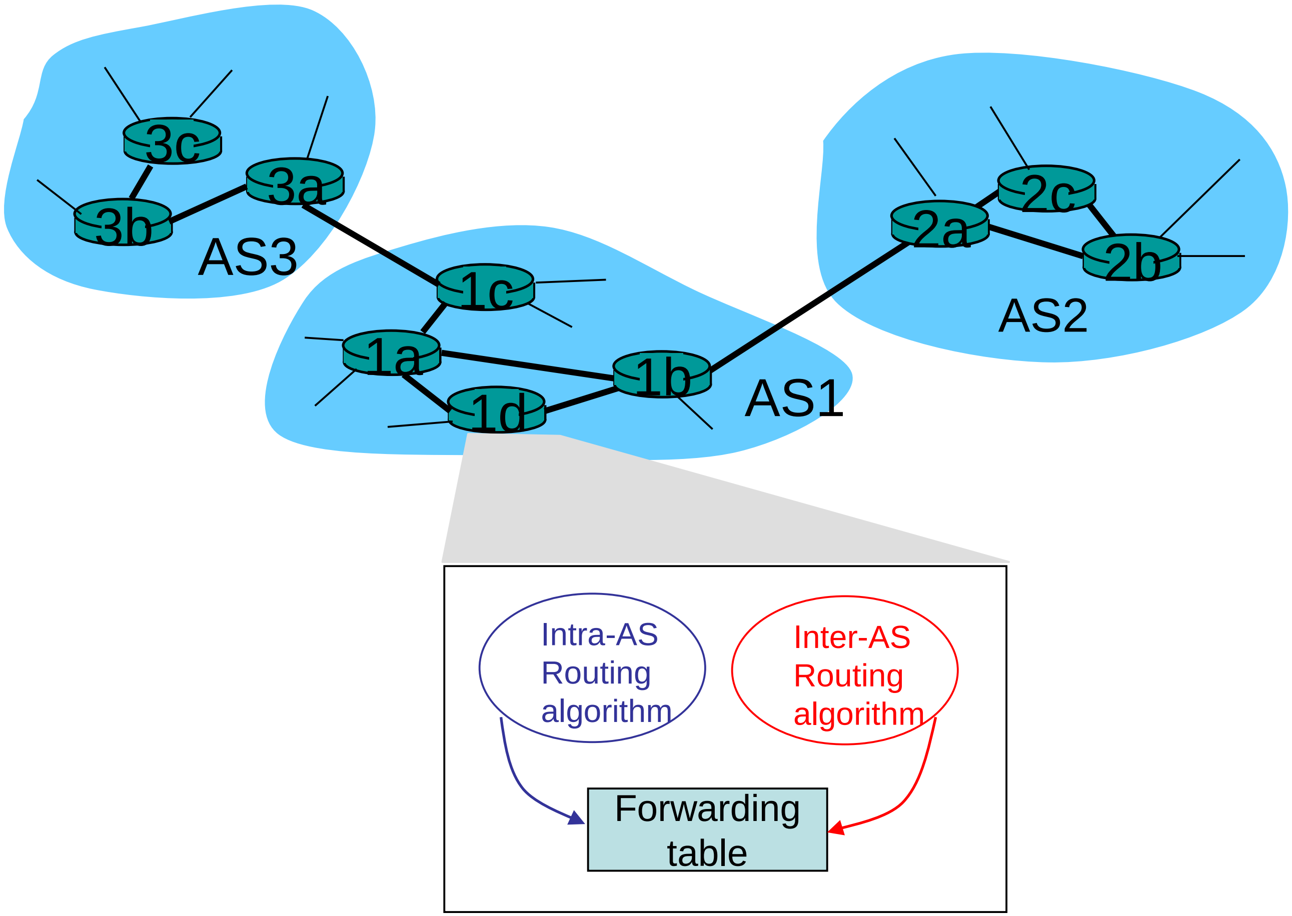
Introduction to Routing- Link State Routing

Dr Barry Wu
Wireless Research Centre
University of Canterbury
barry.wu@canterbury.ac.nz

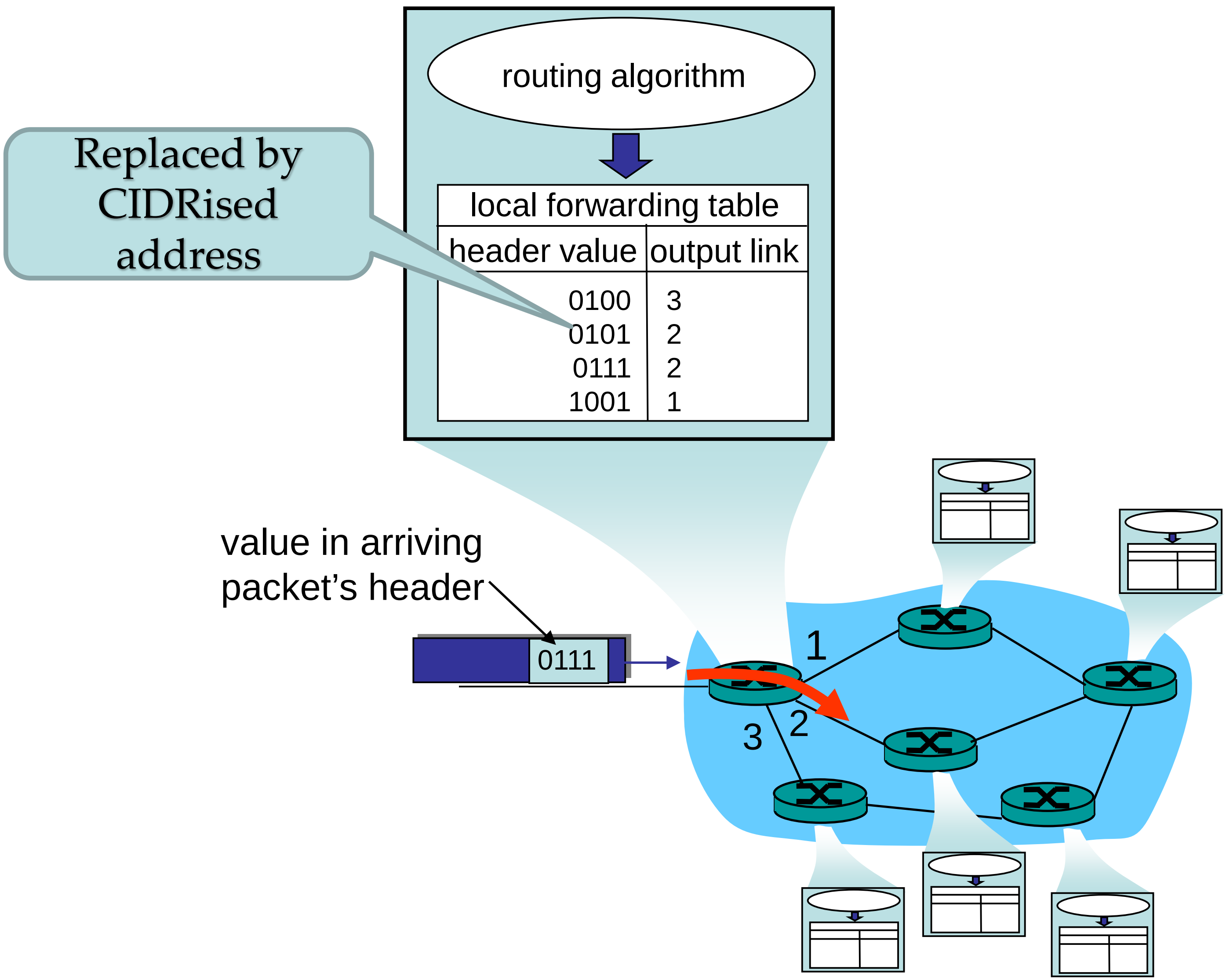
A quick review

- Layer approach and services
- Hierarchical routing and Autonomous System (AS)
- Routing vs forwarding
- Classification of routing algorithms

Hierarchical routing in the Internet



Routing and Forwarding



Outline – today

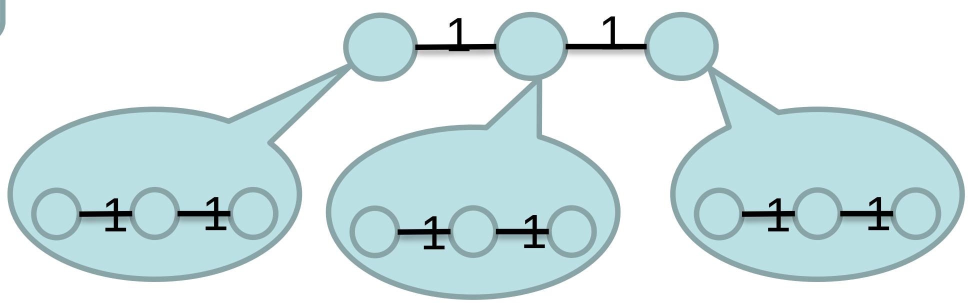
- Network layer overview
- Routing overview
- Link-state routing (Dijkstra's algorithm)
- Distance-vector routing (Bellman-Ford)
- Summary

Routing Algorithms and Routing Protocols

Intra-AS Routing

Routing Protocols	Routing Algorithms
RIP	Bellman-Ford (Distance-vector) Algorithm
OSFP	Dijkstra's Algorithm
BGP	Bellman-Ford (Distance-vector) Algorithm

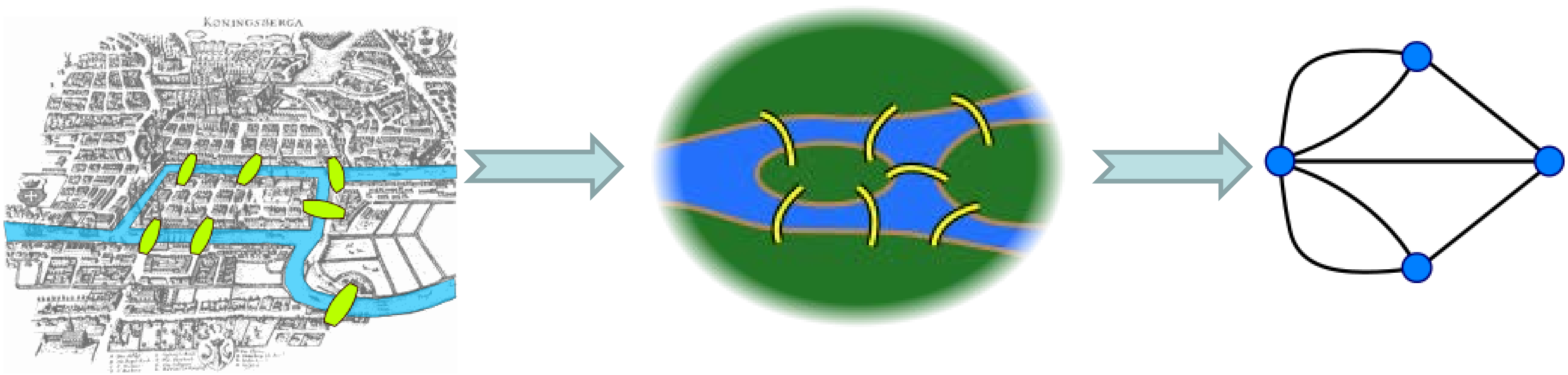
Inter-AS Routing



The Internet routing protocols (RIP, OSPF, and BGP) are ***load-insensitive***.

Euler and Graph Theory

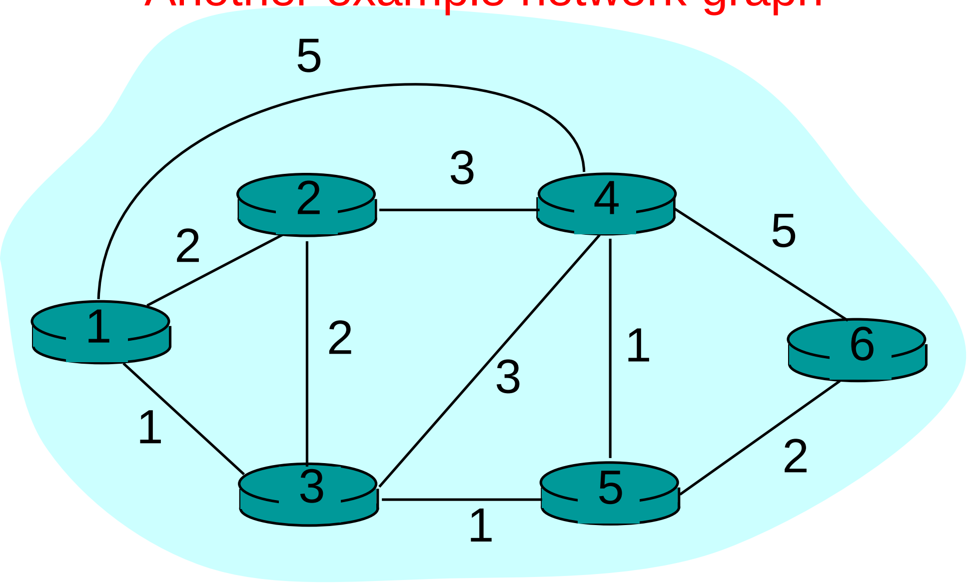
- Seven Bridges of Königsberg. 1783
- [Wiki]



Modeling a network

- Modeled as a graph
  - Routers ⇒ nodes
  - Link ⇒ edges

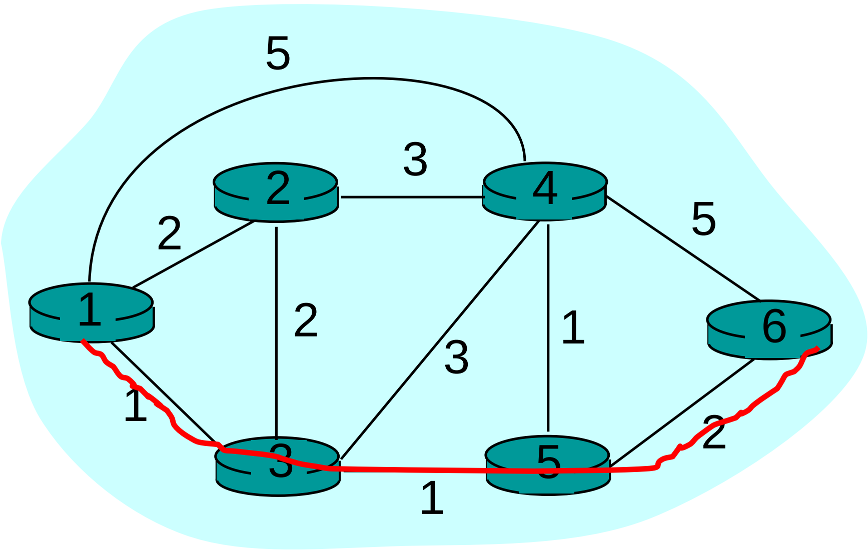
Another example network graph

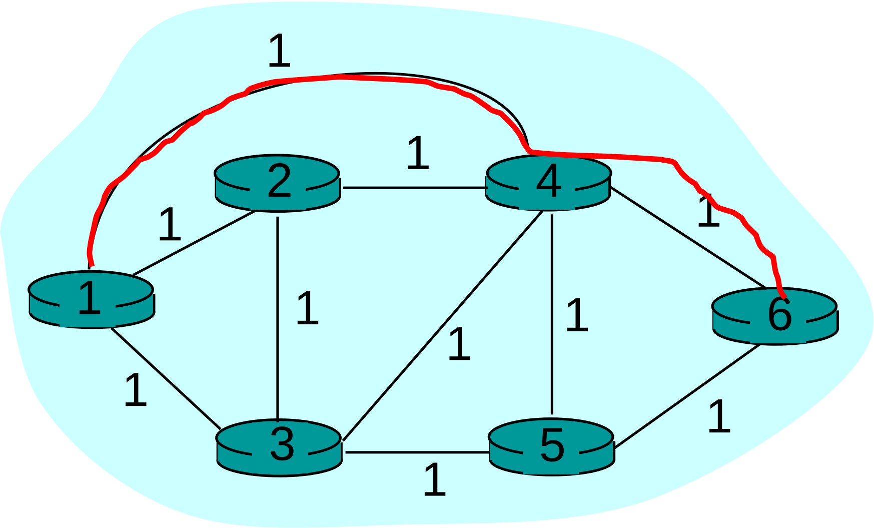


- Edge labels (called metrics) can be interpreted differently
  - as costs, e.g., delay, monetary transmission costs, geographical distance
  - as available resources, e.g., number of available phone trunks, current available capacity given the set of flows that already use this link

Routing algorithms

- To find least cost path
  - **Shortest path** if all link costs equal (measures hops)



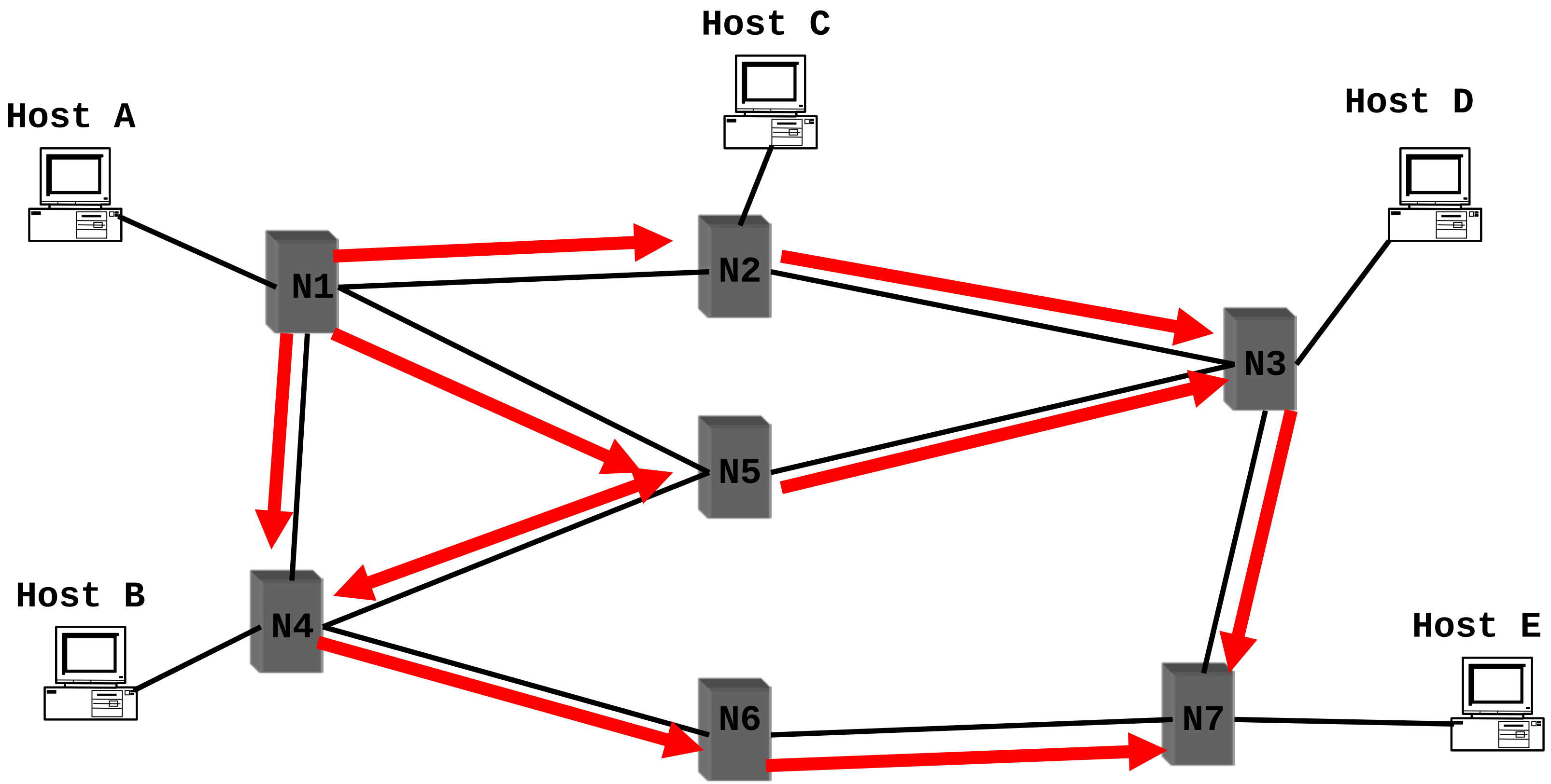


Link State Routing

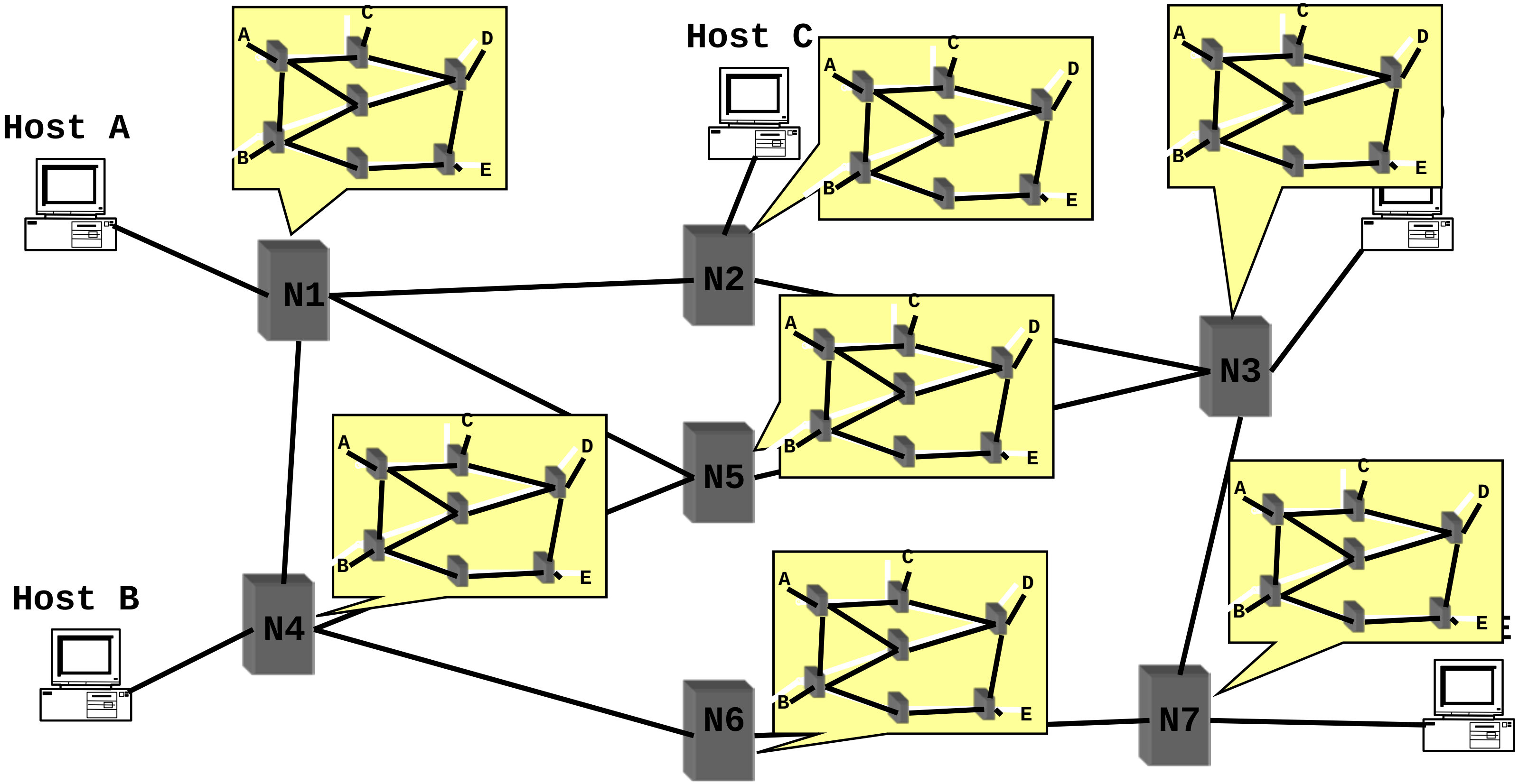
- Each router has **complete network picture**
  - Topology, Link costs
- How does each router get the global state?
  - Each router reliably **floods** information about its neighbors to every other router;
  - All routers have consistent information;

Link State: Control Traffic

- Each node floods its local information
- Each node ends up knowing the *entire* network topology node

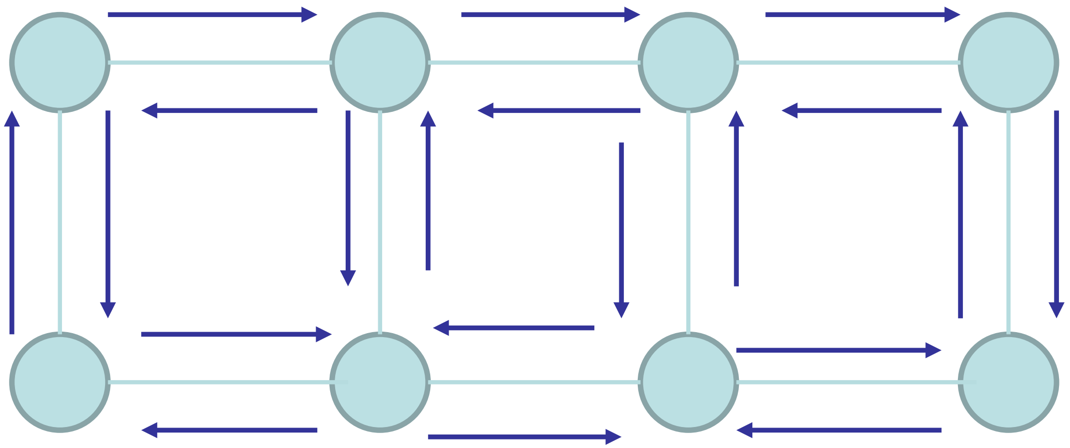


Link State: Node State



Flooding could be a danger!

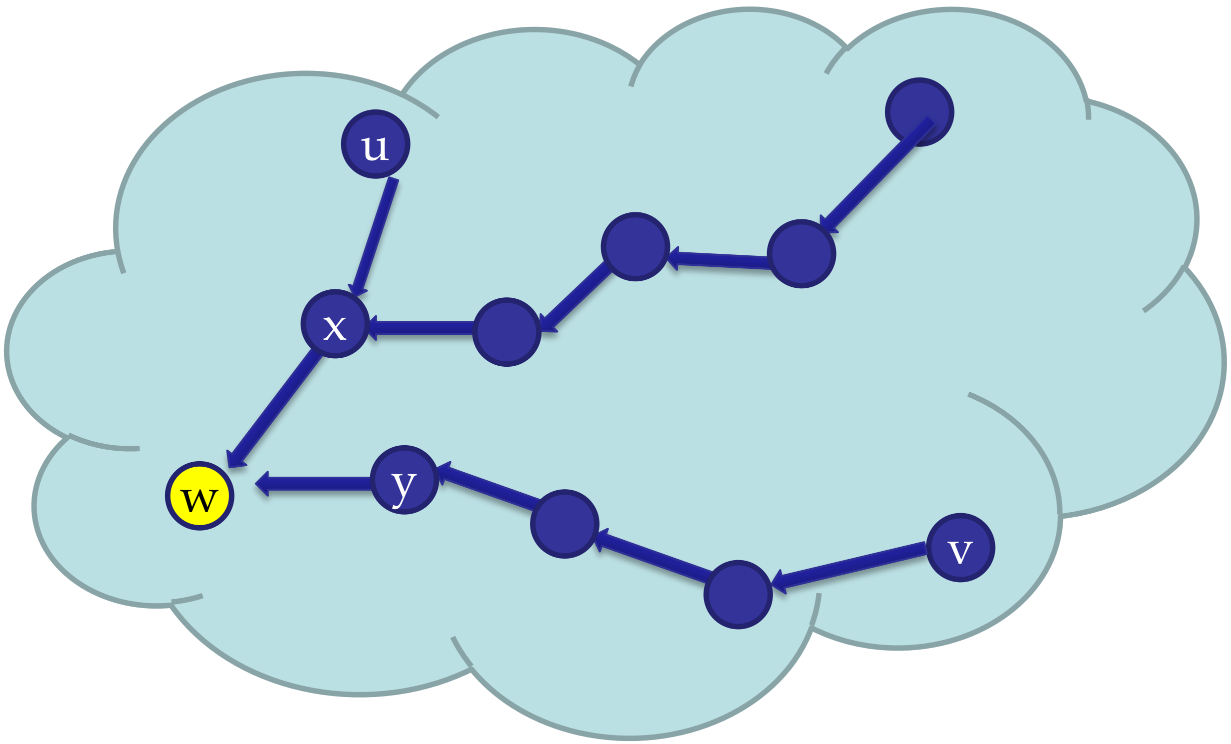
Broadcast storm!



There are sophisticated algorithms doing the broadcasting job!
(Controlled flooding, spanning-tree broadcast)

Link State Routing

- Each router independently calculates the least-cost path from itself to every other router;
  - Using Dijkstra's Algorithm;
  - Generates a forwarding table for every destination;



Dest.	Next-hop
u	x
v	y
…	…

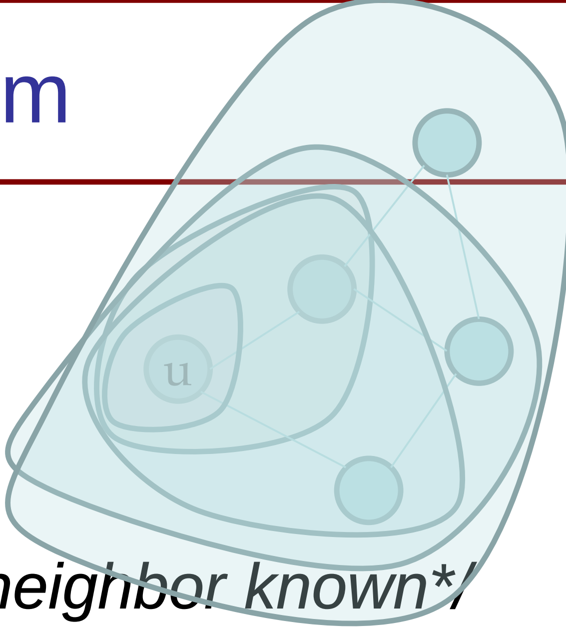
Dijkstra's Algorithm

- INPUT:
  - Network topology (graph), with link costs
- OUTPUT:
  - Least cost paths from one node to all other nodes

Dijkstra's Algorithm

- **S**: nodes whose least-cost path already known
  - Initially, **S = {*u*}** where *u* is the source node
  - Add one node to **S** in each iteration
- **D(v)**: current cost of path from source to node *v*
  - Initially, **D(v) = c(*u,v*)** for all nodes *v* adjacent to *u*
  - … and **D(v) = ∞** for all other nodes *v*
  - Continually update **D(v)** as shorter paths are learned
- *p(v):* predecessor node along path from source to *v*, that is next to *v*
- *c(i,j):* link cost from node *i* to *j*; cost infinite if not direct neighbors; **≥ 0**

Dijkstra's Algorithm

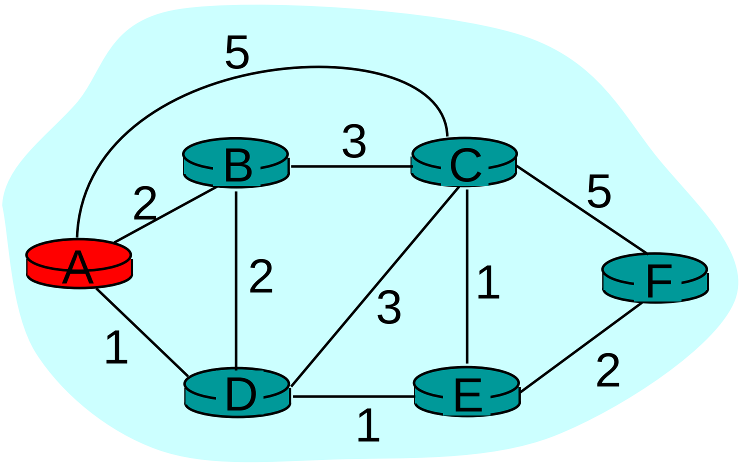



```
Initialization:
  S = {u} /* u is the source*/
  for all nodes v
    if v is adjacent to u {
        then D(v) = c(u,v) /* cost of neighbor known*/
        else D(v) = ∞ /* cost of others unknown*/

Loop
  find w not in S with the smallest D(w)
   add w to S
   update D(v) for all v adjacent to w and not in S:
       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 S
```

Example: Dijkstra's Algorithm

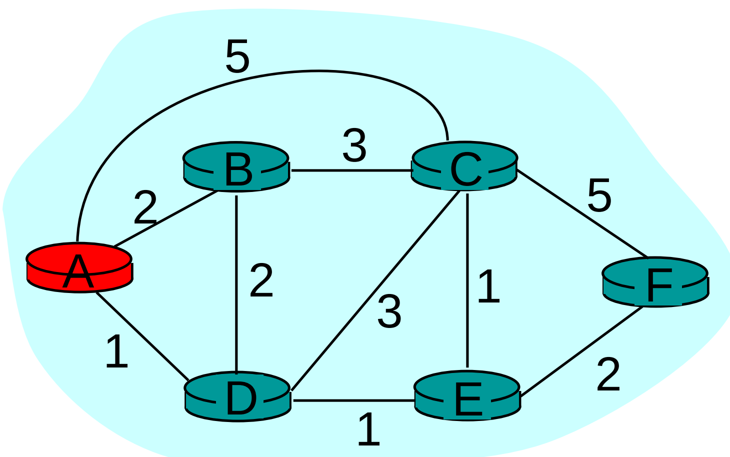
Step	start S	D(B),p(B)	D(C),p(C)	D(D),p(D)	D(E),p(E)	D(F),p(F)
0	A	2,A	5,A	1,A	∞	∞
1						
2						
3						
4						
5						



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

Example: Dijkstra's Algorithm

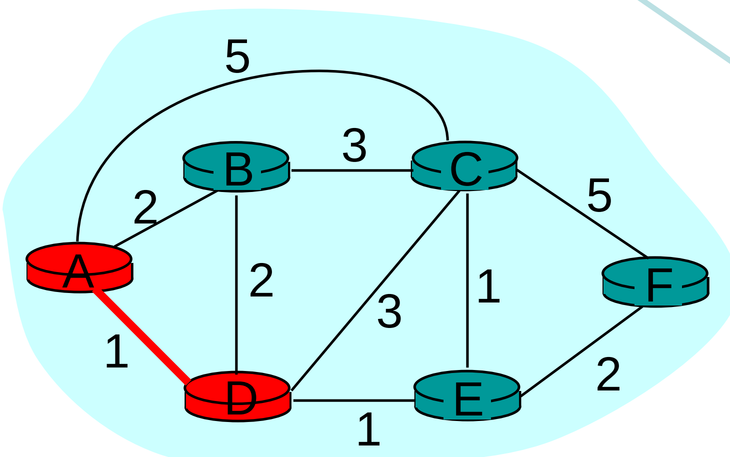
Step	start S	D(B),p(B)	D(C),p(C)	D(D),p(D)	D(E),p(E)	D(F),p(F)
0	A	2,A	5,A	1,A	∞	∞
1						
2						
3						
4						
5						




```
…
Loop
    find w not in S s.t. D(w) is a minimum;
    add w to S;
   update D(v) for all v adjacent
      to w and not in S:
   If D(w) + c(w,v) < D(v) then D(v) =
   D(w) + c(w,v); p(v) = w;
    until all nodes in S;
```

Example: Dijkstra's Algorithm

Step	start 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						



…

Loop

find **w** not in **S** s.t. D(w) is a minimum;

add **w** to **S**;

update D(v) for all **v** adjacent to **w** and not in **S**:

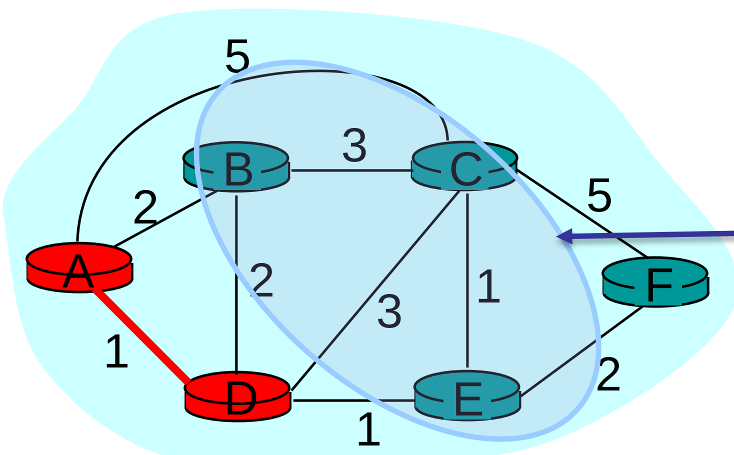
If D(w) + c(w,v) < D(v) then

D(v) = D(w) + c(w,v); p(v) = w;

until all nodes in S;

Example: Dijkstra's Algorithm

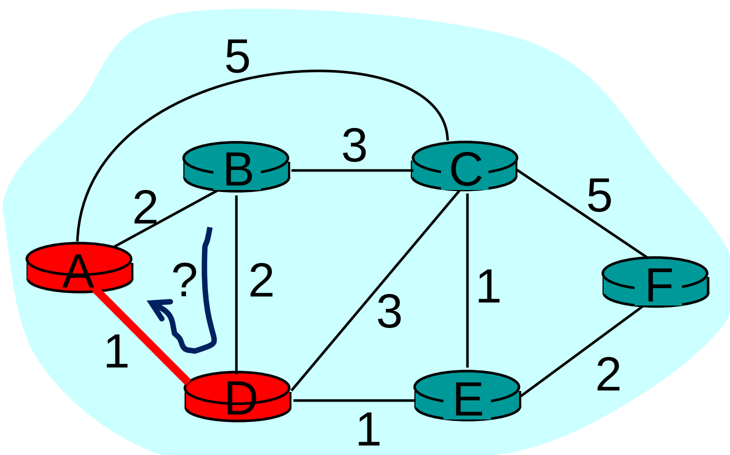
Step	start S	D(B),p(B)	D(C),p(C)	**D(D),p(D)**	D(E),p(E)	D(F),p(F)
0	A	2,A	5,A	**1,A**	∞	∞
1	AD					∞
2						
3						
4						
5						



…
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
D(v) = D(w) + c(w,v); p(v) = w;
13 ***until all nodes in S;***

Example: Dijkstra's Algorithm

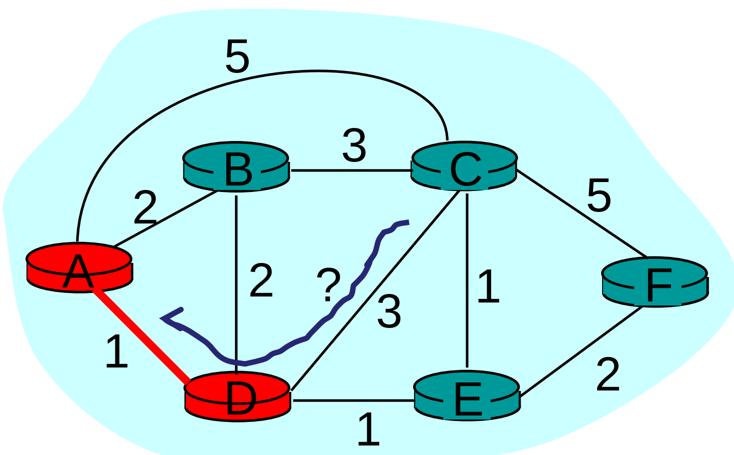
Step	start S	D(B),p(B)	D(C),p(C)	**D(D),p(D)**	D(E),p(E)	D(F),p(F)
0	A	2,A	5,A	**1,A**	∞	∞
1	AD	2,A				∞
2						
3						
4						
5						



…
Loop
  find **w** not in **S** s.t. D(w) is a minimum;
  add **w** to **S**;
  update D(v) for all **v** adjacent
    to **w** and not in **S**:
  If D(w) + c(w,v) < D(v) then
    D(v) = D(w) + c(w,v); p(v) = w;
until all nodes in S;

Example: Dijkstra's Algorithm

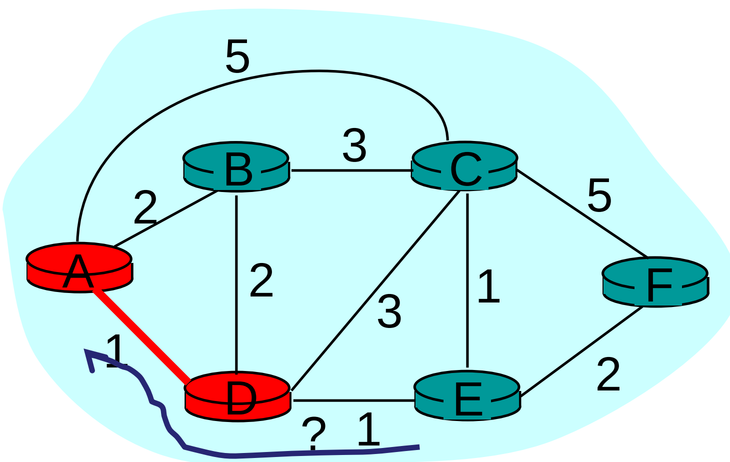
Step	start S	D(B),p(B)	D(C),p(C)	**D(D),p(D)**	D(E),p(E)	D(F),p(F)
0	A	2,A	5,A	**1,A**	∞	∞
1	AD	2,A	4,D	1,A		∞
2						
3						
4						
5						




```
…
Loop
    find w not in S s.t. D(w) is a minimum;
    add w to S;
    update D(v) for all v adjacent
      to w and not in S:
    If D(w) + c(w,v) < D(v) then
    D(v) = D(w) + c(w,v); p(v) = w;
until all nodes in S;
```

Example: Dijkstra's Algorithm

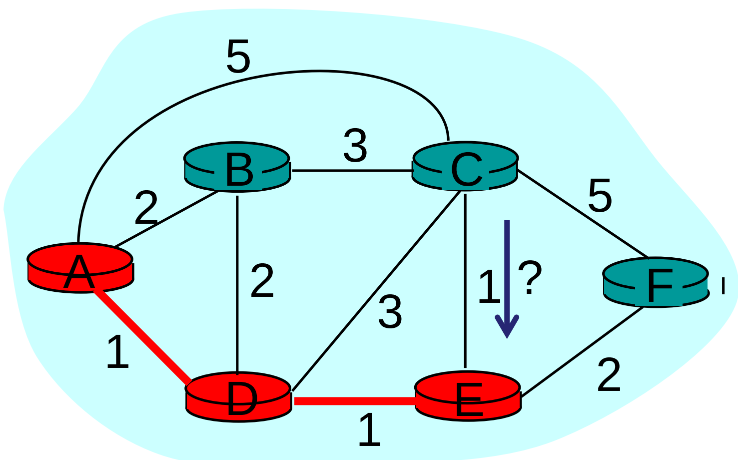
Step	start S	D(B),p(B)	D(C),p(C)	**D(D),p(D)**	D(E),p(E)	D(F),p(F)
0	A	2,A	5,A	**1,A**	∞	∞
1	AD	2,A	4,D	1,A	2,D	∞
2						
3						
4						
5						




```
…
Loop
    find w not in S s.t. D(w) is a minimum;
    add w to S;
    update D(v) for all v adjacent
      to w and not in S:
    If D(w) + c(w,v) < D(v) then
      D(v) = D(w) + c(w,v); p(v) = w;
until all nodes in S;
```

Example: Dijkstra's Algorithm

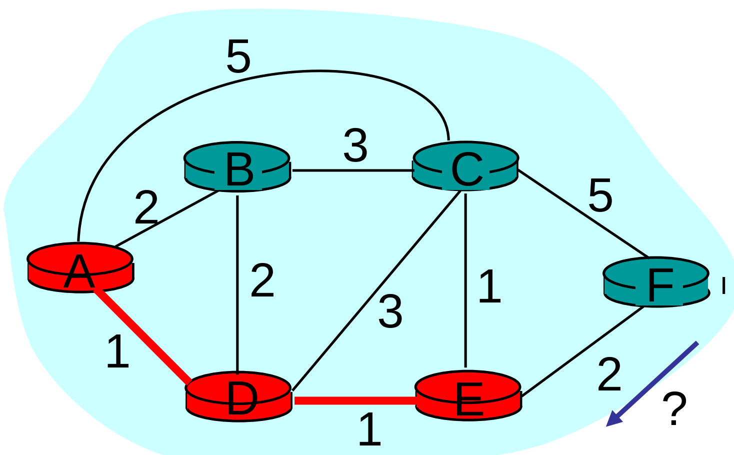
Step	start S	D(B),p(B)	D(C),p(C)	**D(D),p(D)**	**D(E),p(E)**	D(F),p(F)
0	A	2,A	5,A	1,A	∞	∞
1	AD	2,A	4,D	1,A	2,D	∞
2	ADE	2,A	3,E	1,A	2,D	
3						
4						
5						




```
…
Loop
    find w not in S s.t. D(w) is a minimum;
    add w to S;
  update D(v) for all v adjacent
     to w and not in S:
  If D(w) + c(w,v) < D(v) then
       D(v) = D(w) + c(w,v); p(v) = w;
until all nodes in S;
```

Example: Dijkstra's Algorithm

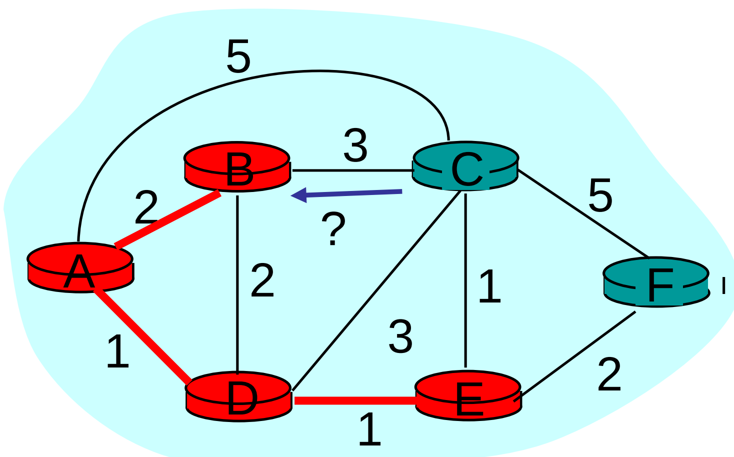
Step	start S	D(B),p(B)	D(C),p(C)	**D(D),p(D)**	**D(E),p(E)**	D(F),p(F)
0	A	2,A	5,A	1,A	∞	∞
1	AD	2,A	4,D	1,A	2,D	∞
2	ADE	2,A	3,E	1,A	2,D	4,E
3						
4						
5						




```
…
Loop
    find w not in S s.t. D(w) is a minimum;
    add w to S;
  update D(v) for all v adjacent
      to w and not in S:
  If D(w) + c(w,v) < D(v) then
        D(v) = D(w) + c(w,v); p(v) = w;
until all nodes in S;
```

Example: Dijkstra's Algorithm

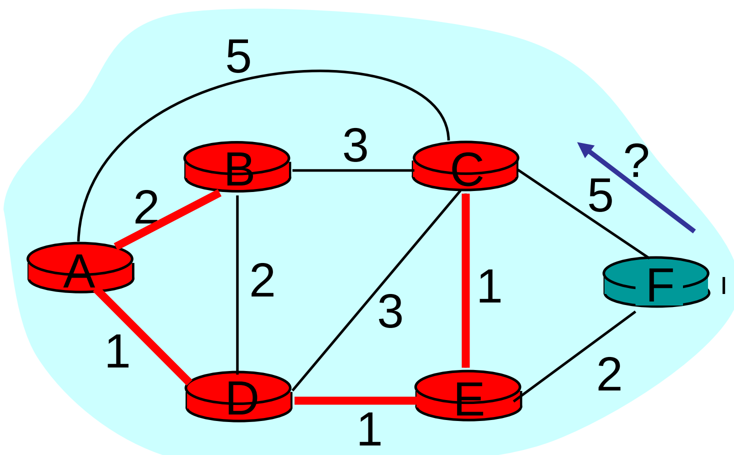
Step	start S	**D(B),p(B)**	D(C),p(C)	**D(D),p(D)**	**D(E),p(E)**	D(F),p(F)
0	A	2,A	5,A	1,A	∞	∞
1	AD	2,A	4,D	1,A	2,D	∞
2	ADE	2,A	3,E	1,A	2,D	4,E
3	ADEB	2,A	3,E	1,A	2,D	4,E
4						
5						




```
…
Loop
    find w not in S s.t. D(w) is a minimum;
    add w to S;
   update D(v) for all v adjacent
      to w and not in S:
   If D(w) + c(w,v) < D(v) then
        D(v) = D(w) + c(w,v); p(v) = w;
    until all nodes in S;
```

Example: Dijkstra's Algorithm

Step	start 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,A	4,D	1,A	2,D	∞
2	ADE	2,A	3,E	1,A	2,D	4,E
3	ADEB	2,A	3,E	1,A	2,D	4,E
4	ADEBC	2,A	3,E	1,A	2,D	4,E
5						



…

Loop

find **w** not in **S** s.t. D(w) is a minimum;

add **w** to **S**;

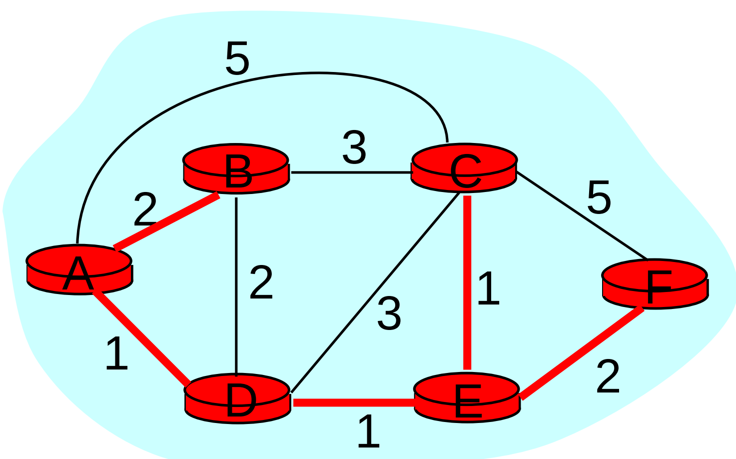
update D(v) for all **v** adjacent
to **w** and not in **S**:

If D(w) + c(w,v) < D(v) then
D(v) = D(w) + c(w,v); p(v) = w;

until all nodes in S;

Example: Dijkstra's Algorithm

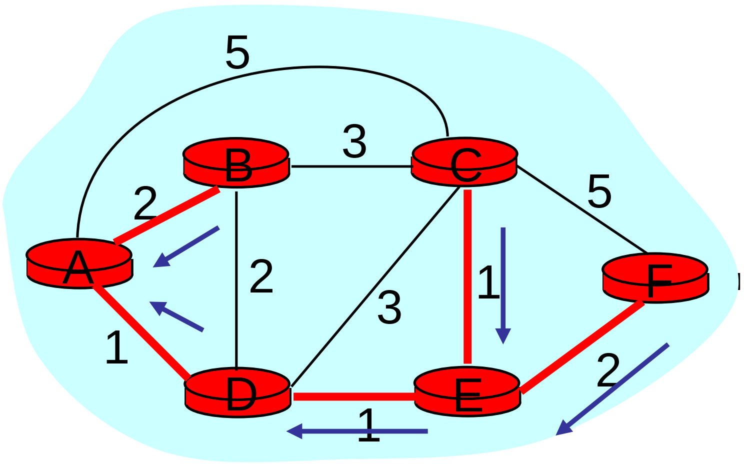
Step	start S	**D(B),p(B)**	**D(C),p(C)**	**D(D),p(D)**	**D(E),p(E)**	D(F),p(F)
0	A	2,A	5,A	1,A	∞	∞
1	AD	2,A	4,D	1,A	2,D	∞
2	ADE	2,A	3,E	1,A	2,D	4,E
3	ADEB	2,A	3,E	1,A	2,D	4,E
4	ADEBC	2,A	3,E	1,A	2,D	4,E
5	ADEBCF	2,A	3,E	1,A	2,D	4,E




```
…
Loop
    find w not in S s.t. D(w) is a minimum;
    add w to S;
   update D(v) for all v adjacent
      to w and not in S:
   If D(w) + c(w,v) < D(v) then
   D(v) = D(w) + c(w,v); p(v) = w;
 until all nodes in S;
```

Example: Dijkstra's Algorithm

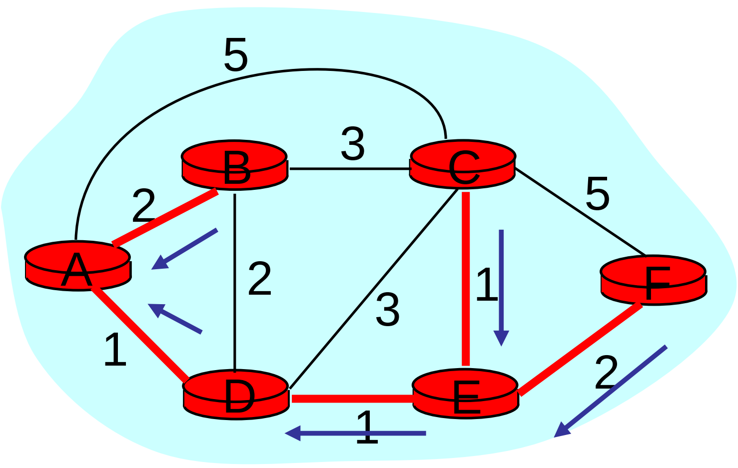
Step	start S	D(B),p(B)	D(C),p(C)	D(D),p(D)	D(E),p(E)	D(F),p(F)
0	A	2,A	5,A	1,A	∞	∞
1	AD	2,A	4,D	1,A	2,D	∞
2	ADE	2,A	3,E	1,A	2,D	4,E
3	ADEB	2,A	3,E	1,A	2,D	4,E
4	ADEBC	2,A	3,E	1,A	2,D	4,E
5	ADEBCF	2,A	3,E	1,A	2,D	4,E



…
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 D(v) = D(w) + c(w,v); p(v) = w;
13 ***until all nodes in S;***

The Forwarding Table

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



Destination	Next hop
B	B
C	D
D	D
E	D
F	D

Dijkstra's Algorithm – set *p(v)*

```
Initialization:
  S = {u} /* u is the source*/
  for all nodes v
    if v is adjacent to u {
        then D(v) = c(u,v) /* cost of neighbor known*/
        else D(v) = ∞ /* cost of others unknown*/

Loop
   find w not in S with the smallest D(w)
   add w to S
   update D(v) for all v adjacent to w and not in S:
      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 S
```

Dijkstra's Algorithm – set *p(v)*

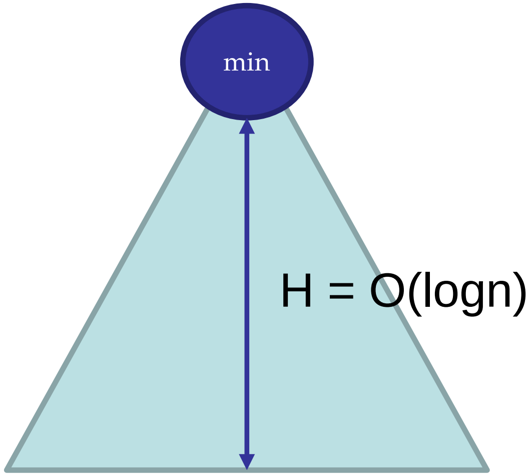
Initialization:
  S = {u} / u is the source*/*
  for all nodes *v*
    if *v* is adjacent to *u* {
      then *D(v) = c(u,v) /* cost of neighbor known*/*
      else *D(v)* = ∞ */* cost of others unknown*/*

Loop
  find *w* not in *S* with the smallest *D(w)*
  add *w* to *S*
  update D(v) for all v adjacent to w and not in S:
    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 S*

Dijkstra's algorithm, discussion

Algorithm complexity: n nodes

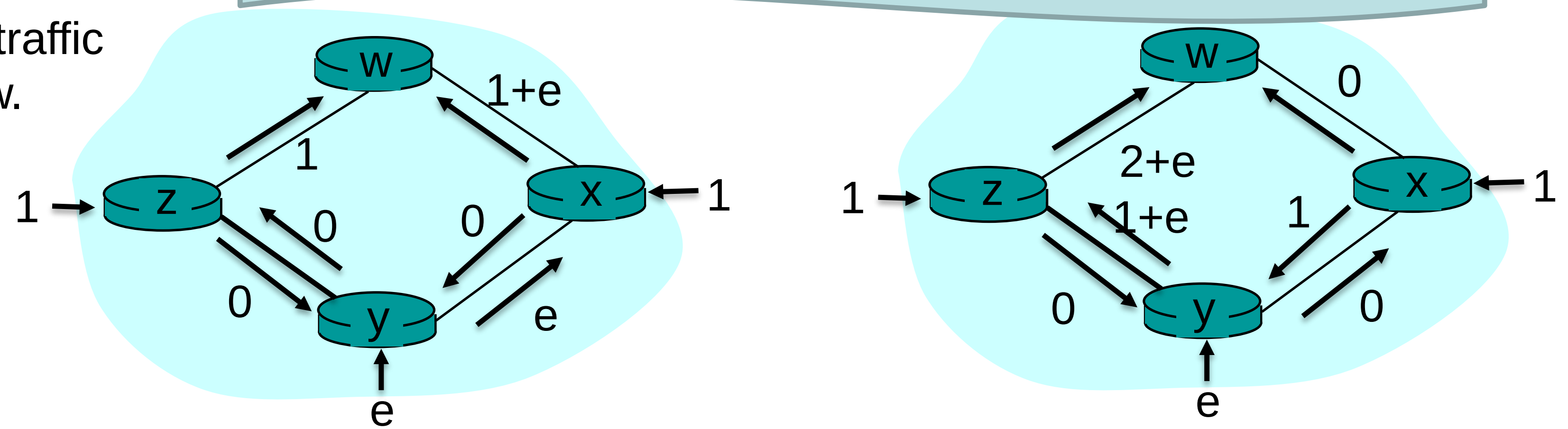
- each iteration: need to check every node, w, not in N
- n(n-1)/2 comparisons: $O(n^2)$
- more efficient implementations possible: O(nlogn)
  - Using a min-heap;
  - we can find out the node with min cost in O(logn);
  - Total cost = O(log(n-1) + log(n-2) + … + log1)
  - = O(nlogn).



Oscillation with link-state routing

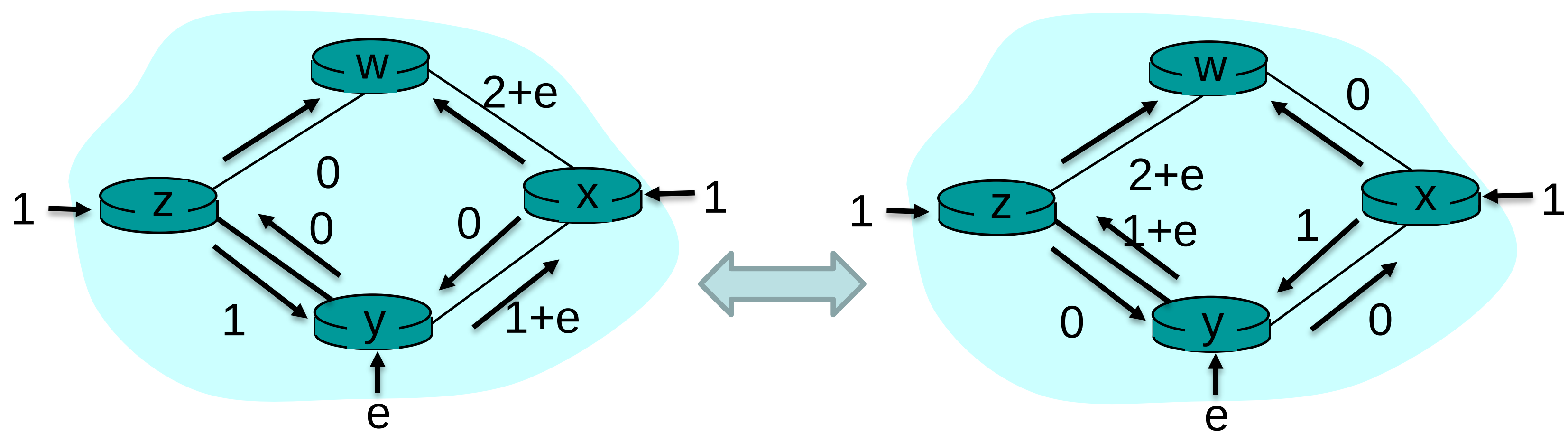
Today's Internet routing algorithms are load-*insensitive*!

Dest. of all traffic is w.



a. Initial routing

b. x,y detect better path to w, clockwise



c. x, y, z detect better path to w, counterclockwise

d. x,y,z detect better path to w, clockwise

Summary: Link-State Routing

- Each router broadcasts the link state
  - To give every router a complete view of the graph
- Each router runs Dijkstra’s algorithm
  - Compute least-cost paths, then construct forwarding table

References

- [KR3] James F. Kurose, Keith W. Ross, *Computer networking: a top-down approach featuring the Internet*, 3rd edition.

Acknowledgements

- Slides are developed mainly based on slides from the following two sources:
  - Prof Aleksandar Kuzmanovic's lecture notes for CS340,Northwestern University, https://users.cs.northwestern.edu/~akuzma/classes/CS340-w05/lecture_notes.htm