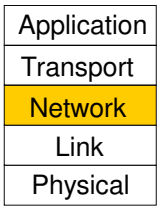
# CSE 461 – Routing

## Routing

- Focus:
  - How to find and set up paths through a network

- Distance-vector and link-state
- Shortest path routing
- Key properties of schemes



## Forwarding / Routing

- Fowarding
  - Router determines the next hop of an incoming packet based on something in the packet (e.g., destination address)
  - Based on table lookup in a *routing table*
  - This scheme says all packets from a src to a dest follow the same path
- Routing
  - Routers engage in a distributed protocol to
    - Exchange information
    - Establish their own routing table
  - Primary goal: loop free routes
  - Secondary goal: efficient routes
  - This scheme says packets from a src to a dest may follow distinct paths

3

## Routing

- Loop-freeness motivates using trees
  - In general, we can deal with paths obeying some monotonicity property
    - The next hop destination is “better” than the source by some measure
- We’ve seen distributed tree construction
  - Learning bridge spanning tree algorithm
- What’s different now?
  - Link layer vs. network layer
  - Single, arbirtrary root vs. every destination is a root
  - Foolproof vs. managed

4

## To find trees – two routing methods

- *Distance-vector* and *Link-state*
- Distance-vector method:
  - Every router collects information about its neighbors' connectivity to every destination
    - I don't need to know the entire path, just who to forward to
- Link-state method:
  - Every router collects information about 1-hop connectivity of every other router
    - The union is the network graph
    - Every router knows the graph…

5

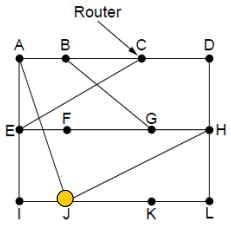
## Distance Vector Algorithm

- Each router maintains:
  - A vector of costs to all destinations
  - A routing table giving next hops
  - Information about one-hop costs to each neighbor
- With that, run a distributed Bellman-Ford (learning bridge-like) shortest path spanning tree algorithm
  - Periodically send copy of distance vector to neighbors
  - On reception of a vector, if your neighbor's path to a destination plus cost to that neighbor cost is better
  - Update the cost and next-hop in your outgoing vectors
- Assuming no changes, will converge to a set of routing tables that represent short-path trees

6

## DV Example

- Consider the activity at node J




| To | A | I | H | K | New estimated delay from J | Line |
|---|---|---|---|---|---|---|
| A | 0 | 24 | 20 | 21 | 8 | A |
| B | 12 | 36 | 31 | 28 | 20 | A |
| C | 25 | 18 | 19 | 36 | 28 | I |
| D | 40 | 27 | 8 | 24 | 20 | H |
| E | 14 | 7 | 30 | 22 | 17 | I |
| F | 23 | 20 | 19 | 40 | 30 | I |
| G | 18 | 31 | 6 | 31 | 18 | H |
| H | 17 | 20 | 0 | 19 | 12 | H |
| I | 21 | 0 | 14 | 22 | 10 | I |
| J | 9 | 11 | 7 | 10 | 0 | – |
| K | 24 | 22 | 22 | 0 | 6 | K |
| L | 29 | 33 | 9 | 9 | 15 | K |
| | JA delay is 8 | JI delay is 10 | JH delay is 12 | JK delay is 6 | New routing table for J | |

Vectors received from J's four neighbors

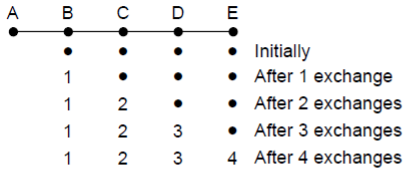


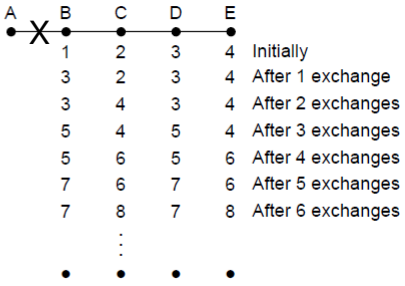
## Are We Done?

- Problems
  - Does it scale?
    - Forwarding table sizes grow with number of destinations
      - This isn't peculiar to DV, though…
  - Does it work?
    - This is another form of scaling issue
    - To work, has to converge (much) faster than things change
    - The rate of change in the network is proportional to the scale of the network

## DV problem – convergence dynamics

- Consider knowledge of cost to reach A at other nodes




Desired convergence "Count to infinity scenario"

9

## DV problem -- dynamics

- Good news (better routes) propagates quickly
- Bad news (failures) propagates slowly
  - inferred by exploration
- Leads to "count to infinity" loops
  - Many heuristics (split horizon, poison reverse)
  - Takes ordered updates to eliminate (e.g., EIGRP uses diffusing computations) that are complicated and slow convergence
  - No great solutions
- No longer widely used except for resource constrained or legacy networks.

10

## Routing Information Protocol (RIP)

- DV protocol with hop count as metric
  - Infinity value is 16 hops; limits network size
  - Includes split horizon with poison reverse
- Routers send vectors every 30 seconds
  - With triggered updates for link failures
  - Time-out in 180 seconds to detect failures
- RIPv1 specified in RFC1058
  - www.ietf.org/rfc/rfc1058.txt
- RIPv2 (adds authentication etc.) in RFC1388
  - www.ietf.org/rfc/rfc1388.txt

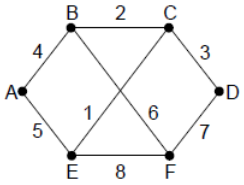
11

## Routing Algorithm #2: Link State Routing

- Same assumptions/goals, but different approach than DV:
  - All routers learn the full network topology (!)
  - Each compute shortest path spanning trees
    - They should all agree…
- Two phases:
  1. Topology dissemination (flooding)
  2. Shortest-path calculation (Dijkstra's algorithm)
- Why?
  - In DV, routers don't know anything about full paths
    - E.g., no information about what next hop router will do with your packet
  - LS uses global information
    - Faster convergence and hopefully better stability

12

## LS example database




(a)

Link

| A |
|---|
| Seq. |
| Age |
| B 4 |
| E 5 |

| B |
|---|
| Seq. |
| Age |
| A 4 |
| C 2 |
| F 6 |

State

| C |
|---|
| Seq. |
| Age |
| B 2 |
| D 3 |
| E 1 |

| D |
|---|
| Seq. |
| Age |
| C 3 |
| F 7 |

Packets

| E |
|---|
| Seq. |
| Age |
| A 5 |
| C 1 |
| F 8 |

| F |
|---|
| Seq. |
| Age |
| B 6 |
| D 7 |
| E 8 |

(b)

- Q: what is the flooding rule to build the database?
- Q: how are shortest paths computed from the database?



## Shortest Paths: Dijkstra's Algorithm

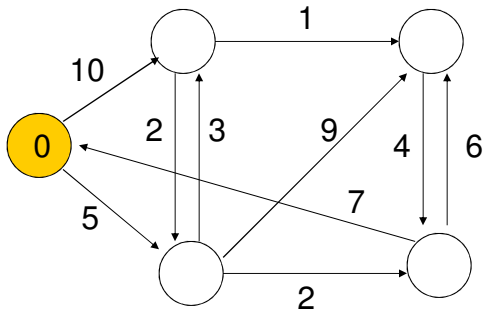
- Graph algorithm for single-source shortest path (i.e., sink tree)

```
S ← {}
Q ← <all nodes keyed by distance>
While Q != {}
        u ← extract-min(Q)
        S ← S plus {u}
        for each node v adjacent to u
                "relax" the cost of v
```

←u is done, add to shortest paths

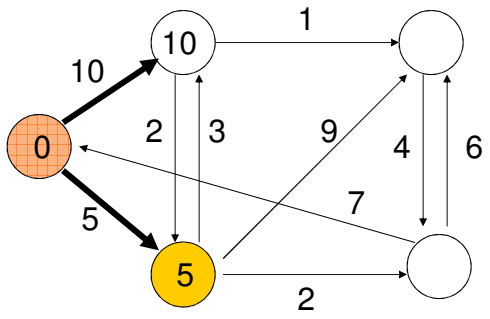
Among the currently unconnected nodes, add to the partial tree the node that is closest to the root

## Dijkstra Example – Step 1



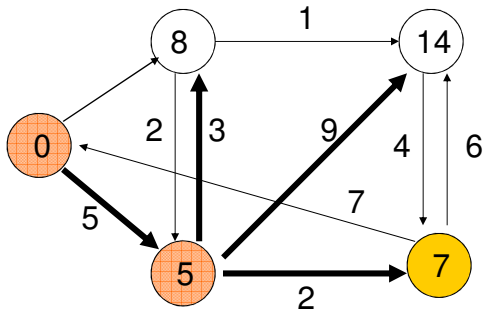

15

## Dijkstra Example – Step 2



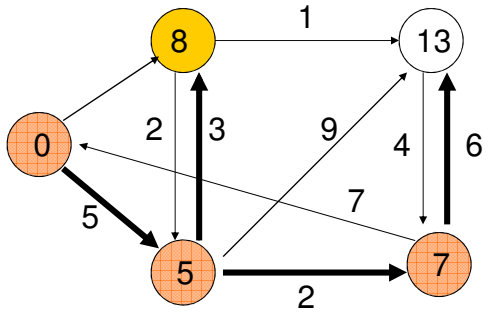

16

## Dijkstra Example – Step 3



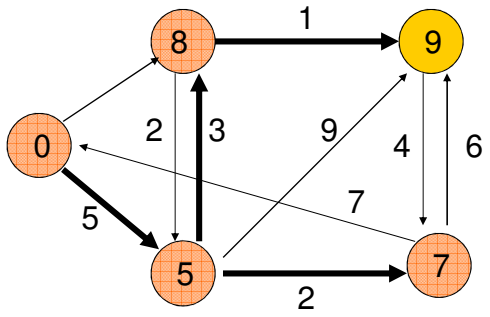

17

## Dijkstra Example – Step 4



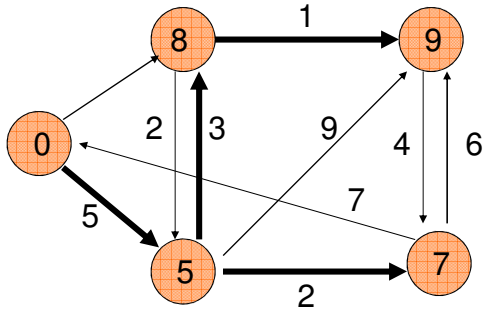

18

## Dijkstra Example – Step 5






## Dijkstra Example – Done



## Open Shortest Path First (OSPF)

- Widely-used Link State protocol today; see also ISIS
- Basic link state algorithms plus many features:
  - Load balancing: multiple equal cost routes
  - Extra hierarchy: partition into routing areas
  - Authentication of routing messages



## Routing – desirable properties

- Correctness
- Network efficiency
- Network fairness
- Rapid convergence
  - To correct routes that are stable after changes, with minimal transient loss
- Scalability
  - Of messages and router state
  - Particularly an issue for large, mobile, or multicast networks

## Comparison

| Property | Distance Vector | Link State |
|---|---|---|
| Correctness | Yes - Distributed Bellman Ford | Yes - Replicated shortest path |
| Efficiency | Approx- Least cost paths | Approx - Least cost paths |
| Fairness | Approx - Least cost paths | Approx - Least cost paths |
| Convergence | Slow – many exchanges | Fast – prop plus compute |
| Scalability | Good – O(1) per node/link | Moderate – at least O(edges) |

23

## Resource allocation timescales today

- From fast (very reactive) to slow (carefully planned)
  - Use of different timescales largely decouples mechanisms
- *Congestion control*
  - Adapts to packet loss; slows source
- *Routing*
  - Adapts to failures; finds paths with connectivity
- *Traffic engineering*
  - Route adjustments for cost/performance (e.g., weights)
- *Provisioning*
  - Build out network to match traffic workload

24

## Routing Implications

- The network may have physical connectivity, but be partitioned at the IP layer
- The path from A to B is likely different than the path from B to A
  - Routers seeing packets sent from host A to host B may not see the ACKs from B to A
- Host A can't reach host C, but A can reach B and B can reach C
- We still don't have a scheme that scales to Internet size

25