14. Intra-Domain Routing

- Distance vector routing – RIP
- Link-state routing – OSPF

Jon Turner – slides adapted from Kurose and Ross
Routing vs Forwarding

- **Routing algorithm** computes best routes thru network
- **Places result in forwarding table**
  - router data path uses forwarding table to forward individual packets

```
<table>
<thead>
<tr>
<th>prefix</th>
<th>out/nh</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.2.0.0/16</td>
<td>3/2.3.4.5</td>
</tr>
<tr>
<td>1.2.3.0/24</td>
<td>2/6.5.4.3</td>
</tr>
<tr>
<td>3.2.1.0/24</td>
<td>2/6.5.4.5</td>
</tr>
<tr>
<td>4.0.0.0/8</td>
<td>1/4.3.2.1</td>
</tr>
</tbody>
</table>
```

Address in arriving packet's header
Graph Abstraction

Graph: $G = (N, E)$

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

$E =$ set of links (more accurately, "adjacencies")

$= \{ (u,v), (u,x), (v,x), (v,w), (x,w), (x,y), (w,y), (w,z), (y,z) \}$

edges have costs

- costs may be uniform, proportional to distance, inversely related to link capacity, etc
- path cost is sum of link costs
Routing Algorithm Characteristics

- **Local knowledge – distance vector algorithms**
  - router knows physically-connected neighbors, link costs to neighbors
  - iterative process of computation, exchange of info with neighbors – eventually converges to global shortest paths

- **Global knowledge – link-state algorithms**
  - routers distribute local link information to all other routers
    - each router then computes best paths to all destinations

- **Static vs dynamic**
  - static: link costs, topology change only rarely
  - dynamic: costs and topology change frequently, requiring frequent routing computations
    - more responsive to traffic conditions
    - can cause large-scale and disruptive changes to routes
Distance Vector Algorithm

Bellman-Ford Equation (dynamic programming)

Define

\[ d_x(y) := \text{cost of least-cost path from } x \text{ to } y \]

Then

\[ d_x(y) = \min_v \{ c(x,v) + d_v(y) \} \]

where min is taken over all neighbors \( v \) of \( x \)

Forms the basis of a simple distributed algorithm for computing shortest paths
Distance-Vector Algorithm

- Each node $x$ maintains a *distance vector* containing “best known distances” to each destination
  - $D_x = \{ d_x(y) = \text{shortest known distance from } x \text{ to } y \}$
  - initially $d_x(y) = c(x,y)$ for neighbors, $\infty$ for non-neighbors

- A node sends its initial distance vector to its neighbors and re-sends it after every change

- When a node $x$ receives a distance vector from a neighbor, it stores a copy and updates its own ($D_x$)
  - for all destinations $y$, $d_x(y) = \min_v c(x,v) + d_v(y)$ where the minimum is taken over all neighbors $v$ of $x$
  - the node does the same update if the cost of one of its links changes

- When link costs are stable, distance vectors eventually converge to the actual shortest path distances
Responding to Link Cost Reductions

Link cost changes:
- node detects local link cost change
- updates routing info, recalculates distance vector
- if DV changes, notify neighbors

“good news travels fast”

$t_0 : x, y$ detect link-cost change, recompute their DVs, and send copies to neighbors

$t_1 :$ all receive updates and recompute DV; $z$ sends copy to neighbors

$t_2 : x, y$ receive $z$’s update, recompute own DV; since DVs do not change, no further activity
Responding to Increase in Costs

- After link cost increase
  - $y$ thinks best path to $x$ is through $z$ (with cost of 6)
  - $z$ now thinks best path to $x$ is through $y$ (with cost of 7)
  - $y$ now thinks best path to $x$ is through $z$ (with cost of 8)
  - and so forth – eventually correct value is reached, but can take a while – bad news travels slowly

- Can improve performance in such cases using *poisoned reverse*
  - if node $a$ thinks its best path to node $c$ is through its neighbor $b$, it tells $b$ that its distance to $c$ is infinity
    - this prevents $b$ from trying to reach $c$ through $a$
  - this helps in simple cases, but is not a general solution
Exercises

1. Consider the graph on the right side of slide 3. For nodes \( W \), \( Y \) and \( Z \) show the final distance vectors computed by the Bellman-Ford algorithm. In the distance vectors, list the nodes in alphabetical order. You need not do the Bellman-Ford computation step-by-step. Just show the final result.

2. Now, suppose the cost of the link from \( Y \) to \( Z \) changes from 2 to 7. Show the distance vectors that \( Y \) and \( Z \) send to each other and to \( W \) as a result of this change. Assume that the poisoned reverse technique is used. Show how each node updates its own distance vector after it receives the distance vector’s from its neighbors.
RIP (Routing Information Protocol)

- Early intra-domain routing protocol
- Distance vector algorithm
  - distance metric: # hops (max = 15 hops), each link has cost 1
  - distance vectors automatically sent to neighbors every 30 sec
    - router may also request its neighbor’s distance vector
  - each “advertisement” lists up to 25 destination subnets (in IP addressing sense)

Here is an example of a routing table from router A to destination subnets:

<table>
<thead>
<tr>
<th>subnet</th>
<th>hops</th>
<th>next router</th>
</tr>
</thead>
<tbody>
<tr>
<td>u</td>
<td>1</td>
<td>-</td>
</tr>
<tr>
<td>v</td>
<td>2</td>
<td>B</td>
</tr>
<tr>
<td>w</td>
<td>2</td>
<td>B</td>
</tr>
<tr>
<td>x</td>
<td>3</td>
<td>B</td>
</tr>
<tr>
<td>y</td>
<td>3</td>
<td>C</td>
</tr>
<tr>
<td>z</td>
<td>2</td>
<td>C</td>
</tr>
</tbody>
</table>
RIP Details

- RIP uses UDP packets to exchange routing information with neighbors (using port number 520)

- In many routers, RIP runs as a user-level process
  - the router typically includes a general purpose processor, often running a standard OS, like Linux
  - the routing process updates router forwarding table whenever the routing computation causes a change to its internal table
    - this may involve accessing special hardware components

- Unix and Linux computers can be configured to operate as routers
  - the OS kernel includes a forwarding table
  - the user level program *routed* (routing daemon) implements RIP and can update the forwarding table
Routing updates sent using multicast address 224.0.0.9.
  » all interested routers listen for updates
    • multicast packets limited to “one hop”
  » means that a RIP router need not be configured with IP addresses of neighboring routers

Poison reverse used to limit “count-to-infinity” problem
  » limited hop count also limits count-to-infinity

If no advertisement heard after 180 sec, neighbor is declared “dead”
  » routes via neighbor invalidated
  » new advertisements sent to (other) neighbors
  » neighbors send out new advertisements (if tables changed)
  » link failure info propagates to entire net

While RIP is easy to use, it does not scale well
  » ok for small LANs, not so effective in larger networks
Example of RIP

- **1.2.8.***
- **1.2.9.***
- **1.2.7.***
- **1.2.10.***
- **1.2.11.***
- **1.2.12.***
- **1.2.2.***
- **1.2.3.***
- **1.2.4.***
- **1.2.5.***

- **A**
- **B**
- **C**
- **D**
- **E**
- **F**
- **G**
- **H**

- **host number part of the IP address of the router interface**
- **interface number**
Exercises

Assume RIP version 2 when answering the following questions.

1. Suppose all routers come up at time 0. Show the initial state of A’s routing table.

2. Suppose A multicasts a route update on its interface 3. Show the routing table at D after this update is processed.

3. Suppose D now sends an update on its interface 1. Show the routing table at G after this update is processed.

4. Suppose G now sends an update on its interface 2. Show the routing table at I after this update is received.

5. Estimate the number of times that A recomputes its routing table as a result of all the updates that occur right after the routers come up.

6. If the link connecting D to subnet 1.2.12.* fails, D’s “metric” for this subnet becomes 16 (interpreted as infinity). How long does it take for D to learn of a new route to subnet 1.2.12.*, assuming that RIP uses the basic “poisoned reverse” technique? How long does it take if poisoned reverse is not used?

7. How many of routing updates does router C send in a 30 second routing update interval? How many does it receive?
Link-State Routing

- Divide route computation into two parts
  - distributing “link-state” to all routers
    - network topology plus link costs
  - computing best routes, using global link-state

- Distributing link-state requires protocol
  - typically uses a “flooding” technique to make sure that all information reaches all routers
  - important to minimize redundant transmissions

- Route computation can use any efficient algorithm
  - Dijkstra’s algorithm is the usual choice
  - can compute shortest paths in large networks in <1 second
Dijsktra’s Algorithm (refresher)

1. **Initialization:**
   
   2. \( S = \{s\}; \ d(s) = 0 \)
   
   3. for all edges \((s,v)\)

   4. \( d(v) = c(s,v) \)
   
   5. \( p(v) = s \)

   6. for non-neighbors, \( d(v) = \infty \)

7. **Loop**

8. select \( u \) not in \( S \) with smallest value of \( d(u) \)

9. add \( u \) to \( S \)

10. for all edges \((u,v)\) where \( v \) is not in \( S \)

11. if \( d(u) + c(u,v) < d(v) \) then

12. \( d(v) = d(u) + c(u,v) \)

13. \( p(v) = u \)

14. until all nodes are in \( S \)

Use heap to efficiently find best \( u \) in \( S \) – \( O(m \log n) \) algorithm
Open Shortest Path First (OSPF) RFC 2328

- Routing info carried in *Link State Advertisements* (LSA)
  - advertise the directly connected subnets and the cost of reaching adjacent routers
  - each LSA advertises one subnet or adjacent router
- Advertisements disseminated to entire routing domain
  - using “flooding” technique
  - sent in response to changes and periodically (every 30 min)
  - carried in OSPF messages directly over IP – (protocol #89)
    - OSPF provides its own mechanisms for reliable delivery
- Several packet types
  - Hello packets used to find neighbors within a L2-subnet
  - Database description packets – used to initialize new router
  - Link state requests, updates, acknowledgments
Example OSPF Packet Format

- Update packets used to distribute link state
- Each may contain multiple advertisements (LSAs)
- Multiple LSA types
  - router LSA describes one router’s interfaces/neighbors
  - network LSA lists an L2 subnet’s attached routers
  - ...
- LS sequence # used to control distribution of LSAs
  - LSAs with “newer” sequence numbers replace older

### link state update packet

<table>
<thead>
<tr>
<th>version</th>
<th>type</th>
<th>length</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>router id</td>
<td></td>
</tr>
<tr>
<td></td>
<td>area id</td>
<td></td>
</tr>
<tr>
<td></td>
<td>. . .</td>
<td></td>
</tr>
<tr>
<td></td>
<td># of LSAs</td>
<td></td>
</tr>
<tr>
<td>LSAs</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

### link state advertisement

<table>
<thead>
<tr>
<th>LS age</th>
<th>LS type</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>LS id</td>
</tr>
<tr>
<td></td>
<td>advertising router</td>
</tr>
<tr>
<td></td>
<td>LS sequence number</td>
</tr>
<tr>
<td></td>
<td>checksum</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Distributing Updates

Basic approach

- each router stores LSAs received from other routers
- LSA updates are propagated hop-by-hop
  - and re-sent if not acknowledged by “next-hop-router”
- when new LSA update comes in, check for a stored copy
  - if stored copy is newer (by seq#) than update, discard update
  - otherwise, store a copy of update and propagate to other neighbors
  - send acknowledgment to “previous-hop-router”
- router may receive several copies of an LSA, but typically only the first one received is propagated

Dealing with L2 subnets

- Each subnet includes special Designated Router (DR)
- link state updates are sent to the DR; it then multicasts them to all routers in subnet
- reduces number of updates that routers must process
OSPF Example

- Host number part of the IP address of the router interface.
- Interface number.

Nodes:
- A
- B
- C
- D
- E
- F
- G
- H
- I

Links:
- A to B: 1.1
- A to C: 1.2
- B to C: 1.1
- B to D: 1.1
- C to D: 1.1
- C to E: 1.2
- D to F: 1.1
- D to G: 1.2
- E to F: 1.1
- E to G: 1.2
- F to G: 1.1
- F to I: 1.2
- G to I: 1.1
- H to G: 1.2
- H to I: 1.1
- I to E: 1.2
- I to C: 1.2

Numbers:
- 1.2.1.*
- 1.2.2.*
- 1.2.3.*
- 1.2.4.*
- 1.2.5.*
- 1.2.6.*
- 1.2.7.*
- 1.2.8.*
- 1.2.9.*
- 1.2.10.*
- 1.2.11.*
- 1.2.12.*
Exercises

Suppose all routers come up at time 0, knowing only about their directly connected subnets and neighbors. Assume that the designated router for each subnet is the one that comes first alphabetically.

1. Suppose router A sends a router LSA on all of its interfaces, announcing all its neighbors and the costs of the links to its neighbors. Explain how router G might learn of this LSA? Would you expect router G to receive this LSA more than once? Explain.

2. Suppose that every subnet 1.2.k.* has a link cost metric of k. Show the routing table computed by node H, after LSAs have been received from all routers. The table should have a separate entry for every subnet.

3. Draw a conventional graph corresponding to the example network, where each link joining a pair of routers represents a connection through a single subnet.
Network With Multiple OSPF Areas

- **Area 1**
- **Area 2**
- **Area 3**

- **boundary router**
- **backbone router**
- **backbone area (0)**
- **area border routers**
- **internal routers**
Details of Multiple OSPF Areas

- **Two-level hierarchy**
  - backbone plus local *areas*
  - link-state advertisements only within area
  - each node has detailed area topology; only know direction (shortest path) to nets in other areas.

- **Area border routers**
  - “summarize” distances to networks in own area
  - advertise to other area border routers.

- **Backbone routers:** run OSPF routing within backbone only

- **Boundary routers**
  - connect to other AS’s
Routing and Traffic Engineering

- Shortest-path routing can be overly simplistic
  » typical cost metric (cost inversely proportional to link rate) ignores distances
    • so may take long distance path to avoid small capacity link
    • difficult to optimize for link capacity and distance at the same time
  » routing does not account for traffic volumes either

- “Traffic engineering” can used to improve routing
  » network operators compute best routes off-line, based on
    • knowledge of network topology (including link lengths/capacities) and traffic volumes among different locations
    • requires more sophisticated optimization methods, like linear programming
  » then, select OSPF link costs to produce desired routing
    • needs to produce reasonable routes even when links or routers fail
    • unfortunate “hack” used to address fundamental deficiency
Exercises

1. One way to account for link capacity and distance in OSPF is to set the link cost equal to $distance + C/capacity$ where $C$ is a constant chosen to provide an appropriate weighting between the two terms. Suppose we have three routers, one in Chicago, one in St. Louis and one in St. Charles. The routers in St. Louis and St. Charles each have a 10 Gb/s link to the Chicago router that is 500 km long. They are also connected by a 1 Gb/s link that is 10 km long. In the cost expression above, assume distance is expressed in units of km and capacity in terms of Gb/s. If $C=2000$, what is the total cost of the two paths joining the routers in St. Louis and St. Charles? What if $C=1000$? What is the largest value of $C$ that ensures that traffic will take the direct route between St. Louis and St. Charles?

2. Consider the method used in the last problem and suppose $C=1000$. Consider a network with 10 Gb/s links connecting LA to San Francisco to Portland to Seattle. Assuming the total distance is 1000 km, what is the cost of the complete path? Suppose there is also a 10 Gb/s link directly from LA to Seattle. Compare the cost of this link to the multi-hop path? Does the difference seem appropriate? What if all the links are 1 Gb/s?