Chapter 4
Network Layer

A note on the use of these ppt slides:

We’re making these slides freely available to all (faculty, students, readers). They’re in PowerPoint form so you can add, modify, and delete slides (including this one) and slide content to suit your needs. They obviously represent a lot of work on our part. In return for use, we only ask the following:

- If you use these slides (e.g., in a class) in substantially unaltered form, that you mention their source (after all, we’d like people to use our book!)
- If you post any slides in substantially unaltered form on a www site, that you note that they are adapted from (or perhaps identical to) our slides, and note our copyright of this material.

Thanks and enjoy! JFK/KWR

All material copyright 1996-2010
J.F Kurose and K.W. Ross, All Rights Reserved

Computer Networking:
A Top Down Approach
5th edition.
Jim Kurose, Keith Ross
Addison-Wesley, April 2009.
Chapter 4: Network Layer

4.1 Introduction
4.2 Virtual circuit and datagram networks
4.3 What’s inside a router
4.4 IP: Internet Protocol
   - Datagram format
   - IPv4 addressing
   - ICMP
   - IPv6
4.5 Routing algorithms
   - Link state
   - Distance Vector
   - Hierarchical routing
4.6 Routing in the Internet
   - RIP
   - OSPF
   - BGP
4.7 Broadcast and multicast routing
Interplay between routing, forwarding
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) \}

Remark: Graph abstraction is useful in other network contexts

Example: P2P, where $N$ is set of peers and $E$ is set of TCP connections
Graph abstraction: costs

- $c(x,x') =$ cost of link $(x,x')$
  - e.g., $c(w,z) = 5$
- cost could always be 1, or inversely related to bandwidth, or inversely related to congestion

Cost of path $(x_1, x_2, x_3, ..., x_p) = c(x_1, x_2) + c(x_2, x_3) + ... + c(x_{p-1}, x_p)$

**Question:** What’s the least-cost path between $u$ and $z$?

**Routing algorithm:** algorithm that finds least-cost path
Routing Algorithm classification

Global or decentralized information?

Global:
- all routers have complete topology, link cost info
- “link state” algorithms

Decentralized:
- router knows physically-connected neighbors, link costs to neighbors
- iterative process of computation, exchange of info with neighbors
- “distance vector” algorithms

Static or dynamic?

Static:
- routes change slowly over time

Dynamic:
- routes change more quickly
  - periodic update
  - in response to link cost changes
Chapter 4: Network Layer

4.1 Introduction
4.2 Virtual circuit and datagram networks
4.3 What’s inside a router
4.4 IP: Internet Protocol
   ▪ Datagram format
   ▪ IPv4 addressing
   ▪ ICMP
   ▪ IPv6

4.5 Routing algorithms
   ▪ Link state
   ▪ Distance Vector
   ▪ Hierarchical routing

4.6 Routing in the Internet
   ▪ RIP
   ▪ OSPF
   ▪ BGP

4.7 Broadcast and multicast routing
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 $y$; $= \infty$ 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:**
   - \( N' = \{u\} \)
   - for all nodes \( v \)
     - if \( v \) adjacent to \( u \)
       - then \( D(v) = c(u,v) \)
     - else \( D(v) = \infty \)

2. **Loop**
   - find \( w \) not in \( N' \) such that \( D(w) \) is a minimum
   - add \( w \) to \( N' \)
   - update \( D(v) \) for all \( v \) adjacent to \( w \) and not in \( N' \) :
     - \( D(v) = \min( D(v), D(w) + c(w,v) ) \)
   - /* new cost to \( v \) is either old cost to \( v \) or known shortest path cost to \( w \) plus cost from \( w \) to \( v \ )* /

3. until all nodes in \( N' \)
### Dijkstra’s algorithm: example

<table>
<thead>
<tr>
<th>Step</th>
<th>N'</th>
<th>D(v)</th>
<th>D(w)</th>
<th>D(x)</th>
<th>D(y)</th>
<th>D(z)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>p(v)</td>
<td>p(w)</td>
<td>p(x)</td>
<td>p(y)</td>
<td>p(z)</td>
</tr>
<tr>
<td>0</td>
<td>u</td>
<td>7,u</td>
<td>3,u</td>
<td>5,u</td>
<td>8,u</td>
<td>9,u</td>
</tr>
<tr>
<td>1</td>
<td>uw</td>
<td>6,w</td>
<td>5,u</td>
<td>11,w</td>
<td>9,w</td>
<td>9,w</td>
</tr>
<tr>
<td>2</td>
<td>uwx</td>
<td>6,w</td>
<td>11,w</td>
<td>14,x</td>
<td>14,x</td>
<td>14,x</td>
</tr>
<tr>
<td>3</td>
<td>uwxv</td>
<td>10,v</td>
<td>14,x</td>
<td>14,x</td>
<td>14,x</td>
<td>14,x</td>
</tr>
<tr>
<td>4</td>
<td>uwxvy</td>
<td>12,y</td>
<td>14,x</td>
<td>14,x</td>
<td>14,x</td>
<td>14,x</td>
</tr>
<tr>
<td>5</td>
<td>uwxvz</td>
<td>14,x</td>
<td>14,x</td>
<td>14,x</td>
<td>14,x</td>
<td>14,x</td>
</tr>
</tbody>
</table>

**Notes:**

- construct shortest path tree by tracing predecessor nodes
- ties can exist (can be broken arbitrarily)
Dijkstra’s algorithm: another example

<table>
<thead>
<tr>
<th>Step</th>
<th>N'</th>
<th>D(v),p(v)</th>
<th>D(w),p(w)</th>
<th>D(x),p(x)</th>
<th>D(y),p(y)</th>
<th>D(z),p(z)</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>u</td>
<td>2,u</td>
<td>5,u</td>
<td>1,u</td>
<td>∞</td>
<td>∞</td>
</tr>
<tr>
<td>1</td>
<td>ux</td>
<td>2,u</td>
<td>4,x</td>
<td>2,x</td>
<td>∞</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>uxy</td>
<td>2,u</td>
<td>3,y</td>
<td>4,y</td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>uxyv</td>
<td>2,u</td>
<td>3,y</td>
<td>4,y</td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>uxyvw</td>
<td>2,u</td>
<td>3,y</td>
<td>4,y</td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>uxyvwz</td>
<td>2,u</td>
<td>3,y</td>
<td>4,y</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Network Layer  4-11
Dijkstra’s algorithm: example (2)

Resulting shortest-path tree from u:

Resulting forwarding table in u:

<table>
<thead>
<tr>
<th>destination</th>
<th>link</th>
</tr>
</thead>
<tbody>
<tr>
<td>v</td>
<td>(u,v)</td>
</tr>
<tr>
<td>x</td>
<td>(u,x)</td>
</tr>
<tr>
<td>y</td>
<td>(u,x)</td>
</tr>
<tr>
<td>w</td>
<td>(u,x)</td>
</tr>
<tr>
<td>z</td>
<td>(u,x)</td>
</tr>
</tbody>
</table>
Dijkstra’s algorithm, discussion

Algorithm complexity: n nodes
- each iteration: need to check all nodes, w, not in N
- n(n+1)/2 comparisons: $O(n^2)$
- more efficient implementations possible: $O(n\log n)$

Oscillations possible:
- e.g., link cost = amount of carried traffic
Chapter 4: Network Layer

4.1 Introduction
4.2 Virtual circuit and datagram networks
4.3 What’s inside a router
4.4 IP: Internet Protocol
   - Datagram format
   - IPv4 addressing
   - ICMP
   - IPv6

4.5 Routing algorithms
   - Link state
   - Distance Vector
   - Hierarchical routing

4.6 Routing in the Internet
   - RIP
   - OSPF
   - BGP

4.7 Broadcast and multicast routing
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 \( x \)
Bellman-Ford example

Clearly, \( d_v(z) = 5 \), \( d_x(z) = 3 \), \( d_w(z) = 3 \)

B-F equation says:

\[
d_u(z) = \min \{ c(u,v) + d_v(z), c(u,x) + d_x(z), c(u,w) + d_w(z) \}
\]

\[
= \min \{ 2 + 5, 1 + 3, 5 + 3 \} = 4
\]

Node that achieves minimum is next hop in shortest path ➔ forwarding table
Distance Vector Algorithm

- $D_x(y)$ = estimate of least cost from $x$ to $y$
  - $x$ maintains distance vector $D_x = [D_x(y) : y \in N]$
- node $x$:
  - knows cost to each neighbor $v$: $c(x,v)$
  - maintains its neighbors’ distance vectors. For each neighbor $v$, $x$ maintains $D_v = [D_v(y) : y \in N]$
Distance vector algorithm (4)

**Basic idea:**
- from time-to-time, each node sends its own distance vector estimate to neighbors
- when x receives new DV estimate from neighbor, it updates its own DV using B-F equation:

\[ D_x(y) \leftarrow \min_v \{ c(x,v) + D_v(y) \} \quad \text{for each node } y \in N \]

- under minor, natural conditions, the estimate \( D_x(y) \) converge to the actual least cost \( d_x(y) \)
Distance Vector Algorithm (5)

Iterative, asynchronous:
  each local iteration caused by:
  - local link cost change
  - DV update message from neighbor

Distributed:
  - each node notifies neighbors only when its DV changes
    - neighbors then notify their neighbors if necessary

Each node:

1. wait for (change in local link cost or msg from neighbor)
2. recompute estimates
3. if DV to any dest has changed, notify neighbors
D_x(y) = \min\{c(x,y) + D_y(y), c(x,z) + D_z(y)\} = \min\{2+0, 7+1\} = 2

D_x(z) = \min\{c(x,y) + D_y(z), c(x,z) + D_z(z)\} = \min\{2+1, 7+0\} = 3
\[ D_x(y) = \min\{c(x, y) + D_y(y), c(x, z) + D_z(y)\} = \min\{2+0, 7+1\} = 2 \]

\[ D_x(z) = \min\{c(x, y) + D_y(z), c(x, z) + D_z(z)\} = \min\{2+1, 7+0\} = 3 \]
Distance Vector: link cost changes

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

“good news travels fast”

$t_0$: $y$ detects link-cost change, updates its DV, informs its neighbors.

$t_1$: $z$ receives update from $y$, updates its table, computes new least cost to $x$, sends its neighbors its DV.

$t_2$: $y$ receives $z$’s update, updates its distance table. $y$’s least costs do not change, so $y$ does not send a message to $z$. 
**Distance Vector: link cost changes**

**Link cost changes:**
- good news travels fast
- bad news travels slow - “count to infinity” problem!
- 44 iterations before algorithm stabilizes: see text

**Poisoned reverse:**
- If Z routes through Y to get to X:
  - Z tells Y its (Z’s) distance to X is infinite (so Y won’t route to X via Z)
- will this completely solve count to infinity problem?
Distance Vector: link cost increases

node x table

<table>
<thead>
<tr>
<th></th>
<th>x</th>
<th>y</th>
<th>z</th>
</tr>
</thead>
<tbody>
<tr>
<td>x</td>
<td>0</td>
<td>4</td>
<td>5</td>
</tr>
<tr>
<td>y</td>
<td>4</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>z</td>
<td>5</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

node y table

<table>
<thead>
<tr>
<th></th>
<th>x</th>
<th>y</th>
<th>z</th>
</tr>
</thead>
<tbody>
<tr>
<td>x</td>
<td>0</td>
<td>4</td>
<td>5</td>
</tr>
<tr>
<td>y</td>
<td>4</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>z</td>
<td>5</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

node z table

<table>
<thead>
<tr>
<th></th>
<th>x</th>
<th>y</th>
<th>z</th>
</tr>
</thead>
<tbody>
<tr>
<td>x</td>
<td>0</td>
<td>4</td>
<td>5</td>
</tr>
<tr>
<td>y</td>
<td>4</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>z</td>
<td>5</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

Distance Vector: link cost increases

Network Layer 4-24
Same with poison reverse!
Comparison of LS and DV algorithms

Message complexity
- **LS**: with n nodes, E links, $O(nE)$ msgs sent
- **DV**: exchange between neighbors only
  - convergence time varies

Speed of Convergence
- **LS**: $O(n^2)$ algorithm requires $O(nE)$ msgs
  - may have oscillations
- **DV**: convergence time varies
  - may be routing loops
  - count-to-infinity problem

Robustness: what happens if router malfunctions?

**LS**:
- node can advertise incorrect *link* cost
- each node computes only its *own* table

**DV**:
- DV node can advertise incorrect *path* cost
- each node’s table used by others
  - error propagate thru network