ECSE-4730: Computer Communication Networks (CCN)

Network Layer (Routing)

Shivkumar Kalyanaraman: shivkuma@ecse.rpi.edu
Biplab Sikdar: sikdab@rpi.edu
http://www.ecse.rpi.edu/Homepages/shivkuma
Network layer functions - 1

- transport packet from sending to receiving hosts
- network layer protocols in every host, router
three important functions:

- **path determination**: route taken by packets from source to dest. *Routing algorithms*
- **Switching (forwarding)**: move packets from router’s input to appropriate router output
- **call setup**: (optional) some network architectures require router call setup along path before data flows
Q: What service model for “channel” transporting packets from sender to receiver?

- guaranteed bandwidth?
- preservation of inter-packet timing (no jitter)?
- loss-free delivery?
- in-order delivery?
- congestion feedback to sender?

The most important abstraction provided by network layer:

virtual circuit or datagram?
Datagram networks: the Internet model - 1

- no call setup at network layer
- routers: no state about end-to-end connections
  - no network-level concept of “connection”
- packets typically routed using destination host ID
  - packets between same source-dest pair may take different paths
Datagram networks: the Internet model - 2

1. Send data
2. Receive data
Routing

Routing protocol
Goal: determine “good” path (sequence of routers) thru network from source to dest.

• Graph abstraction for routing algorithms:
  • graph **nodes** are routers
  • graph **edges** are physical links
    • link cost: delay, $ cost, or congestion level

“good” path:
  typically means minimum cost path
  other def’s possible
Routing Algorithm classification - 1

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 partial info with neighbors
• “distance vector” algorithms
Routing Algorithm classification - 2

Static or dynamic?

Static:
• routes change slowly over time

Dynamic:
• routes change more quickly
  – periodic update
  – in response to link cost changes
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 routing table for that node
  – iterative: after k iterations, know least cost path to k dest.’s
A Link-State Routing Algorithm - 2

Notation:

- **c(i,j):** link cost from node i to j. cost infinite if not direct neighbors

- **D(v):** current value of cost of path from source to dest. V

- **p(v):** predecessor node (neighbor of v) along path from source to v

- **N:** set of nodes whose least cost path definitively known
### Dijkstra’s algorithm: example

<table>
<thead>
<tr>
<th>Step</th>
<th>start N</th>
<th>D(B),p(B)</th>
<th>D(C),p(C)</th>
<th>D(D),p(D)</th>
<th>D(E),p(E)</th>
<th>D(F),p(F)</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>A</td>
<td>2,A</td>
<td>5,A</td>
<td>1,A</td>
<td>infinity</td>
<td>infinity</td>
</tr>
<tr>
<td>1</td>
<td>AD</td>
<td>2,A</td>
<td>4,D</td>
<td>2,D</td>
<td>infinity</td>
<td>infinity</td>
</tr>
<tr>
<td>2</td>
<td>ADE</td>
<td>2,A</td>
<td>3,E</td>
<td></td>
<td></td>
<td>4,E</td>
</tr>
<tr>
<td>3</td>
<td>ADEB</td>
<td></td>
<td>3,E</td>
<td></td>
<td></td>
<td>4,E</td>
</tr>
<tr>
<td>4</td>
<td>ADEBC</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>4,E</td>
</tr>
<tr>
<td>5</td>
<td>ADEBCF</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

![Graph](image_url)
Dijsktra’s Algorithm

1 **Initialization:**
2 \( N = \{A\} \)
3 for all nodes \( v \)
4 \( \text{if } v \text{ adjacent to } A \)
5 \( \text{then } D(v) = c(A,v) \)
6 \( \text{else } D(v) = \text{infty} \)
7
8 **Loop**
9 find \( w \) not in \( N \) such that \( D(w) \) is a minimum
10 add \( w \) to \( N \)
11 update \( D(v) \) for all \( v \) adjacent to \( w \) and not in \( N \):
12 \( D(v) = \min( D(v), D(w) + c(w,v) ) \)
13 /* new cost to \( v \) is either old cost to \( v \) or known
14 shortest path cost to \( w \) plus cost from \( w \) to \( v \) */
15 **until all nodes in** \( N \)
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
Distance Vector Routing Algorithm - 1

**iterative:**
- continues until no nodes exchange info.
- *self-terminating*: no “signal” to stop

**asynchronous:**
- nodes need *not* exchange info/iterate in lock step!

**distributed:**
- each node communicates *only* with directly-attached neighbors
Distance Vector Routing Algorithm - 2

Distance Table data structure
- each node has its own
- row for each possible destination
- column for each directly-attached neighbor to node
- example: in node X, for dest. Y via neighbor Z:

\[ D_{(Y,Z)}^X = \text{distance from } X \text{ to } Y, \text{ via } Z \text{ as next hop} \]
\[ = c(X,Z) + \min_Z \{ D_{(Y,w)} \} \]
Distance table: example

<table>
<thead>
<tr>
<th>Destination via</th>
<th>A</th>
<th>B</th>
<th>D</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>1</td>
<td>14</td>
<td>5</td>
</tr>
<tr>
<td>B</td>
<td>7</td>
<td>8</td>
<td>5</td>
</tr>
<tr>
<td>C</td>
<td>6</td>
<td>9</td>
<td>4</td>
</tr>
<tr>
<td>D</td>
<td>4</td>
<td>11</td>
<td>2</td>
</tr>
</tbody>
</table>

- $D^E(C,D) = c(E,D) + \min_w \{D^D(C,w)\}$
  - $= 2+2 = 4$

- $D^E(A,D) = c(E,D) + \min_w \{D^D(A,w)\}$
  - $= 2+3 = 5$ (loop!)

- $D^E(A,B) = c(E,B) + \min_w \{D^B(A,w)\}$
  - $= 8+6 = 14$ (loop!)
Distance table gives routing table

<table>
<thead>
<tr>
<th></th>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>1</td>
<td>14</td>
<td>5</td>
<td></td>
</tr>
<tr>
<td>B</td>
<td>7</td>
<td>8</td>
<td>5</td>
<td></td>
</tr>
<tr>
<td>C</td>
<td>6</td>
<td>9</td>
<td>4</td>
<td></td>
</tr>
<tr>
<td>D</td>
<td>4</td>
<td>11</td>
<td>2</td>
<td></td>
</tr>
</tbody>
</table>

Outgoing link to use, cost:

<table>
<thead>
<tr>
<th></th>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>A,1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>B</td>
<td></td>
<td>D,5</td>
<td></td>
<td></td>
</tr>
<tr>
<td>C</td>
<td></td>
<td></td>
<td>D,4</td>
<td></td>
</tr>
<tr>
<td>D</td>
<td></td>
<td></td>
<td></td>
<td>D,4</td>
</tr>
</tbody>
</table>
**Iterative, asynchronous**: each local iteration caused by:
- local link cost change
- message from neighbor: its least cost path change from neighbor

**Distributed:**
- each node notifies neighbors *only* when its least cost path to any destination changes
  - neighbors then notify their neighbors if necessary
Each node:

- wait for (change in local link cost of msg from neighbor)
- recompute distance table
- if least cost path to any dest has changed, notify neighbors
At all nodes, $X$:

1. Initialization:
2. for all adjacent nodes $v$:
3. $D^X_\{(*,v)\} = \infty$ /* the * operator means "for all rows" */
4. $D^X_\{(v,v)\} = c(X,v)$
5. for all destinations, $y$
6. send $\min_{w} D^X_\{(y,w)\}$ to each neighbor /* $w$ over all $X$'s neighbors */
Distance Vector Algorithm - 2

8 loop
9 wait (until I see a link cost change to neighbor V
10 or until I receive update from neighbor V)
11
12 if (c(X,V) changes by d)
13 /* change cost to all dest's via neighbor v by d */
14 /* note: d could be positive or negative */
15 for all destinations y: D(X,y,V) = D(X,y,V) + d
16
17 else if (update received from V wrt destination Y)
18 /* shortest path from V to some Y has changed */
19 /* V has sent a new value for its \( \min_{w} DV(Y,w) \) */
20 /* call this received new value is "newval" */
21 for the single destination y: D(X,Y,V) = c(X,V) + newval
22
23 if we have a new \( \min_{w} D(X,Y,w) \)for any destination Y
24 send new value of \( \min_{w} D(X,Y,w) \) to all neighbors
25
26 forever
Distance Vector Algorithm: example - 1

\[ D^X(Y,Z) = c(X,Z) + \min_w \{D^Z(Y,w)\} \]
\[ = 7 + 1 = 8 \]

\[ D^X(Z,Y) = c(X,Y) + \min_w \{D^Y(Z,w)\} \]
\[ = 2 + 1 = 3 \]
Distance Vector: link cost changes - 1

Link cost changes:
node detects local link cost change
updates distance table (line 15)
if cost change in least cost path, notify neighbors (lines 23,24)

“good news travels fast”
Distance Vector: link cost changes - 2

Link cost changes:
good news travels fast
bad news travels slow - “count to infinity” problem!

algorithm continues on!
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?
Comparison of LS and DV algorithms - 1

Message complexity

- **LS**: with n nodes, E links, O(nE) msgs sent each
- **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
Comparison of LS and DV algorithms - 1

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
Internet AS Hierarchy

Intra-AS border (exterior gateway) routers

Inter-AS interior (gateway) routers
Intra-AS Routing

• Also known as Interior Gateway Protocols (IGP)

• Most common IGPs:
  – RIP: Routing Information Protocol
  – OSPF: Open Shortest Path First
  – IGRP: Interior Gateway Routing Protocol (Cisco propr.)
RIP (Routing Information Protocol) - 1

- Distance vector algorithm
- Included in BSD-UNIX Distribution in 1982
- Distance metric: # of hops (max = 15 hops)

- Distance vectors: exchanged every 30 sec via Response Message (also called advertisement)
- Each advertisement: route to up to 25 destination nets
### RIP
(Routing Information Protocol) - 2

<table>
<thead>
<tr>
<th>Destination Network</th>
<th>Next Router</th>
<th>Num. of hops to dest.</th>
</tr>
</thead>
<tbody>
<tr>
<td>w</td>
<td>A</td>
<td>2</td>
</tr>
<tr>
<td>y</td>
<td>B</td>
<td>2</td>
</tr>
<tr>
<td>z</td>
<td>B</td>
<td>7</td>
</tr>
<tr>
<td>x</td>
<td>--</td>
<td>1</td>
</tr>
<tr>
<td>....</td>
<td>....</td>
<td>....</td>
</tr>
</tbody>
</table>

**Routing table in D**
RIP: Link Failure and Recovery

If no advertisement heard after 180 sec -->
neighbor/link declared dead
– routes via neighbor invalidated
– new advertisements sent to neighbors
– neighbors in turn send out new advertisements (if tables changed)
– link failure info quickly propagates to entire net
– poison reverse used to prevent ping-pong loops (infinite distance = 16 hops)
RIP Table processing - 1

- RIP routing tables managed by application-level process called route-d (daemon)
- advertisements sent in UDP packets, periodically repeated
RIP Table processing - 2

Router: giroflee.eurocom.fr

<table>
<thead>
<tr>
<th>Destination</th>
<th>Gateway</th>
<th>Flags</th>
<th>Ref</th>
<th>Use</th>
<th>Interface</th>
</tr>
</thead>
<tbody>
<tr>
<td>127.0.0.1</td>
<td>127.0.0.1</td>
<td>UH</td>
<td>0</td>
<td>26492</td>
<td>lo0</td>
</tr>
<tr>
<td>192.168.2.</td>
<td>192.168.2.5</td>
<td>U</td>
<td>2</td>
<td>13</td>
<td>fa0</td>
</tr>
<tr>
<td>193.55.114.</td>
<td>193.55.114.6</td>
<td>U</td>
<td>3</td>
<td>58503</td>
<td>le0</td>
</tr>
<tr>
<td>192.168.3.</td>
<td>192.168.3.5</td>
<td>U</td>
<td>3</td>
<td>25</td>
<td>qaa0</td>
</tr>
<tr>
<td>224.0.0.0</td>
<td>193.55.114.6</td>
<td>U</td>
<td>3</td>
<td>0</td>
<td>le0</td>
</tr>
<tr>
<td>default</td>
<td>193.55.114.129</td>
<td>UG</td>
<td>0</td>
<td>143454</td>
<td></td>
</tr>
</tbody>
</table>

Three attached class C networks (LANs)
Router only knows routes to attached LANs
Default router used to “go up”
Route multicast address: 224.0.0.0
Loopback interface (for debugging)
OSPF (Open Shortest Path First)

- “open”: publicly available
- Uses Link State algorithm
  - LS packet dissemination
  - Topology map at each node
  - Route computation using Dijkstra’s algorithm
- OSPF advertisement carries one entry per neighbor router
- Advertisements disseminated to entire AS (via flooding)