Network Layer: Routing and Addressing

Sandhya Dwarkadas
Department of Computer Science
University of Rochester

Assignments

- Project 1: Web Proxy Server
  - DUE OCT 3

Internet Architecture

Bottom-up:
- **physical**: electromagnetic signals “on the wire”
- **link**: data transfer between neighboring network elements
  - encoding, framing, error correction, access control for shared links
- **network**: host-to-host connectivity
  - routing, addressing
- **transport**: host-host data transport
  - reliable data transport, congestion control, flow control
- **application**: anything you want to do on computer networks

Network Layer Function

- transport packet from sending to receiving hosts
  - **routing**: determine a path from source to dest and route packets along the path
  - **addressing**: uniquely identify each node in the network
- network connecting devices
  - called “routers”
  - participate in network protocols
- links
  - connect adjacent hosts, routers
Routing Principles

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

Goal: determine "good" path (sequence of routers) thru network from source to dest.

“good” path:
- typically means minimum cost path

Routing Algorithm Classification

Global information:
- all routers have complete topology, link cost info
- "link state" algorithm

Decentralized:
- router knows connected neighbors, link costs to neighbors
- exchange of info with neighbors to learn remote parts of the network, may take many learning rounds
- "distance vector" algorithm

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
Dijkstra’s algorithm

Initialization:
1. $N' = \{u\}$
2. for all nodes $v$
3. if $v$ adjacent to $u$
4. then $D(v) = c(u,v)$
5. else $D(v) = \infty$
6. Loop
7. find $w$ not in $N'$ such that $D(w)$ is a minimum
8. add $w$ to $N'$
9. update $D(v)$ for all $v$ adjacent to $w$ and not in $N'$:
10. \[ D(v) = \min(D(v), D(w) + c(w,v)) \]
11. new cost to $v$ is either old cost to $v$ or known
12. until all nodes in $N'$

Notes:
- construct shortest path tree by tracing predecessor nodes
- ties can exist (can be broken arbitrarily)

Network Layer: Control Plane: 5-9

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>0</td>
<td>u</td>
<td>7, u</td>
<td>5, u</td>
<td>\infty</td>
<td>\infty</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>uw</td>
<td>6, w</td>
<td>5, u</td>
<td>11, w</td>
<td>14, x</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>uwx</td>
<td>6, w</td>
<td>11, w</td>
<td>14, x</td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>uwxy</td>
<td>10, y</td>
<td>14, x</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>uwxyz</td>
<td>12, y</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Network Layer: Control Plane: 5-10
**Dijkstra’s algorithm: another example**

**Step** | **N'** | **D(v),p(v)** | **D(w),p(w)** | **D(x),p(x)** | **D(y),p(y)** | **D(z),p(z)**
--- | --- | --- | --- | --- | --- | ---
0 | u | 2,u | 5,u | 1,u | ∞ | ∞
1 | ux | 2,u | 4,x | 2,x | ∞ | 5,u
2 | uxy | 2,u | 3,y | 4,y | 3,y | 3,y
3 | uxyv | 2,u | 3,y | 4,y | 4,y | 4,y
4 | uxyvw | 2,u | 3,y | 4,y | 4,y | 4,y
5 | uxyvwz | 2,u | 3,y | 4,y | 4,y | 4,y

* Check out the online interactive exercises for more examples: [http://gaia.cs.umass.edu/kurose_ross/interactive/](http://gaia.cs.umass.edu/kurose_ross/interactive/)

**Dijkstra’s Algorithm: Complexity**

**Algorithm complexity:** n nodes, e links
- Each iteration: need to check all nodes, v, not in N
  \( \Rightarrow n^2(n-1)/2 \) checks: O(n²)
- Update 1-more-hop paths: O(e)
- Total: O(n²+e), or O(n²)
- Using Fibonacci heap to find minimum distance node
  \( \Rightarrow O(n \log n + e) \)

**Dijkstra’s Algorithm: Stability**

**Oscillations possible:**
- e.g., link cost = amount of carried traffic

**Solutions:**
- asynchronous (at different time) adjustments across routers
- stable cost metric (independent of routing policy)
Network Routing

“Link state” routing
- Dijkstra’s algorithm – efficient approach to calculate least cost routes
- all routers need complete topology, link cost info
  ⇒ costly (or impossible) to acquire such information in large networks

Decentralized routing: distributed, asynchronous, iterative
- router only needs to know physically-connected neighbors, link costs to neighbors
  ⇒ learn more by info exchanges between neighbor routers

Distance Vector Routing

Routing table (at each host): the next hop for each destination in the network
- Distance vector routing:
  - routing table can be derived from the distance vector at each node
  - distance vectors can be maintained in a decentralized fashion

Distance vector algorithm

Bellman-Ford equation

let
\[ d_x(y) := \text{cost of least-cost path from } x \text{ to } y \]
then
\[ d_x(y) = \min_v \{ c(x,v) + d_v(y) \} \]

node achieving minimum is next hop in shortest path, used in forwarding table
Distance vector algorithm

- $D_x(y) = \text{estimate of least cost from } x \text{ to } y$
  - $x$ 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 $D_v = [D_v(y): y \in N]$

key 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:
  $$D_x(y) \leftarrow \min_v \{c(x, v) + D_v(y)\} \text{ for each node } y \in N$$
- under minor, natural conditions, the estimate $D_x(y)$ converge to the actual least cost $d_x(y)$

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

Network Layer: Control Plane

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>2</td>
<td>7</td>
</tr>
<tr>
<td>y</td>
<td>inf</td>
<td>inf</td>
<td>inf</td>
</tr>
<tr>
<td>z</td>
<td>inf</td>
<td>inf</td>
<td>inf</td>
</tr>
</tbody>
</table>

Network Layer: Control Plane
**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 $z$.

* Check out the online interactive exercises for more examples: [http://gaia.cs.umass.edu/kurose_ross/interactive/](http://gaia.cs.umass.edu/kurose_ross/interactive/)
Distance Vector: Link Cost Changes

A pathological case:
- recursive distance vector updates
- applies only to link cost increase
  - bad news settles slowly

Algorithm continues on!

Distance vector: link cost changes

**link cost changes:**
- node detects local link cost change
- **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?

Routing Loops

- In packet switching networks, each node computes its own routing table independently
- What if?
  - Y’s next hop to X is Z;
  - and Z’s next hop to X is Y.
- Routing loops
  - during recursive distance vector updates
  - routing mis-behaviors
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

The Internet Network Layer

**Transport layer**: TCP, UDP

**Routing protocols**: RIP, OSPF, BGP

**ICMP protocol**: error reporting, router “signaling”

Network layer

physical layer

IP Addressing: Introduction

- **IP address**: 32-bit identifier for each host, router interface
- **Interface**: connecting point into each data link
  - A router typically has multiple interfaces
  - A host often has single interface

ADDRESSING
**IP Network and Hierarchical Addressing**

- What's an IP network?
  - can physically reach each other without intervening router (intervening switches?)

- IP address:
  - network part (high order bits); host part (low order bits)
  - devices with same network part of IP address are in the same IP network

**IP Addresses: Original Standard**

Given notion of “network”, let’s re-examine IP addresses:

"class"-based addressing:

<table>
<thead>
<tr>
<th>Class</th>
<th>Network</th>
<th>Host</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>0/24</td>
<td></td>
</tr>
<tr>
<td>B</td>
<td>10/8</td>
<td></td>
</tr>
<tr>
<td>C</td>
<td>110/16</td>
<td></td>
</tr>
<tr>
<td>D</td>
<td>1110/11</td>
<td></td>
</tr>
</tbody>
</table>

multicast address

| A          | 1.0.0.0 to 127.255.255.255 |
| B          | 128.0.0.0 to 191.255.255.255 |
| C          | 192.0.0.0 to 223.255.255.255 |
| D          | 224.0.0.0 to 239.255.255.255 |

**IP Addressing: CIDR**

- "Class"-based addressing:
  - inefficient use of address space, address space exhaustion
  - e.g., class B network allocated enough addresses for 65K hosts, even if only 2K hosts in that network

- CIDR: classless addressing
  - network portion of address of arbitrary length
  - address format: a.b.c.d/x, where x is # bits in network portion

```
 00010000 00010111 00010000 00000000
network part host part
```

200.23.16.0/23

**IP Address Depletion (DHCP)**

Address depletion ⇒ 32-bit address space soon to be used up.

Observation: not every host is online at a given time.

DHCP: allow host to dynamically obtain its IP address from network server when it joins network
- can renew its lease on address in use
- allows reuse of addresses (only hold address while connected)
IP Address Depletion (NAT)

Observations:
- A lot of traffic is local
- Although IP addresses are few, possible (IPaddr, port) tuples are more abundant and they can identify communication endpoint

Disclaimer

- Parts of the lecture slides are adapted from and copyrighted by James Kurose and Keith Ross and from those by Prof. Kai Shen. The slides are intended for the sole purpose of instruction of computer networks at the University of Rochester. All copyrighted materials belong to their original owner(s).