CS 557
ARPANet Routing Algorithms

An Overview of the New Routing Algorithm for the ARPANET
J. McQuillan, I. Richer, and E. Rosen, 1979

The Revised ARPANET Routing Metric
Atul Khanna and John Zinky, 1989

Spring 2013
Routing Algorithm Basics

- Routing algorithms view the network as a graph
- Problem: find lowest cost path between two nodes
- Factors
  - static: topology
  - dynamic: load
  - policy
Two Main Approaches

- **Link State Protocols (Today)**
  - New ARPANET Routing Algorithm
  - Revised New ARPANET Routing Algorithm
  - OSPF

- **Distance Vector Protocols (Thurs)**
  - Original ARPANET Routing Algorithm
  - RIP
Basic Steps

Each node assumed to know state of links to its neighbors

- **Step 1**: Each node broadcasts its state to all other nodes
- **Step 2**: Each node locally computes shortest paths to all other nodes from global state
Building Blocks

• Reliable broadcast mechanism
  – flooding
  – sequence number issues
• Shortest path tree (SPT) algorithm
  – Dijkstra’s SPT algorithm
• Metric
  – Cost assigned to each link
  – Rules for varying the cost
Periodically, each node creates a Link state packet containing:

- Node ID
- List of neighbors and link cost
- Sequence number
- Time to live (TTL)

Node outputs LSP on all its links.
Reliable Flooding

When node i receives LSP from node j:
• If LSP is the most recent LSP from j that i has seen so far, i saves it in database and forwards a copy on all links except link LSP was received on.
• Otherwise, discard LSP.
Sequence Number Space Issues

- Problem: sequence number may wrap around
- Solution: treat space as circular, continue after wrap around:
  - A is less than B if
    - A<B and B-A < N/2, or
    - A>B and A-B > N/2
Problem: Router Failure

• A failed router and comes up but does not remember the last sequence number it used before it crashed

• New LSPs may be ignored if they have lower sequence number
One Solution: LSP Aging

• Nodes periodically decrement age (TTL) of stored LSPs
• LSPs expire when TTL reaches 0
  – LSP is re-flooded once TTL = 0
• Rebooted router waits until all LSPs have expired
• Trade-off between frequency of LSPs and router wait after reboot
SPT Algorithm (Dijkstra)

SPT = \{a\}

for all nodes \(v\)
    if \(v\) adjacent to \(a\) then \(D(v) = \text{cost}(a, v)\)
    else \(D(v) = \text{infinity}\)

Loop
    find \(w\) not in SPT, where \(D(w)\) is min
    add \(w\) in SPT
    for all \(v\) adjacent to \(w\) and not in SPT
        \(D(v) = \min(D(v), D(w) + C(w, v))\)

until all nodes are in SPT
Example

<table>
<thead>
<tr>
<th>step</th>
<th>SPT</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>~</td>
<td>~</td>
</tr>
</tbody>
</table>
Example

<table>
<thead>
<tr>
<th>step</th>
<th>SPT</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>~</td>
<td>~</td>
</tr>
<tr>
<td>1</td>
<td>AD</td>
<td>2, A</td>
<td>4, D</td>
<td></td>
<td>2, D</td>
<td>~</td>
</tr>
</tbody>
</table>
### Example

![Graph Diagram]

<table>
<thead>
<tr>
<th>step</th>
<th>SPT</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>~</td>
<td>~</td>
</tr>
<tr>
<td>1</td>
<td>AD</td>
<td>2, A</td>
<td>4, D</td>
<td>2, D</td>
<td>~</td>
<td>~</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>
</tbody>
</table>
Example

<table>
<thead>
<tr>
<th>step</th>
<th>SPT</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>~</td>
<td>~</td>
</tr>
<tr>
<td>1</td>
<td>AD</td>
<td>2, A</td>
<td>4, D</td>
<td>2, D</td>
<td>~</td>
<td>~</td>
</tr>
<tr>
<td>2</td>
<td>ADE</td>
<td>2, A</td>
<td>3, E</td>
<td></td>
<td>4, E</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>ADEB</td>
<td>3, E</td>
<td></td>
<td></td>
<td>4, E</td>
<td></td>
</tr>
</tbody>
</table>
### Example

<table>
<thead>
<tr>
<th>Step</th>
<th>SPT</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>~</td>
<td>~</td>
</tr>
<tr>
<td>1</td>
<td>AD</td>
<td>2, A</td>
<td>4, D</td>
<td></td>
<td>2, D</td>
<td>~</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>3, E</td>
<td></td>
<td></td>
<td>4, E</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>ADEBC</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>4, E</td>
</tr>
</tbody>
</table>
Example

```
<table>
<thead>
<tr>
<th>step</th>
<th>SPT</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>~</td>
<td>~</td>
</tr>
<tr>
<td>1</td>
<td>AD</td>
<td>2, A</td>
<td>4, D</td>
<td></td>
<td>2, D</td>
<td>~</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>3, E</td>
<td></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>
```
Link State Algorithm

Flooding:
1) Periodically distribute link-state advertisement (LSA) to neighbors
   - LSA contains delays to each neighbor
2) Install received LSA in LS database
3) Re-distribute LSA to all neighbors

Path Computation
1) Use Dijkstra’s shortest path algorithm to compute distances to all destinations
2) Install <destination, nexthop> pair in forwarding table
Link State Characteristics

- With consistent LSDBs, all nodes compute consistent loop-free paths
- Limited by Dijkstra computation overhead, space requirements
- Can still have transient loops

Packet from C->A may loop around BDC
[KZ89] Main Points

• **Objective:**
  – Devise a *new metric* that limits routing oscillations and poor path selection under heavy load.

• **Approach:**
  – Modify the metric for assigning a link cost.

• **Contributions:**
  – An example of link-state routing.
  – An example of why adaptive metrics are problematic.
Components of Revised ARPANET Routing

• Routing Algorithm Components
  – Every router learns the state of every link in the network
    (e.g. a link state algorithm)
  – Link state information exchanged via flooding
  – Shortest Path Algorithm for computing distances
  – Link state includes a “metric” for each link

• Link Metric Component
  – Describes the current state of the link in terms of delay,
    bandwidth, congestion, etc.
  – This is what the current paper focuses on changing
Router B learns:
link (B,C)=1
link (B,A)=1
link (C,D)=1
link (D,X)=1
link (A,D)=7
Computes shortest path to X is B,C,D,X
Sets NextHop(X)=C
Failure of Link D-X

Router D learns:
- link (B,C)=1
- link (B,A)=1
- link (C,D)=1
- link (D,X)=infinite

Computes shortest path to X none

Update from A to D says:
- link (B,A)=1
- link (A,D)=7

No loop forms and no counting to infinity
Some Challenges of Link-State Routing

• High Storage at cost at each router:
  – Must learn the full network topology.

• High computation cost at each router:
  – Use flooding to exchange state of every link
  – Re-run shortest path algorithm after each change
    • Paper notes you don’t need to rerun the shortest path algorithm if an link not in the tree increases its metric.
How Do You Assign the Link Metric?

• Original Solution:
  link metric = actual link delay.

• Link delay is defined as:
  processing delay + propagation delay +
  transmission delay + queuing delay

• First three are independent of traffic
  – Processing: (roughly) how fast is the router CPU
  – Propagation: (line length)/(line speed)
  – Transmission: (packet size)/(line bandwidth)

• Queuing Delay: varies with traffic load
Light vs. Heavy Traffic

- Metric = processing + prop + transmission + queue
- In light traffic,
  - First 3 are fixed, queuing is roughly 0
  - Thus link metric is basically fixed
- In heavy traffic,
  - First 3 are fixed, queuing creates dependencies
  - Increase in queue => increase in metric
  - Increase in metric => route including link longer
  - Longer route => switch to shorter route
  - Switch => decrease in traffic
  - Decrease in traffic => queue reduced
  - Now repeat cycle....
Adaptive Metric Problem

- Assume all traffic from cloud A to cloud B initially uses A1-B1
- All routers see A2-B2 as a shorter path and now all traffic shifts to A2-B2.
Internet Metrics Today

• Paper suggests approach to control queuing part of link delay.
  – Metric = proc + prop + transmission + queue
  – Note that TCP is also adapting to the queue

• Today OSPF (link-state) metrics are typically static.
  – Metric based on proc + prop + bandwidth
  – Let TCP adapt to congested links

• Fundamental event is a link up/link down.
Conclusions

• Link-State Routing is one alternative to Distance Vector.
  – Routers learn the full topology
  – In other words, every router learns the state (up/down) of every link in the network.
  – No counting to infinity loops

• Moving toward large-scale routing…
  – Distance vector counting to infinity is a problem
  – Link-state topology knowledge is a problem
  – Neither appropriate for today’s global routing.