Dynamic Routing Algorithms

Mario Baldi

Politecnico di Torino
(Technical University of Turin)
http://staff.polito.it/mario.baldi
Copyright Notice

This set of transparencies, hereinafter referred to as slides, is protected by copyright laws and provisions of International Treaties. The title and copyright regarding the slides (including, but not limited to, each and every image, photography, animation, video, audio, music and text) are property of the authors specified on page 1.

The slides may be reproduced and used freely by research institutes, schools and Universities for non-profit, institutional purposes. In such cases, no authorization is requested.

Any total or partial use or reproduction (including, but not limited to, reproduction on magnetic media, computer networks, and printed reproduction) is forbidden, unless explicitly authorized by the authors by means of written license.

Information included in these slides is deemed as accurate at the date of publication. Such information is supplied for merely educational purposes and may not be used in designing systems, products, networks, etc. In any case, these slides are subject to changes without any previous notice. The authors do not assume any responsibility for the contents of these slides (including, but not limited to, accuracy, completeness, enforceability, updated-ness of information hereinafter provided).

In any case, accordance with information hereinafter included must not be declared.

In any case, this copyright notice must never be removed and must be reported even in partial uses.
Forwarding and Routing

1. Routing (proactive routing)

2. Forwarding (on-the-fly routing)
Proactive Routing

- Realized independently of actual traffic
- Determine destinations reachable by each node
- Computation of best route
- Routing information is computed and stored locally on each node
  - Routing table
  - Forwarding table
On-the-fly Routing

- Realized when handling each packet
- Based on local information
  - Routing table
Proactive Routing Algorithms

- Objectives
  - Coherence of the routing
    - Avoid loop, black holes, etc...
  - Automation of the algorithm

- Optimization
  - Chose preferred coherent route

- Metric
  - Unit by which optimization is measured

Example of Coherence: “always send data clockwise”
Metrics

- They express the quality of the path
- Also known as costs
- Comparison of two paths: according to the lower cost

Examples
- Number of hops
- Delay
- Load on links
Optimal Path Computation

- Combination of criteria combined together with certain weights
  - Minimizing number of hops and delay
- There may be contrasting criteria
  - Maximizing the use of the network and minimizing delay
- There might be constraints
  - Shortest route with delay below D
Routing Algorithm Characteristics

- Multi objective optimization $\rightarrow$ high complexity
- May or may not require the knowledge of the network
  - Topology
  - Metrics
Routing Algorithm Characteristics

- Simplicity
  - Implementation
  - Processing requirements
- Memory Requirements
- Robustness and Adaptability
  - Fault Detection
  - Auto-stabilization
Routing Algorithm Characteristics

- Stability
  - It should be active only in case of topological and metrics variations

- Fairness
  - Avoid to favor/hamper particular nodes
Transient State

- Routing information propagation
  - Topology change distribution takes time

- Inconsistency
  - Some ISs may operate according to the updated situation, while others still know the old situation
Black Hole and Routing Loop

**Black Hole**

- **A** to **B**: B,C,D
- **B** to **C**: A, C
- **C** to **D**: A,B, D
- **D** to **A**: A,B, C

**Routing Loop**

- **A** to **B**: B,C,D
- **B** to **C**: A, D
- **C** to **D**: A,B, D
- **D** to **A**: A,B, C
- **C** to **A**: Blue arrow
- **B** to **A**: Orange arrow

**D?**

To:D

To:C
Routing Algorithm Classification

- Non-adaptive algorithms (static)
  - Fixed Directory routing (static routing)
  - (Selective) flooding and derivates

- Adaptive algorithms (dynamic)
  - Centralized Routing
  - Isolated Routing
  - Distributed Routing
    - Distance Vector
    - Link State
Fixed Directory Routing

Routing Table of B
A ←
C, D →

Routing Table of D
A, B, C ←

Routing Table of C
A, B ←
D →

Routing Table of A
B, C, D →
Static and dynamic routing

The only way to the “rest of the world”

Routing information exchange

Core
Dynamic routing

No routing information

Edge
Static routing
Distributed Routing

- Joined advantages of Isolated and Centralized routing
  - Centralized: routers co-operate in exchanging connectivity information
  - Isolated: routers cooperate and there is not a main router

- Two main algorithms
  - Distance Vector (DV) or Bellman-Ford
  - Link State (LS)
Distance Vector Basic Principles

2. I know the targets A,0 and E,1 (from port A1)

3. I know the targets A at cost 0 and E at cost 1

1. I know the target E,0

4. I know the targets C,0, A, 1 (from port C1), F,1 (from port C2), and E,2 (from ports C1 and C2 at the same cost)

2. I know the targets F,0 ed E,1 (from port F1)

3. I know the targets F at cost 0 and E at cost 1
Information in Distance Vector

- There is a series of reachable destinations
- For each targets, it is reachable
  - In particular direction
  - Through a certain X node
    - The one sending the distance vector
  - Along a path with a certain cost
    - Advertised cost combined with the cost of the link through which DV is received
Distance Vector

Set of pairs target – cost

- It is generated independently by each node
- Basically, it is an extract of the routing table
- Each node inserts the list of the known targets and the cost of the best path from this node to the target
- Each node memorizes the DV of the neighbors, plus the local information of the node itself

![Diagram of a network with nodes A, B, C, D, E and their connections]

Memory of A

<table>
<thead>
<tr>
<th>Loc (A)</th>
<th>DV (B)</th>
<th>DV (D)</th>
</tr>
</thead>
<tbody>
<tr>
<td>A, 0</td>
<td>A, 1</td>
<td>A, 1</td>
</tr>
<tr>
<td>B, 0</td>
<td>B, 1</td>
<td></td>
</tr>
<tr>
<td>C, 1</td>
<td>C, 2</td>
<td></td>
</tr>
<tr>
<td>D, 1</td>
<td>D, 0</td>
<td></td>
</tr>
<tr>
<td>E, 1</td>
<td></td>
<td>E, 1</td>
</tr>
</tbody>
</table>

Local information of the node
Fusion and generation of DV

IS A:

<table>
<thead>
<tr>
<th>Loc (A)</th>
<th>DV (B)</th>
<th>DV (D)</th>
<th>ROUT. TABLE (A)</th>
<th>DV (A)</th>
</tr>
</thead>
<tbody>
<tr>
<td>A, 0</td>
<td>A, 1</td>
<td>A, 1</td>
<td>A, local, 0</td>
<td>A, 0</td>
</tr>
<tr>
<td>B, 0</td>
<td>B, 1</td>
<td>B, 1</td>
<td>B, 1</td>
<td>B, 1</td>
</tr>
<tr>
<td>C, 1</td>
<td>C, 2</td>
<td>C, A1</td>
<td>C, 2</td>
<td>C, 2</td>
</tr>
<tr>
<td>D, 1</td>
<td>D, 0</td>
<td>D, A2</td>
<td>D, 1</td>
<td>D, 1</td>
</tr>
<tr>
<td>E, 1</td>
<td>E, 1</td>
<td>E, A2</td>
<td>E, 2</td>
<td>E, 2</td>
</tr>
</tbody>
</table>
Distance Vector: algorithm

1. **Main process**
   - Receiving a new DV
     - It memorizes the DV in the memory
     - Combine the new DV with the other memorized DV
   - Give your DV to adjacent ISs
     - Waits for TIMEOUT
   - Reliability: it avoids using the link-up signal which may not be available

2. **Failure of a link**
   - It deletes DV coming from that link
   - Combine together remaining DVs
     - Change Routing table?
       - YES
       - NO

3. **DV not received (timeout)**
   - It deletes the timeout DV
Example: Cold Start

A emits the DV

B and C emit the DV

All emit the DV
Example: failure of a link

- Procedure:
  - Void DV coming from A1
  - Maintained other DVs valid
  - Activated fusion process

- Efficiency
  - Node A receives the new RT without exchanging DV with adjacent nodes

**IS A:**

<table>
<thead>
<tr>
<th>Loc (A)</th>
<th>DV (B)</th>
<th>DV (D)</th>
<th>ROUT. TABLE (A)</th>
<th>DV (A)</th>
</tr>
</thead>
<tbody>
<tr>
<td>A, 0</td>
<td>A, 1</td>
<td>A, 1</td>
<td>A, local, 0</td>
<td>A, 0</td>
</tr>
<tr>
<td>E, 0</td>
<td>B, 1</td>
<td>B, 1</td>
<td>B, A2, 2</td>
<td>B, 2</td>
</tr>
<tr>
<td>C, 1</td>
<td>C, 2</td>
<td>C, 2</td>
<td>C, A2, 3</td>
<td>C, 3</td>
</tr>
<tr>
<td>D, 1</td>
<td>D, 0</td>
<td>D, A2, 1</td>
<td>D, A2, 1</td>
<td>D, 1</td>
</tr>
<tr>
<td>E, 1</td>
<td>E, 1</td>
<td>E, A2, 2</td>
<td>E, A2, 2</td>
<td>E, 2</td>
</tr>
</tbody>
</table>
Topologic knowledge

- The Distance Vector does not recognize topology
  - DV of C is the same in both cases
    - In the first on it will originate a Count to infinity
    - In the second case it will not
Count to Infinity

IS B:

<table>
<thead>
<tr>
<th>Loc (B)</th>
<th>DV (A)</th>
<th>DV (C)</th>
<th>RT (B)</th>
</tr>
</thead>
<tbody>
<tr>
<td>B, 0</td>
<td>A, 0</td>
<td>A, 2</td>
<td>A, B2, 3</td>
</tr>
<tr>
<td>B, 1</td>
<td>B, 1</td>
<td>B, loc</td>
<td>B, loc, 1</td>
</tr>
<tr>
<td>C, 2</td>
<td>C, 0</td>
<td>C, B2</td>
<td>C, B2, 1</td>
</tr>
</tbody>
</table>

B emette il DV

Loc (B) | DV (C) | RT (B) |
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>A, 0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>B, 1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>C, 2</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

IS C:

<table>
<thead>
<tr>
<th>Loc (C)</th>
<th>DV (B)</th>
<th>RT (C)</th>
</tr>
</thead>
<tbody>
<tr>
<td>A, 1</td>
<td>B, 1</td>
<td>A, C1, 2</td>
</tr>
<tr>
<td>B, 0</td>
<td>B, 0</td>
<td>B, C1, 1</td>
</tr>
<tr>
<td>C, 1</td>
<td>C, 1</td>
<td>C, loc, 0</td>
</tr>
</tbody>
</table>

C emette il DV

Loc (C) | DV (B) | RT (C) |
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>A, 3</td>
<td>B, 1</td>
<td>A, C1, 4</td>
</tr>
<tr>
<td>B, 0</td>
<td>B, 0</td>
<td>B, C1, 1</td>
</tr>
<tr>
<td>C, 1</td>
<td>C, 1</td>
<td>C, loc, 0</td>
</tr>
</tbody>
</table>

Count to Infinity!
Solutions

- Additions / modifications to the original Distance Vector algorithm

- Most popular algorithms
  - Split Horizon
  - Path Hold Down
  - Route Poisoning

- Partial solutions
  - Distance Vector: it presents the basic problem linked to lack of topology knowledge
  - Further techniques overload the protocol and tend to not making it 100% reliable
Advantages

- Simplicity
Shortcomings

- Exponential worst case complexity and convergence time
  - Between $O(n^2)$ and $O(n^3)$;
- Convergence limited by slower links and routers
- Complex troubleshooting
  - Behavior on big and complex networks is difficult to understand and foresee
    - Nodes do not have a network map
Shortcomings

- It does not work properly on large and complex highly meshed networks
  - Routing loops might be caused by particular topology changes
- “Infinity” threshold limits the use of such algorithm to small networks
  - Usually a few hops (e.g., 15)
  - It may be changed through management
Link State principle

1. I, node E, know links E-A, E-F

2. I, node A, know links A-E and A-C

3. I, node F, know links F-E and F-C
Link State Basic Principles

- Information on the state of each link to neighboring nodes
  - Link state
- Each node sends a link state to all other nodes
- Each node builds a network map
  - The same on all nodes
  - Information is exchanged cooperatively
Link State Basic Elements

- Neighbor Greetings
  - Hello Packets
  - Bringing up Adjacencies
- Link State (Packet)
- Selective Flooding of link state
- Route calculation algorithm
  - Dijkstra (or Shortest Path First)
Neighbor Greetings (Hello)

- It is necessary to recognize the existence of adjacent nodes
  - Periodic operation, very similar to sending of Distance Vector
  - High periodicity for recognizing variations on adjacencies in a reasonable time

![Network diagram showing adjacencies between nodes E, C, A, and F.]

1. I am E
2. Node E belongs to my adjacencies
2. Node E belongs to my adjacencies
Link State (Packet)

- **Set of adjacencies** (*adjacency – cost*)
  - Link: characterized by source node – target node
    - Source node: implicit in LS sender
  - Generated independently by each node
  - Each node inserts the list of adjacencies and their cost
  - Each node memorizes LS of all the other nodes in the network
  - Fast spreading (no local processing of the LS)

```
<table>
<thead>
<tr>
<th></th>
<th>LS (A)</th>
<th>LS (B)</th>
<th>LS (C)</th>
<th>LS (D)</th>
<th>LS (E)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>B, 1</td>
<td>A, 1</td>
<td>B, 1</td>
<td>A, 1</td>
<td>B, 1</td>
</tr>
<tr>
<td></td>
<td>D, 1</td>
<td>C, 1</td>
<td>E, 1</td>
<td>B, 1</td>
<td>C, 1</td>
</tr>
<tr>
<td>Memory of A</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
```

A network diagram is also shown with nodes A, B, C, D, E and their connections.
Link State Database

Memory of A

<table>
<thead>
<tr>
<th>LS (A)</th>
<th>LS (B)</th>
<th>LS (C)</th>
<th>LS (D)</th>
<th>LS (E)</th>
</tr>
</thead>
<tbody>
<tr>
<td>B, 1</td>
<td>A, 1</td>
<td>B, 1</td>
<td>A, 1</td>
<td>B, 1</td>
</tr>
<tr>
<td>D, 1</td>
<td>C, 1</td>
<td>E, 1</td>
<td>B, 1</td>
<td>C, 1</td>
</tr>
<tr>
<td>D, 1</td>
<td>E, 1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>E, 1</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Each node has the network graph:
- the nodes represent the routers
- the arcs (with related cost) represent the links

Link State Database

<table>
<thead>
<tr>
<th>A</th>
<th>B/1</th>
<th>D/1</th>
</tr>
</thead>
<tbody>
<tr>
<td>B</td>
<td>A/1</td>
<td>C/1</td>
</tr>
<tr>
<td>C</td>
<td>B/1</td>
<td>E/1</td>
</tr>
<tr>
<td>D</td>
<td>A/1</td>
<td>B/1</td>
</tr>
<tr>
<td>E</td>
<td>B/1</td>
<td>C/1</td>
</tr>
</tbody>
</table>

Database: same on all network nodes
Flooding

- Link State Packets
  - Should be sent in “broadcast” to the whole network
  - Should be received without variations by any network node
- Real protocols: implement a form of Selective flooding
  - The routing table may not yet be used to access (in unicast) all nodes in the network
Dijkstra algorithm

- Dijkstra algorithm
  - Used for calculating the Spanning Tree (tree of minimum-cost paths having the node as the root) of the graph

- Operation
  - The node “nearest” to the root is determined
  - If the node exists:
    - This path is inserted into the routing table
    - The operation is repeated, selecting the node at an immediately superior cost
  - Otherwise: it ends when a path for all nodes is found
Dijkstra algorithm

- **Defining**
  - 1 root node, the node calculating the algorithm
  - 1 PATH set of nodes with the best path
  - 1 set of TEMP nodes without a path

- **Algorithm**
  - The root node is inserted in PATH
  - All the nodes adjacent to the previous node are inserted in TEMP
  - Node N with the lowest-cost path in TEMP is passed to PATH
  - For each adjacent V of the passed node N
    - If V is not in TEMP yet, it is inserted now
    - If V already exists, its cost is analyzed towards the root \( D(\text{root}, \text{N}) + D(\text{N}, \text{V}) \) and if this is lower than the previous one in TEMP cost and link of that node in TEMP are updated
Example
Dijkstra Algorithm Complexity

- Algorithmic complexity
  - Dijkstra: $L \times \log N$
  - Bellman-Ford (DV): $N \times N$
    - $L$ is the number of links, $N$ is the number of nodes
    - Normally, $N$ and $L$ are of the same order of magnitude
  - Dijkstra is far more scalable than Bellman-Ford
Generating Link State

- In theory: when the router notes a variation in local topology (adjacencies)
  - It realizes it has a new neighbor
  - The cost towards a neighbor has changed
  - It has lost connectivity towards a neighbor that was previously accessible

- Practically: periodic generation
  - Increase of reliability
Link State Features

- Rapid convergence
  - LSs spread quickly without intermediate processing
- It rarely generates loops
  - It is able to determine and interrupt them easily
    - It has a network map
- Simple to understand and troubleshoot
  - All nodes have identical databases
Link State Features

- High scalability
  - Limited traffic generated by link state distribution
DV vs. LS: Neighbor Discovery

- LS uses Neighbor Greetings protocols
  - However small messages
- DV knows neighbors through periodic exchange of distance vectors
DV vs. LS: Information Exchange

- LSs contain information on all neighbors and are sent to all IS
  - Smaller size
  - Neighbor greeting messages have a small size

- DVs contain information on all systems and are sent to neighbors
  - Large size
Distance Vector vs. Link State

- Network map
  - LS ISs exchange information on the network
    - Topology
    - Metrics
    - All have the same information
  - DV ISs exchange routing tables
    - ISs do not know then network topology
    - Trust calculations of other IS
DV vs. LS: Implementation Complexity

- Distance Vector has very simple implementation
  - Merging of vectors
- Link State includes many distinct components
  - LS flooding is very complicated
  - Route computing algorithm
DV vs. LS: Memory Requirements

- Dijkstra: $I \times A$
  - Each LS contains $A$ adjacencies
  - Each IS stores LSs from all other ISs
- Bellman-Ford
  - $A \times E$
  - Each DV contains $E$ end-systems
- Normally $E >> I$