Distributed Routing Algorithms

• Assumptions
  - Network is modeled as a connected, undirected graph
  - Nodes represent both destinations and relays
    (No distinction between routers and hosts)
  - Each node has a unique ID (natural number)
  - Edges are communication channels (bidirectional)
    • Edges may also have an associated "length" or "cost"
      - \( d[i,j] = \) length of edge between \( i \) and \( j \) (\( \infty \) if no edge exists)
    • Each node knows the lengths of its incident edges
    • Each node knows the identity of nodes it is connected to
  - Nodes may not communicate except via channels
Problem Statement

• At convergence, for every node $i$ and $j$:
  – $D_{i[j]}$ is the length of the shortest path from $i$ to $j$
  – $h_{i[j]}$ is the next hop on the shortest path from $i$ to $j$

• Algorithm converges if channels are not broken
Bellman-Ford Algorithm

• Based on R. Bellman's well-known principle of optimality, which in this context, says:

  If the **first step** on the **shortest path from i to j** is **k**
  then **the rest** is the **shortest path from k to j**

(Suppose not, i.e. there is some shorter path from k to j. Then a shorter path from i to i to j exists, namely that first step followed by that path!.)
Distributed Bellman-Ford Algorithm

- **Initialize:** For each i and j: \( D_i[j] := d[i,j] \)

- **At each i, iterate forever:**
  \[
  \forall j: D_i[j] = \min_k d[i,k] + D_k[j]
  \]
  or:
  
  for each j:
  
  for each neighbor k:
  
  if \((d[i,k] + D_k[j] < D_i[j])\)
  
  \{ \ h_i[j] := k; \ D_i[j] := d[i,k] + D_k[j] \ }

- **Nodes exchange their \( D_i[j] \) tables periodically**
  - Vector of distances ⇒ "**Distance Vector**" Algorithm
Simplified Bellman-Ford Example

To C:
Next Hop: -
Distance: \(\infty\)
via A: \(\infty\)
via B: \(\infty\)
via D: \(\infty\)
via E: \(\infty\)

To C: Next Hop: -
Distance: \(\infty\)
via A: \(\infty\)
via D: \(\infty\)
via C: \(\infty\)

To C: Next Hop: -
Distance: 0

To C: Next Hop: -
Distance: \(\infty\)
via C: \(\infty\)
via D: \(\infty\)
Simplified Bellman-Ford Example

To C:
- Next Hop: -
- Distance: ∞
  via A: ∞
  via B: ∞
  via D: ∞
  via E: ∞

From: C
To: C
Dist: 0
Simplified Bellman-Ford Example

To C:
Next Hop: -
Distance: ∞
via B: ∞
via D: ∞

To C:
Next Hop: -
Distance: ∞
via A: ∞
via B: ∞
via E: ∞

To C:
Next Hop: -
Distance: ∞
via A: ∞
via D: ∞
via C: ∞

To C:
Next Hop: -
Distance: ∞
via C: ∞
via D: ∞

To C:
Next Hop: -
Distance: 0
Simplified Bellman-Ford Example

- From B to C:
  - Next Hop: C
  - Distance: 1
  - via A: ∞
  - via D: ∞
  - via C: 1

- From E to C:
  - Next Hop: C
  - Distance: 2
  - via C: 2
  - via D: ∞

From B:
- Next Hop: C
- Distance: 1
- via B: ∞
- via D: ∞
- via C: 1

To C:
- Next Hop: C
- Distance: 1

To A:
- Distance: ∞
- via B: ∞
- via D: ∞

To D:
- Distance: ∞
- via A: ∞
- via B: ∞
- via C: 1

To E:
- Distance: ∞
- via B: ∞
- via D: ∞
- via C: 1

To C:
- Distance: 0
**Simplified Bellman-Ford Example**

- **To C:**
  - Next Hop: B
  - Distance: 3
    - via B: 3
    - via D: ∞

- **To C:**
  - Next Hop: -
  - Distance: ∞
    - via A: ∞
    - via B: ∞
    - via E: ∞

- **To C:**
  - Next Hop: C
  - Distance: 1
    - via A: ∞
    - via D: ∞
    - via C: 1

- **To C:**
  - Next Hop: C
  - Distance: 2
    - via C: 2
    - via D: ∞

- **To C:**
  - Next Hop: -
  - Distance: 0
**Simplified Bellman-Ford Example**

From: A
To: C
Next Hop: B
Distance: 3
via B: 3
via D: ∞

To C:
Next Hop: C
Distance: 1
via A: ∞
via D: ∞
via C: 1

From: D
To: C
Next Hop: -
Distance: 0

To C:
Next Hop: C
Distance: 2
via C: 2
via D: ∞
Simplified Bellman-Ford Example

To C:
Next Hop: B
Distance:  3
via B: 3
via D: $\infty$

To C:
Next Hop: A
Distance:  $\infty$
via A: $\infty$
via B: 2
via E: 4

To C:
Next Hop: C
Distance:  1
via A: $\infty$
via D: $\infty$
via C: 1

To C:
Next Hop: C
Distance:  2
via C: 2
via D: $\infty$

To C:
Next Hop: -
Distance:  0
Simplified Bellman-Ford Example

To C:
Next Hop: B
Distance: 3
via B: 3
via D: 3

To C:
Next Hop: B
Distance: 2
via A: 4
via B: 2
via E: 4

To C:
Next Hop: C
Distance: 1
via A: 5
via D: 3
via C: 1

To C:
Next Hop: -
Distance: 0
"Bad News Travels Slowly"

To C:
Next Hop: B
Distance: 3
via B: 3
via D: 3

To C:
Next Hop: B
Distance: 2
via A: 4
via B: 2
via E: 4

To C:
Next Hop: C
Distance: 1
via A: 5
via D: 3
via C: 1

To C:
Next Hop: -
Distance: 0

To C:
Next Hop: C
Distance: 2
via C: 2
via D: 4
"Bad News Travels Slowly"

To C:
Next Hop: B
Distance: 3
via B: 3
via D: 3

To C:
Next Hop: B
Distance: 2
via A: 4
via B: 2
via E: 4

To C:
Next Hop: D
Distance: 3
via A: 5
via D: 3
via C: ∞

To C:
Next Hop: -
Distance: 0

To C:
Next Hop: C
Distance: 2
via C: 2
via D: 4
"Bad News Travels Slowly"

To C:
Next Hop: D
Distance: 3
via B: 5
via D: 3

To C:
Next Hop: B
Distance: 4
via A: 4
via B: 4
via E: 4

From: D
To: C
Dist: 4

To C:
Next Hop: D
Distance: 3
via A: 5
via D: 3
via C: \infty

To C:
Next Hop: C
Distance: 2
via C: 2
via D: 4

To C:
Next Hop: -
Distance: 0
"Bad News Travels Slowly"

To C:
Next Hop: D
Distance: 5
via B: 5
via D: 5

To C:
Next Hop: B
Distance: 4
via A: 4
via D: 5
via C: \infty

To C:
Next Hop: -
Distance: 0

To C:
Next Hop: C
Distance: 2
via C: 2
via D: 6
"Bad News Travels Slowly"

To C:
Next Hop: D
Distance: 5
via B: 7
via D: 5

To C:
Next Hop: E
Distance: 4
via A: 6
via B: 6
via E: 4

To C:
Next Hop: D
Distance: 5
via A: 7
via D: 5
via C: ∞

To C:
Next Hop: C
Distance: 2
via C: 2
via D: 6

To C:
Next Hop: -
Distance: 0
Distance-Vector Algorithms

• Advantage: Simple
• Disadvantage: Convergence time after topology/cost change depends on graph & costs!
  - May take a long time to detect changes & stabilize
  - Especially when the network becomes disconnected
    "Counting to Infinity" problem: Cost just keeps increasing
    Meanwhile, packets loop!
    • Partial solutions: "split horizon", "poison reverse" (see text)

• Disadvantage: Routing messages can be expensive
  - Dump entire forwarding table in each message!
Link-State Algorithms

• Basic Idea:
  – Nodes exchange topology information
    • Each announces the state of its attached links
    • Link-state announcements
  – Link-state announcements are broadcast throughout the network
    • Flooding mechanism implements a broadcast function
  – Each node builds a graph model of the network
    • Collects every other node's link-state announcements
  – Each node runs Dijkstra's all-nodes shortest-path algorithm on its graph
    • Requirement: all nodes have the same graph model!
Flooding Mechanism

• Every node forwards every new flooded message to all of its neighbors
  – "New" = not already in the node's database

• Challenge: distinguishing new from old
  – Solution: sequence numbers on LSAs
    • But: What about wrapping sequence numbers?

• Challenge: lost messages
  – Solution: acknowledge received flooded LSAs
  – Each node retransmits until ack received on each link
Simplified Link-State Example

Source: A
Link to: B
Metric: 2
Seq #: 0

Link to: D
Metric: 1
Seq #: 0

A's Link-State DB
A-B 2 0
A-D 1 0

B's Link-State DB
A-B 2 0
B-C 1 0
B-D 1 0

C's Link-State DB
B-C 1 0
C-E 2 0

D's Link-State DB
A-D 2 0
B-D 1 0
D-E 2 0

E's Link-State DB
C-E 2 0
D-E 2 0
Simplified Link-State Example

A's Link-State DB
A-B 2 0
A-D 1 0

B's Link-State DB
A-B 2 0
A-D 1 0
B-C 1 0
B-D 1 0

C's Link-State DB
B-C 1 0
C-E 2 0

D's Link-State DB
A-B 2 0
A-D 1 0
B-D 1 0
D-E 2 0

E's Link-State DB
C-E 2 0
D-E 2 0
Simplified Link-State Example

A's Link-State DB
- A-B 2 0
- A-D 1 0

B's Link-State DB
- A-B 2 0
- A-D 1 0
- B-C 1 0
- B-D 1 0

C's Link-State DB
- A-B 2 0
- A-D 1 0
- B-C 1 0
- C-E 2 0

D's Link-State DB
- A-B 2 0
- A-D 1 0
- B-D 1 0
- D-E 2 0

E's Link-State DB
- A-B 2 0
- A-D 1 0
- C-E 2 0
- D-E 2 0
Simplified Link-State Example

A's Link-State DB
- A-B: 2, 0
- A-D: 1, 0

B's Link-State DB
- A-B: 2, 0
- A-D: 1, 0
- B-C: 1, 0
- B-D: 1, 0

C's Link-State DB
- A-B: 2, 0
- A-D: 1, 0
- B-C: 1, 0
- C-E: 2, 0

D's Link-State DB
- A-B: 2, 0
- A-D: 1, 0
- B-D: 1, 0
- D-E: 2, 0

E's Link-State DB
- A-B: 2, 0
- A-D: 1, 0
- C-E: 2, 0
- D-E: 2, 0

Source: D
Link to: B
Metric: 1
Seq #: 0

Link to: A
Metric: 1
Seq #: 0

Link to: E
Metric: 2
Seq #: 0
Simplified Link-State Example

A's Link-State DB
A-B  2   0
A-D  1   0
B-D  1   0
D-E  2   0

B's Link-State DB
A-B  2   0
A-D  1   0
B-C  1   0
B-D  1   0
D-E  2   0

C's Link-State DB
A-B  2   0
A-D  1   0
B-C  1   0
C-E  2   0

D's Link-State DB
A-B  2   0
A-D  1   0
B-D  1   0
D-E  2   0

E's Link-State DB
A-B  2   0
A-D  1   0
B-D  1   0
C-E  2   0
D-E  2   0
Simplified Link-State Example
...after B, C, & E flood their LSAs

A's Link-State DB
- A-B 2 0
- A-D 1 0
- B-C 1 0
- B-D 1 0
- C-E 2 0
- D-E 2 0

B's Link-State DB
- A-B 2 0
- A-D 1 0
- B-C 1 0
- B-D 1 0
- C-E 2 0
- D-E 2 0

C's Link-State DB
- A-B 2 0
- A-D 1 0
- B-C 1 0
- B-D 1 0
- C-E 2 0
- D-E 2 0

D's Link-State DB
- A-B 2 0
- A-D 1 0
- B-C 1 0
- B-D 1 0
- C-E 2 0
- D-E 2 0

E's Link-State DB
- A-B 2 0
- A-D 1 0
- B-C 1 0
- B-D 1 0
- C-E 2 0
- D-E 2 0
Link-State Algorithms

• Advantages:
  – Nodes send information about only their attached links
  – Fast convergence after change

• Disadvantages:
  – Each node "knows" the whole topology!
  – Dijkstra running time grows with topology
  – Flooding consumes bandwidth