Network Layer V

Dmitri Loguinov
Texas A&M University

April 17, 2024
Chapter 4: Roadmap

4.1 Introduction
4.2 Virtual circuit and datagram networks
4.3 What’s inside a router
4.4 IP: Internet Protocol
4.5 Routing algorithms
   - Link state
   - Distance Vector
   - Hierarchical routing
4.6 Routing in the Internet
4.7 Broadcast and multicast routing
Graph Abstraction

Graph: $G = (V, E)$
$V =$ set of routers $= \{u, v, w, x, y, z\}$
$E =$ set of links $= \{(u,v), (u,x), (u,w), (v,x), (v,w), (x,w), (x,y), (w,y), (w,z), (y,z)\}$
Graph Abstraction: Costs

- \( c(x,y) = \text{cost of link } (x,y) \)
  - E.g., \( c(w,z) = 5 \)
- Cost options:
  - Could always be 1
  - Could be inversely related to bandwidth or be proportional to congestion
  - Physical distance/delay

Cost of path \((x_1, x_2, x_3, \ldots, x_p)\) = \( c(x_1, x_2) + c(x_2, x_3) + \ldots + c(x_{p-1}, x_p) \)

Question: What’s the least-cost path between \(u\) and \(z\)?

Routing algorithms find least-cost paths
Routing Algorithm Classification

Global or local information?

- **Global**: Routers have complete topology, link cost info
  - “Link state” algorithms
- **Local (decentralized)**: Router knows physically-connected neighbors, link costs to neighbors
  - Iterative process of computation, exchange of info with neighbors
  - “Distance vector” algorithms

Static or dynamic?

- **Static**: Useful when routes change slowly over time
  - Manual or DHCP-based route creation
- **Dynamic**: Routes change more quickly
  - Periodic update in response to link cost changes
Chapter 4: Roadmap

4.1 Introduction
4.2 Virtual circuit and datagram networks
4.3 What’s inside a router
4.4 IP: Internet Protocol
4.5 Routing algorithms
  - Link state
  - Distance Vector
  - Hierarchical routing
4.6 Routing in the Internet
4.7 Broadcast and multicast routing
Simple Link-State Routing Algorithm

Dijkstra’s algorithm

- Entire network topology and link costs known
  - Accomplished via “link state broadcast”
  - Eventually, 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 $k$ closest destinations

**Notation:**

- $c(x,y)$: link cost from $x$ to $y$
  - Cost is $\infty$ if not direct neighbors
- $D(v)$: current estimate of the cost from source to destination $v$
- $p(v)$: predecessor of $v$ along the least-cost path back to source
- $F$: set of closest nodes whose least-cost path has been finalized (i.e., known for a fact)
**Dijkstra’s Algorithm**

**Initialization:**
- \( F = \{u\}, \ D(u) = 0 \)
- for all nodes \( v \neq u \)
  - if \( v \) is adjacent to \( u \)
    - \( D(v) = c(u,v) \)
  - else
    - \( D(v) = \infty \)

\[
\text{do } \left\{ \begin{array}{l}
\text{find node } i \text{ not in } F \text{ such that } D(i) \text{ is minimum} \\
\text{add } i \text{ to } F \\
\text{for all } j \text{ adjacent to } i \text{ and not in } F : \\
D(j) = \min(D(j), D(i) + c(i,j)) \\
\text{/* new cost to } j \text{ is either old cost to } j \text{ or known shortest path cost to } i \text{ plus cost from } i \text{ to } j */
\end{array} \right. \\
\text{while (not all nodes in } F) \\
\]
**Dijkstra’s Algorithm: Example**

<table>
<thead>
<tr>
<th>Step</th>
<th>$F$</th>
<th>$D(v), p(v)$</th>
<th>$D(w), p(w)$</th>
<th>$D(x), p(x)$</th>
<th>$D(y), p(y)$</th>
<th>$D(z), p(z)$</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>$u$</td>
<td>$2, u$</td>
<td>$5, u$</td>
<td>$1, u$</td>
<td>$\infty$</td>
<td>$\infty$</td>
</tr>
<tr>
<td>1</td>
<td>$ux$</td>
<td>$2, u$</td>
<td>$4, x$</td>
<td>$\infty$</td>
<td>$2, x$</td>
<td>$\infty$</td>
</tr>
<tr>
<td>2</td>
<td>$uxy$</td>
<td>$2, u$</td>
<td>$3, y$</td>
<td>$\infty$</td>
<td>$4, y$</td>
<td>$\infty$</td>
</tr>
<tr>
<td>3</td>
<td>$uxyw$</td>
<td>$\infty$</td>
<td>$3, y$</td>
<td>$\infty$</td>
<td>$4, y$</td>
<td>$\infty$</td>
</tr>
<tr>
<td>4</td>
<td>$uxywz$</td>
<td>$\infty$</td>
<td>$\infty$</td>
<td>$\infty$</td>
<td>$4, y$</td>
<td>$\infty$</td>
</tr>
</tbody>
</table>

![Graph Diagram](image_url)
Dijkstra’s Algorithm Discussion

Algorithm complexity: \( n \) nodes

- Iteration \( k \): need to find min of \((n-k)\) costs, visit \(d_i\) neighbors
- Naïve implementation: \(O(|E|+|V|^2)\) complexity
- Heap-based implementation: \(O(|E|+|V|\cdot\log|V|)\)

Oscillations possible, but only for traffic-dependent cost:
- e.g., Link cost = amount of carried traffic
Chapter 4: Roadmap

4.1 Introduction
4.2 Virtual circuit and datagram networks
4.3 What’s inside a router
4.4 IP: Internet Protocol
4.5 Routing algorithms
   - Link state
   - Distance Vector
   - Hierarchical routing
4.6 Routing in the Internet
4.7 Broadcast and multicast routing
Distance Vector (DV) Algorithm

• Two metrics known to each node $x$
  - Estimate $D_x(y)$ of least cost from $x$ to $y$
  - Link cost $c(x,v)$ to reach $x$’s immediate neighbors
• Each node maintains a distance vector:
  $$\vec{D}_x = \{D_x(y) : y \in V\}$$
• Node $x$ periodically receives from neighbors their distance vectors
  - Thus, $x$ has access to the following for each neighbor $v$
    $$\vec{D}_v = \{D_v(y) : y \in V\}$$
Distance Vector (DV) Algorithm (cont’d)

Basic idea (Bellman-Ford):

• When a node $x$ receives new DV estimate from neighbor $v$, it updates its own DV using the Bellman-Ford equation:

$$D_x(y) \leftarrow \min \{D_x(y), c(x,v) + D_v(y)\}, \forall y \in V$$

• Centralized Bellman Ford requires $O(|V| \cdot |E|)$ time
  - Dijkstra’s algorithm was $O(|V| \cdot \log|V|)$
  - Convergence of decentralized version depends on topology, link weights, update delays, and timing of events

• Bellman Ford advantage – no need for entire graph
Distance Vector (DV) Algorithm (cont’d)

Iterative, asynchronous

Each 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

Each node:

wait for (change in local link cost or msg from neighbor)

recompute estimates

if DV to any dest has changed, notify neighbors
Distance Vector: Link Cost Changes

Link cost changes:
- Node detects local link cost change
- Recalculates distance vector, updates routing info if needed
- If DV changes, notifies neighbors

“good news travels fast”

- Node $y$ detects link-cost change, updates its distance to $x$, and informs its neighbors
- Node $z$ receives $y$’s message and updates its table; computes a new least-cost to $x$ and sends its DV to $x$ and $y$
- Finally, node $y$ receives $z$’s vector and updates its distance table; $y$’s least costs do not change and hence $y$ does not send any messages after that
Distance Vector: Link Cost Changes

Link cost changes:
- Good news travels fast
- Bad news travels slow – “count to infinity” problem!
- 46 iterations before algorithm stabilizes

Poisoned reverse (“split horizon”):
- If \( z \) routes through \( y \) to get to \( x \):
  - \( z \) tells \( y \) that 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

Message complexity

- **LS**: with $n$ nodes & $E$ links, $nE$ msgs sent
- **DV**: exchange between neighbors only
  - Depends on convergence time

Time to Convergence

- **LS**: $|V| \cdot \log|V|$ CPU time + delay to send $nE$ msgs
  - Oscillations (cost = congestion)
- **DV**: convergence time varies
  - May have routing loops
  - Count-to-infinity problem

Robustness: what happens if router malfunctions?

- **LS**:
  - Node can advertise incorrect link cost
  - Affects only a small portion of the graph

- **DV**:
  - DV node can advertise incorrect path cost
  - Each node’s table used by others
  - Errors propagate thru network