The Network Layer: Routing & Addressing

Jasleen Kaur

April 14, 2015

Outline

- Network layer services
- Routing algorithms
  - Least cost path computation algorithms
- Hierarchical routing
  - Connecting networks of networks
- IP Internet Protocol
  - Addressing
  - IPv6
- Routing on the Internet
  - Intra-domain routing
  - Inter-domain routing
Transport protocols provide logical end-to-end delivery because there fundamentally is no notion of a connection (in the Internet).

For this reason routers do not run application or transport-layer protocols. The network layer performs the actual physical transport of packets. (Recall that packets are the unit of data transfer at the network-layer.)

In order to achieve physical transport of packets, the network-layer must perform several functions:

- **Addressing** implies some means of mapping logical addresses to physical entities. In the Internet, the physical entity is either the actual destination or the next-hop router.

- **Path determination** is done on a packet-by-packet basis which is why packets from the same connection may take different paths (routers do not maintain connection state!). Algorithms that compute paths are called routing algorithms.

- **Switching** is an intra-router function.

- **Call setup**: Some network architectures (not the Internet!) require a call setup procedure to be executed along the path from source to destination before any application data is transmitted.
VC networks provide better service but have a significantly more complex network-layer.

IP networks represent an inversion of telephony networks:
- Smart end-systems (computers) with lots of storage and a dumb (stateless) network core.
- This is what enabled internetworking.
- Easy to add new services (since all services are implemented at the edge of the network).

Now we get into the meat of the network layer: routing.

Network-Layer Service Models
Datagram v. Virtual Circuit networks
- IP networks:
  - Data exchanged among computers
  - "Elastic" service, no strict timing requirements
  - "Smart" end systems (computers)
    - Can adapt, perform control, error recovery
    - Simple inside the network, complexity at "edges"
  - Operates over "any" link layer technology
    - Uniform service difficult
    - But interoperation "easy"
  - New services easily added
- ATM Networks
  - Evolved from telephony
    - Human conversation:
      - Strict timing, reliability requirements
    - Need for guaranteed service
  - "Dumb" end systems (telephones)
    - Tremendous complexity inside the network
  - No interoperability with other networks
  - New services require "the network" to be upgraded

The Network Layer: Routing & Addressing
Outline
- Network layer services
- Routing algorithms
  - Least cost path computation algorithms
- Hierarchical routing
  - Connecting networks of networks
- IP Internet Protocol
  - Addressing
  - IPv6
- Routing on the Internet
  - Intra-domain routing
  - Inter-domain routing
**Routing Algorithms**

### Least-cost path computation

- **Goal:** To determine a “good” path through the network from source to destination
- **Graph abstraction for routing algorithms:**
  - Nodes are routers
  - Edges are physical links
  - Edges have a “cost” metric
  - Cost can be delay, monetary cost, level of congestion, etc.

- “Good” path typically means minimum cost path
  - Also shortest path, …
- (But often ISPs define “good” in terms of business models)

---

**Routing Algorithms**

### Taxonomy

- **Global or decentralized?**
  - Global — all routers have complete graph (topology, costs)
    - “Link state” algorithms
  - Decentralized — router knows link costs to physically connected adjacent nodes
    - Run iterative algorithm to exchange information with adjacent nodes
    - ”Distance vector” algorithms

- **Static or dynamic?**
  - Static — routes change slowly over time
  - Dynamic — routes change more quickly
    - Periodic updates, or
    - Updates in response to link outages or cost changes
**Global Routing Algorithms**

*A link-state routing algorithm*

- Uses Dijkstra’s shortest path graph algorithm
- Complete network topology and link costs known at all nodes
  - Accomplished via *link state flooding*
  - All nodes learn the “same” topology and cost data
- Each node computes least cost paths from itself to all other nodes
  - Produces a *routing table* for that node
  - All nodes compute consistent routing tables
- Algorithm complexity:
  - \( N \) nodes (routers) in the network
  - \( N \times (N+1)/2 \) comparisons
  - (More efficient implementations possible)

---

**Link State Routing**

*Dijsktra’s Algorithm*

1. **Initialization:**
   - \( N = \{A\} \)
2. for all nodes \( v \)
3. if \( v \) adjacent to \( A \)
4. then \( D(v) = c(A,v) \)
5. else \( D(v) = \text{infinity} \)
6. \( c(n,m) \) is the cost of the link from \( n \) to \( m \)
7. \( N \) is the set of nodes to which we have computed the minimum cost path
8. **Loop**
9. find node \( w \) not in \( N \) such that \( D(w) \) is a minimum
10. add node \( w \) to \( N \)
11. update \( D(v) \) for all nodes \( v \) adjacent to \( w \) and not in \( N \):
12. \( D(v) = \min( D(v), D(w) + c(w,v) ) \)
13. */ new cost to node \( v \) is either old cost to \( v \) or known */
14. shortest path cost to \( w \) plus cost from \( w \) to \( v */
15. until all nodes in \( N \)
Again, while $D(w)$ represents the minimum cost, for each $w$ we have to store some path information when we fix $D(w)$. — Recording the predecessor of $w$ suffices. — This is because since all nodes have the graph, each can rely on the next node to forward data to $w$ along the same minimum-cost path.

The goal in all of this is to compute a routing table. The table tells us for any given destination, on which outbound link should we forward the packet. — We don’t have to store the entire path because all we are responsible for is ensuring the packet is heading towards its destination. — If all routers have the same network map (graph) they will all agree on the path and hence we only need store the next-hop. — The next-hop is determined by working backwards through the table on the previous slide. From $A$… — The shortest path to $F$ (and $C$) was through $E$. — The shortest path to $E$ was through $D$. — The shortest path to $D$ was through $D$.
Link State Routing

Link State Flooding Algorithm

- The data stored for an edge in the graph (the link between nodes $X$ and $Y$) consists of:
  - Cost from $X$ to $Y$ ($X-Y$) and from $Y$ to $X$ ($Y-X$)
  - A unique timestamp for the last update to each cost
- A node that discovers a change in cost for one of its attached links forwards the update to all adjacent nodes
- A node receiving an update forwards it based on a comparison of the update timestamp and the timestamp on its local data for the link:
  - Update is later (or new): Forward to all adjacent nodes (except sender) and update local data
  - Update is earlier: Send local data back to sender
  - Update is equal: Do nothing

Step 1:
A link fails and nodes C and D notice it.

Step 2:
C and D send the updated link information to all attached neighbors with a timestamp.

Step 3:
A floods the $D \rightarrow C$ update message it received from D and "F floods the $C \rightarrow D$ update message it received from C.

Step 4:
B floods the $C \rightarrow D$ update message it received from C and "E floods the $D \rightarrow C$ update message it received from D.

---

Link State Flooding Algorithm

Example

1. 
2.  
3.  
4.  

At step 4, all nodes have received the updated costs but the flooding continues.

In general these algorithms are hard to think about because of the high degree of parallelism they exhibit.

For example, in reality steps 2 through (at least) 4 occur in parallel.

The key is convince yourself that the algorithm will terminate.
Step 3: A floods the D → C update message it received from D and "C floods the C → D update message it received from C."

Step 4: B floods the C → D update message it received from C and "E floods the D → C update message it received from D."

Step 5: A floods the C → D update message it received from C, "C floods the D → C update message it received from A, (note that C received the same message from E in step 4), "E floods the D → C update message it received from D." (in Step 2).

Step 6: B floods the D → C update message it received from D in Step 2 (note that B received the same message from A in Step 3), "F floods the D → C update message it received from E in Step 4, "D floods the C → D update message it received from B in Step 4.

In this example all traffic is destined for node A.

Once D, C, and B start forwarding, they send out link state update messages with the new values of traffic load on their attached links.

Nodes C and B then decide that the cheapest path to A is clockwise through D.

Links D, C, and B again send out link state update messages with the value of the traffic load on their attached links.

Nodes D, C, and B then decide that the cheapest path to A is counter clockwise through B.

Traffic will oscillate from all traffic going through D to all going through B, to D to B...

The oscillation problem can occur anytime you have load sensitive link costs.

The problem is fixed with random delays to break the synchrony. (Convince yourself that this solves the problem.)