Network Layer: Routing

Shivkumar Kalyanaraman
Rensselaer Polytechnic Institute
shivkuma@ecse.rpi.edu
http://www.ecse.rpi.edu/Homepages/shivkuma

Based in part upon the slides of Prof. Raj Jain
(Ohio State), S. Keshav (Cornell), L. Peterson (Arizona)

Overview

- The network layer problem
- Routing:
  - Forwarding vs switching vs routing
  - Telephony vs data networks
  - Distance vector vs Link State
  - Bellman-Ford vs Dijkstra’s algorithm
  - Addressing issues and Virtual-circuits
- Module: http://links.math.rpi.edu/devmodules/graph_networking

The Network Layer Problem

- Two nodes communicating across a “network of networks”…
- How to transport packets through this maze?

- Ans: Routing.
- We will study heterogeneity and scaling issues later under the heading “internetworking”
**Forwarding**

- Problem: Finding *which output port* packet needs to go to.
  - *Trivial* in the case of a *dual-port* node.
  - Eg: Repeaters or ring topologies
  - *Simple pt-to-pt* transfer if destination *directly-connected*
    - Eg: mesh
  - *Flooding* if destination *logically connected* on a bus.
    - Eg: ethernet
  - *Multi-stage switching* by matching address bit-by-bit
    - Eg: Star topology
  - *Table-lookup* otherwise. Why?
    - Destination address does not have any other coded information.

---

**Routing**

- Problem: sets up a *forwarding table* (also called "routing table") in *routers* and *switch controllers*
  - A node makes a *local next-hop* route choice depending on *global topology*: this is the fundamental problem.
Is routing easy or hard?

Case A
1) Assume each link has equal weight. Is routing easy?
2) What if there were a non-negligible probability of links going down?

Case B
If the numbers above refer to link weights, what is the path (sequence of links) from h to d which has the minimum total weight (shortest path)?

Key problem
- How to make correct local decisions?
  - Each router must know something about global state
- Global state
  - Inherently large
  - Dynamic
  - Hard to collect
- A routing protocol must intelligently summarize relevant information

Requirements
- Consistent routing tables
- Minimize routing table space
  - Fast to look up
  - Less to exchange
- Minimize number and frequency of control messages
- Robustness: avoid
  - Black holes, brown-outs
  - Loops
  - Oscillations
- Find optimal path
Routing is simple, because topology is simple
- 3-level hierarchy, with a fully-connected core (clique)
- AT&T: 135 core switches with nearly 5 million circuits
- LECs may connect to multiple cores

### Telephone network topology

- **Routing:**
  - Directly connect if endpoints are within the same CO.
  - Use one-hop path between COs if call is between COs in the same LEC.
  - Otherwise send call to one of the cores.
  - Only major decision is at the toll switch: one-hop or two-hop path to the destination toll switch.
  - (Why don’t we need longer paths?)

### Telephony routing algorithm

- **Stable load:**
  - Can predict pairwise load throughout the day.
  - Can choose optimal routes in advance.

- **Extremely reliable switches:**
  - Downtime is less than a few minutes per year.
  - Can assume that a chosen route is available.
  - Can’t do this in the Internet.

- **Single organization controls entire core:**
  - Can collect global statistics and implement global changes.

- **Very highly connected network:**
  - Connections require resources (but all need the same).

### Features of telephone network routing

- **Features:**
  - Single organization controls entire core.
  - Can collect global statistics and implement global changes.
  - Very highly connected network.
  - Connections require resources (but all need the same).
The cost of simplicity

- Simplicity of routing a historical necessity
  - No digital equipment/computers in 1890 - only "switches"
- But requires
  - reliability in every component
  - logically fully-connected core
- Can we build an alternative that has same features as the telephone network, but is cheaper because it uses more sophisticated routing?
  - Yes: that is one of the motivations for ATM networks
    - But economics says that 80% of cost is in the local loop!
    - Moreover, many of the software systems assume topology
      - too expensive to change them

Dynamic nonhierarchal routing (DNHR)

- Naive protocol:
  - accept call if a one-hop path is available, else drop
- DNHR
  - divides day into around 10 periods
  - in each period, each toll switch is assigned a primary one-hop path and a list of alternatives (alternate-path idea...)
  - can overflow to alternative if needed
- crankback
- drop call only if all alternate paths are busy
- Problems
  - does not work well if actual traffic differs from prediction
  - there are some simple extensions to DHNR

Data Network Routing Issues

- Unreliable routers, links: Why?
  - Cheap-n-dirty components, little hardware redundancy or backup, heterogeneity in equipment
- Complex load structure:
  - Internet aggregate traffic is possibly self-similar or is not easy to deal with mathematically.
- Large number of organizations with autonomous domains:
  - Can’t implement global changes quickly
- Sparsely interconnected network:
  - Few alternative paths
  - Unlike a clique of toll-switches
  - +ve: No resource reservation for best effort ⇒ flexible
Find the shortest path between node \(a\) and node \(b\).

How did you find the path? Can you outline a method in general one could use in networks like this?

Routing alternatives

- **Random routing**: At every intersection, randomly choose a next-hop
  - **Problem**: infinite looping, inefficient paths
- **Flooding**: send packet to all next-hops, except ones you have visited earlier
  - **Problem**: per-packet broadcast is inefficient
- **AAA-style**: Get a map from the nearest AAA, plot a course from source-to-destination, and follow that.
  - You can use road-signs for emotional satisfaction
  - Knowledge of construction/road work/detours also known
  - Latest: Magellan GPS receivers, Mapquest/Expedia etc
  - This is known as "source-based routing"
  - **Problem**: every packet needs to carry path information

Routing alternatives

- Provide a map at every intersection:
  - These maps should be consistent
  - Find the min-distance path to each destination from that intersection (just like AAA-style)
  - Then, point their next-hop in the right direction
  - Called "link-state routing": because map is maintained in terms of link-states
  - Provide a marker to every destination along with the currently best-known distance to that destination
  - The next-hop points in the min-distance direction
  - Update markers by simply exchanging markers and seeing if there is a new min-distance path per-destination
  - This is known as "distance-vector" routing
**Distance Vector routing**

- “Vector” of distances (signposts) to each possible destination at each router.
- How to find distances?
  - Distance to local network is 0.
  - Look in neighbors’ distance vectors, and add link cost to reach the neighbor.
  - Find which direction yields minimum distance to a particular destination. Turn signpost that way.
  - Keep checking if neighbors change their signposts and modify local vector if necessary.
- And that’s it!
- Called the “Bellman-Ford algorithm”

**Routing Information Protocol (RIP)**

- Uses hop count as metric.
- Tables (vectors) “advertised” to neighbors every 30 s.
- **Counting-to-infinity** problem:
  - Simple configuration A→B→C. If C fails, B needs to update and thinks there is a route through A. A needs to update and thinks there is a route thru B.
  - No clear solution, except to set “infinity” to be small (eg 16).
  - Split horizon: If A’s route to C is thru B, then A advertises C’s route (only to B) as infinity.
- **Slow convergence** after topology change:
  - Due to count to infinity problem.
  - Also information cannot propagate through a node until it recalculates routing info.

**Link State protocols**

- Create a network “map” at each node.
  - For a map, we need inks and attributes (link states), not of destinations and metrics (distance vector).
- 1. Node collects the state of its connected links and forms a “Link State Packet” (LSP).
- 2. Broadcast LSP => reaches every other node in the network.
- 3. Given map, run Dijkstra’s shortest path algorithm => get paths to all destinations.
- 4. Routing table = next hops of these paths.
Dijkstra’s algorithm

- A.k.a “Shortest Path First” (SPF) algorithm.
- Idea: compute shortest path from a “root” node to every other node. “Greedy method”:
  - $P$ is a set of nodes for which shortest path has already been found.
  - For every node “o” outside P, find shortest one-hop path from some node in P.
  - Add that node “o” which has the shortest of these paths to P. Record the path found.
  - Continue till we add all nodes (&paths) to P.

- $P$: (ID, path-cost, next-hop) triples.
  - ID: node id.
  - Path-cost: cost of path from root to node
  - Next-hop: ID of next-hop on shortest path from the root to reach that node
  - $P$: Set of nodes for which the best path cost (and next-hop from root) have been found.

- $T$: (ID, path-cost, next-hop):
  - Set of candidate nodes at a one-hop distance from some node in $P$.
  - Note: there is only one entry per node. In the interim, some nodes may not lie in $P$ or $T$.

- $R$: Routing table: (ID, next-hop) to be created

1. Put root i.e., (myID, 0, 0) in $P$ & (myID,0) to $R$.
2. If node $N$ is just put into $P$, look at $N$’s links (i.e. its LSP).
   - 2a. For each link to neighbor $M$ add cost of the root-to-$N$-path to the cost of the $N$-to-$M$-link (from LSP) to determine a new cost: $C$.
   - 2b. The “next-hop” corresponds to the next-hop ID in $N$’s tuple (or $N$ if $M$ is the root itself): $h$
   - 2c. If $M$ not in $T$ (or $P$) with better path cost, add $(M, C, h)$ to $T$.
3. If $T$ = empty, terminate. Else, move the min-cost triple from $T$ to $P$, and add $(M, h)$ to $R$. Go to step 2.
**Topology dissemination**

- aka LSP distribution
- 1. Flood LSPs on links except incoming link
  - Require at most 2E transfers for n/w with E edges
- 2. Sequence numbers to detect duplicates
  - Why? Routers/links may go down/up
  - Problem: wrap-around => have large seq # space
- 3. Age field (similar to TTL)
  - Periodically decremented after acceptance
  - Zero => discard LSP & request everyone to do so
  - Router awakens => knows that all its old LSPs would have been purged and can choose a new initial sequence number

**Link state vs Distance vector**

- Advantages:
  - More stable (aka fewer routing loops)
  - Faster convergence than distance vector
  - Easier to discover network topology, troubleshoot network.
  - Can do better source-routing with link-state
  - Type & Quality-of-service routing (multiple route tables) possible
- Caveat: With path-vector-type distance vector routing, these arguments don’t hold

**Role of Addresses**

- Address structure required for scalability
  - Why? Routing table sizes, control traffic etc depends upon the number of nodes in the network.
  - By capturing an entire sub-network as a "virtual node", you can reduce the number of "virtual nodes” core routers see.
  - Need hierarchical addressing, and address allocation according to topology for this.
  - Telephony and ATM networks use variable sized, large (upto 20 bytes) addresses.
  - The large address is only carried during signaling
ATM Networks: VCs & Label Switching

- Virtual circuits (VCs): like telephony "circuit", but multiple VCs may be mapped onto physical links
- Label switching: Use 20-byte address during VC-setup, and establish local 32-bit labels
- Packets (cells) then carry only short labels in header...

Summary

- Routing, switching, forwarding
- Telephony routing
- Data networks routing
  - Distance-vector, link-state routing
  - Dijkstra’s algorithm, Bellman-Ford algorithm
- Address and ATM labels