Tutorial 7

Gidon Rosalki

2025-12-11

Notice: If you find any mistakes, please open an issue at `https://github.com/robomarvin1501/notes_networking`

1 What is routing

We have spoken about DHCP, IP, DNS, and ARP, which all aid us in communicating in a LAN. Whenever we wanted to send information beyond the router that is located between our LAN, and the rest of the network, we have stated that this happens, but stopped at the router. Today, we will discuss what the router does.

Routing is the process of selecting a path for the traffic in a network, or between multiple networks. Sometimes packets must cross a few networks to reach the target network, so the relaying of a packet from one network interface to another is called *forwarding*.

In order to route, we need to make a more abstract version of the network. Let us consider the network as a graph $G = (V, E)$. We can turn this into a weighted graph, to represent bandwidth, and so on, but this will not be discussed in this tutorial. We can then run graph based algorithms for finding the shortest path, such as Dijkstra, and Bellman Ford. These are abstracted into two routing algorithms, that then allow us to save the routes in the routing table. This effectively stores the cost of sending information from a local node, to any other destination in the network.

There are 2 different routing algorithms:

- Distributed (decentralised): This uses a similar idea to STP, where each router computes a view of the network on its own, and we hope that everything will converge.
- Global: This assumes that every node knows the complete network topology

2 How to route between L3 networks

2.1 Distance vector

This is a distributed algorithm, based on Bellman Ford. The distance between a node s, to another node y is given as:

$$D_s(y) = \min_{v \in \Gamma(s)} \{c(s, v) + D_v(y)\}$$

Where

- $c(s, v)$ is the current value of the edge from s to v
- $D_V(y)$ is the currently known distance from v to y, based on the table that s has made
- $\Gamma(s)$ is the neighbours of node s

Every node in the network will compute a vector of distances to any other node in the network, using ∞ for distances that are unknown. The vector will continue updating until stabilisation.

The main idea (from the router's perspective): Each node keeps the DV of its neighbours, and if it gets an update from its neighbour, or one of its edges, it

1. Updates the DV
2. Distributes the updated DV to all neighbours

This is repeated until there are no further changes in this router's DV. As we can see, this breaks into two sub sections:

1. Initialisation:
   - Set the DV:
     $$D_s(v) = \begin{cases} c(s, v), & \text{if } v \in \Gamma(s) \\ \infty, & \text{else} \end{cases}$$
   - Set the neighbours DV: $\forall w \in \Gamma(s), \forall v \in V: \ D_w(v) = \infty$
   - Send the DV to all neighbours

2. Update:

    - If some neighbour's DV, **or** one of the edge's weights change:

    $$D_s(y) = \min_{v \in \Gamma(s)} \{c(s,v) + D_v(y)\}$$

    - Resend the DV if anything changed

So, we build a distance table, with our immediate information about our neighbours, and then forward this to our neighbours, who update things accordingly. This keeps happening until everyone's tables have stabilised, at which point it is transferred into a forwarding table, and all will remain stable until the weights change. The changes can be negative (links disconnected, and the like) or even positive (a new link appears in the table).

By doing examples (not shown here), one can see that after a positive change, where network weights are reduced, the algorithm will converge quickly to its stabilised state. However, when there is a negative change, then the routing loop of trying to compute the new distance vectors will continue for a long time, until the nodes establish that there is no shorter paths.
This is called *Count to Infinity*, and will continue until $c(a,b) + D_b(c) \geq c(a,c)$ (Here, a, b are routers, and a link not to a from b has been updated negatively). This can be solved with what is called *poisoned reverse*. Given the nodes a, b, c, a path $a \to b \to c$, then a can report in its DV to b that its distance to c is ∞. This means that b will not try and reroute anything to c through a, which is exactly the problem earlier that caused very slow stabilisation in the network updated with an increased weight. It should however be noted that poisoned reverse does not work in every situation, it will fail when there are loops longer than 2.

2.2 Link state

Link state is a *global* routing algorithm. This means that every node maintains a complete picture of *every* other router in the network, not just its neighbours. This means that every router knows everything about the entire network. The main idea is to iteratively find the shortest path to any destination in the network, employing Dijkstra's algorithm to do this.
Similar to DV, we keep a table of the distances from us (the node s) to every other node in the network. Starting with $c(s,v)$, for every neighbour v of s, and ∞ to every other node that is **not** a neighbour of s. We also keep the predecessor to that destination, the node one before the destination node v, marked as $p_s(v)$. This predecessor node lets us compute the path to the desired node, by recursively building the path from the predecessor nodes, until we reach a node that is a neighbour. In actuality, we would not build the entire route, but rather make note of the neighbour that begins that route. When forwarding a packet to a destination, we simply send it to the neighbour. Each node then simply handles sending the packet to its neighbour. A set of of nodes, N' is also kept, for which we definitely know the shortest paths from s, and in each step, we add a node to N', and update the values of other nodes should the need arise.
In general, the algorithm is as follows, for every node s:

1. Initialisation:

    - Set

    $$D(v) = \begin{cases} c(s,v), & \text{if } v \in \Gamma(s) \\ \infty, & \text{else} \end{cases}$$

    - Set $N' = \{s\}$ and set $\forall v \in \Gamma(s),\ p(v) = s$

2. Loop while $N' \neq N$:

    - Find $w \notin N'$ such that $D(w)$ is minimised
    - Add w into N'
    - $\forall v \in \Gamma(w) : v \notin N'$, if $D(v) > D(w) + c(w,v)$, then

    $$\begin{aligned} D(v) &= D(w) + c(w,v) \\ p(v) &= w \end{aligned}$$

2.3 Comparison and Real Life

Both DV and LS algorithms are implemented, and used in real life. Routing Information Protocol (RIP) is a DV implementation, and Open Shortest Paths First (OSPF) is an LS implementation. DV is usually better for smaller networks, because it takes time to stabilise, which is not a problem in smaller networks. In larger networks, one should use LS. However, DS is a simpler protocol, that is easier for each router to run, so there is some desire to use it.

RIP is based on the hop count (the number of subnets in the path), and it updates every 30 seconds. If no update is received from a neighbour for 180 seconds, then it is marked as unreachable. OSPF is an LS implementation, where each router broadcasts its LS, at least once every 30 minutes, and when a change in the topology is detected.