ECE 333: Introduction to Communication Networks
Fall 2001

Lecture 22: Routing and Addressing I

- Introduction to Routing/Addressing

Lectures 19-21 described the main components of point-to-point networks, i.e. multiplexed transmission lines and switches or routers. These networks usually have a mesh topology with multiple paths between any two hosts. A ***routing algorithm*** determines the specific route that traffic takes through a network. For example, in datagram networks, the routing algorithm determines what information is the routing table at each node in the network. In the next few lectures, we will study problem of routing in more detail. Our focus will mainly be on packet switched data networks.

Routing is one of the main functions of the network layer. Network layer protocols, such as routing, involve every node in the subnet. Protocols at lower layers are generally point-to-point, and protocols at higher layers, e.g. the transport layer, are end-to-end. The basic service the network layer provides the transport layer is the delivery of packets from the source to destination. As with the data link layer, the network layer can offer different variation of this service - for example it may be connection-oriented (as in virtual circuit networks) or connectionless. The network layer could try to ensure some level of reliability or guarantee some level of quality. For example, a circuit switched network provides user end-to-end delivery of data at a fixed rate and with known delay.

The network layer in ATM networks (see Lecture 21) is called the ATM layer. The ATM layer provides a choice of several different services such as a constant rate, connection-oriented, low delay service and a variable-rate unreliable service. Different services are designed to be suitable for different applications, such as voice, data, etc.

In TCP/IP, the IP layer serves as the network layer. It provides a connectionless, unreliable service to the transport layer. In other words, packets are delivered from source to destination but they may not arrive, or if they arrive, they are not guaranteed to be in order. In TCP/IP networks, any additional reliability needed by an application is addressed at the transport layer.

A key trade-off between these two approaches is how much intelligence is put in the subnet and how much is put in the hosts. In TCP/IP the decision was to put as much burden on the host as possible; with ATM more burden is placed on the subnet. In either case, routing is addressed at the network layer.

Routing

We looked at routing when considering bridges and extended LANs in Lecture 19. For transparent bridges, routing was done by forming a spanning tree and sending all packets along this tree. In this case, the routing algorithm is the protocol the bridges use to exchange messages and find a spanning tree. A similar approach can be used in a WAN, but this is generally not a good idea. One reason is that with a spanning tree some links will be unused, while other links will be heavily loaded. In a large network, it is desirable to spread out the load among multiple links to reduce congestion and more efficiently use the network.

The main characteristics desired from a routing algorithm include:

- **Correctness** - deliver packets to their correct destinations.
- **Robustness** - when nodes and/or links fail, the routing algorithm should be able to recover from this.
- **Simplicity** - the algorithm should be able to be implemented by many switches in a large network, and not require a large amount of overhead.

Given the above characteristics, it would also be desirable to have an algorithm that finds the "best" paths between a source and destination and is "fair" to different sessions. However, concepts like "best" and "fair" can be interpreted in several different ways.

Routing Characteristics

A variety of different routing algorithms have been proposed and used in networks. These algorithms can be classified in several ways. A routing algorithm is ***static*** if it uses predetermined routes that do not change over time. A ***dynamic*** routing algorithm bases the route assignment on the current network load, in this case route assignments may change with each session or with each packet. Routing algorithms may be ***centralized***, where routes are calculated at one place and disseminated to all nodes, or ***distributed***, in which case all routers exchange information and calculate routes locally. For example, the spanning tree algorithm used by transparent bridges is a dynamic, distributed algorithm. Most of the algorithms we will consider in the following are also of this type.

Flooding

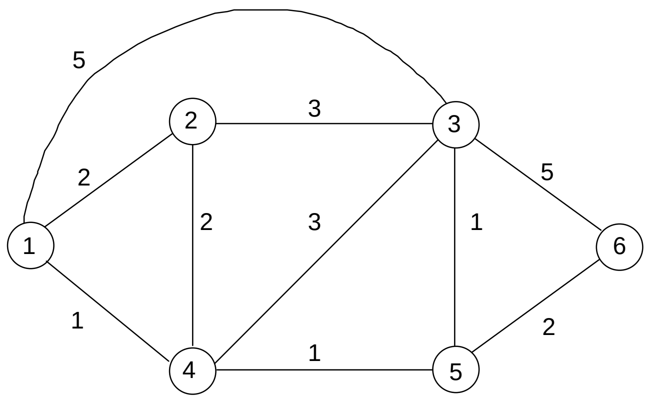
A simple approach to routing is ***flooding***. With flooding, upon receiving a packet each node forwards it over all links except the one from which it is received. Thus, every node on the network will eventually receive the packet (including the desired destination). One problem with flooding is how to make sure that packets do not circulate forever. This can be accomplished by several methods including:

1. Put a sequence number on each packet, and have each switch make sure it does not forward a packet more than once.
2. Put an age field in the packet header that is decreased every time a packet is forwarded, don't forward any packets once this field reaches zero.
3. Put an identifier for the links a packet has traversed in the packet header, don't forward a packet over a link more than once.

An advantage of flooding is that no routing tables or topological knowledge needed at any switch, and little processing is required at each node. The disadvantage is that much unneeded traffic is generated - every packet will be sent over every link. Two places where flooding is used are (1) in some ad hoc wireless networks, where nodes are moving around to quickly for any other techniques to work, and (2) for forwarding control or topological information (e.g. in link state routing discussed later).

Shortest path routing

Many routing algorithms are variants of ***shortest path algorithms.*** In such algorithms, each link is assigned a ***cost*** – this could represent some quantity of interest such as distance or delay for that link. A shortest path algorithm then attempts to find the route between any pair of nodes that minimizes the sum of the link costs. To visualize such algorithms, consider a **graph** corresponding to the subnet as shown below. In this graph, the nodes represent routers and the arcs represent links. The number by each link is the cost of the link. In this figure, the shortest path between nodes 1 and 5 is through node 4 and has a total cost of 1+1 = 2.



To build routing tables we would like to find the shortest path from each node to all other nodes. Also, for dynamic routing, link costs will change with time and so may the shortest paths. A method is needed to efficiently calculate shortest paths. Two of the main approaches for calculating shortest paths are:

1. **Distant vector routing** - uses distributed **Bellman-Ford algorithm**.

2. **Link state Routing** – uses **Dijkstra's algorithm**.

Both of these can be implemented as distributed, dynamic routing algorithms.

Distance Vector Routing

In distance vector routing, each router must maintain a table (vector) of its best-known distance (cost) to each destination as well as which link to use to get there.

Each router is assumed to know the cost of the links that connect it directly to each of its neighbors[1].

Routers then exchange their distance vectors **locally,** i.e. with their neighbors.

They update their distance vectors based on these exchanges.

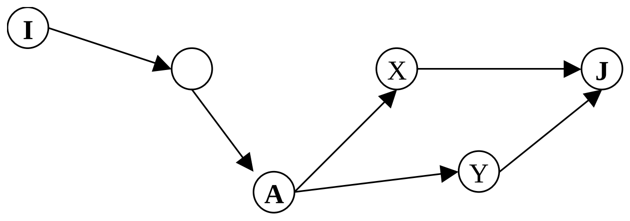
The **Bellman-Ford algorithm** is used for this updating.

[1] In a graph two nodes are called *neighbors* if there is a link that directly connects them.

"Optimality principal"

The basic idea behind the Bellman-Ford algorithm is the so-called "optimality principal". This states that if node **A** is on a shortest path from node **I** to **J**, then the segment of the path from **A** to **J** is also a shortest path from **A** to **J**.

In the figure below, assume that the shortest path from **I** to **J** is through node **A** and then node **X**. Suppose that the optimality principal was not true and that the path **A -Y- J** is a shorter path from **A** to **J** than **A -X- J.** Then by taking the path from **I** to **A** followed by the path **A-Y-J**, must result in a shorter path from **I** to **J** than the path through **X.** This contradicts the assumption that the original path was the shortest.



Bellman-Ford

Next we describe the Bellman-Ford algorithm. We describe the algorithm in the context of calculating the shortest path from all nodes in a network to a given destination node. A version of the algorithm would be implemented for each destination.

For the given destination, let $D_n(j)$ be the estimated distance from node j to the destination at time n .

Let $d_{i,j}$ be the distance from node i to j.

Basic idea:

Each neighbor j of node i sends $D_n(i)$ to node i, then node i updates its estimate according to:

$$D_{n+1}(i) = \min_{j \in N(i)} (d_{i,j} + D_n(j))$$

The quantities on the right-hand side of this expression are the distances to each neighbor, j plus the minimum distance from j to the destination. The port towards the neighbor j that attains the minimum is chosen as the port towards the destination. (Ties can be broken using any rule.)

The algorithm is initialized in the following manner:

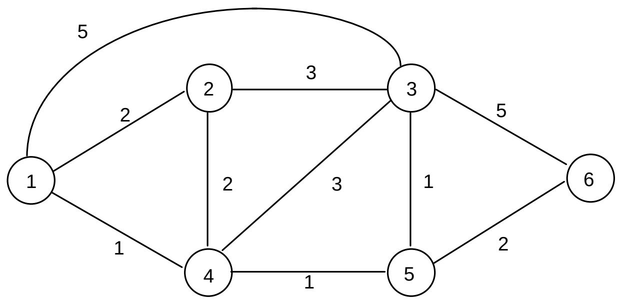
Set $D_0(i) = \infty$ for all nodes i, except the destination, d.

Set $D_0(d) = 0$.

From this initialization, it can be shown that above algorithm will terminate (no more changes will occur) in at most N steps where N is the number of nodes. (Provided there are no link failures, see count-to-infinity problem below) At termination each node knows the shortest path to the destination.

The Bellman Ford algorithm can be shown to work *asynchronously,* where each node sends out its distance vector at arbitrary times.

Bellman - Ford Example



An example of the Bellman-Ford algorithm for the above network is given. Assume the destination, $d = 6$, so $D_0(6)$=0 and D_0(i)= ∞, for all $i \neq 6$.

We consider a synchronous version of the algorithm, where all nodes send out there updates to all their neighbors at the same time.

After the first cycle of updates, all nodes receive costs D_0(i) from their neighbors. The only ones that are going to have non-infinite cost are the nodes 1 link away from d=6. These are nodes 3 and 5, they will receive D_0(6) =0. Thus at node 5 we have:

$$D_1(5) = \min(2+0, 1+\infty, 1+\infty) = 2$$

and the next node on the shortest path for node 5 becomes 6.

Similarly, D_1(3) will be updated to 5.

After the second cycle of updates, node 3 will receive non-infinite costs from nodes 5 and 6.

Thus we will have:

$$D_2(3) = \min(5+0, 2+1, 3+\infty, 3+\infty) = 3$$

The next node on its shortest path will be 5.

Bellman-Ford example

The complete progress of the algorithm is shown in the table below. The notation used here is

(next node on shortest path, distance to destination)

Cycle	1	2	3	4	5
Initial	(-,∞)	(-,∞)	(-,∞)	(-,∞)	(-,∞)
1	(-,∞)	(-,∞)	(6,5)	(-,∞)	(6,2)
2	(3,10)	(3,8)	(5,3)	(5,3)	(6,2)
3	(4,4)	(4,5)	(5,3)	(5,3)	(6,2)
4	(4,4)	(4,5)	(5,3)	(5,3)	(6,2)

Thus the shortest path to node 6 from node 1 is through node 4 and has distance 4.

Note when the costs don't change after a cycle of the algorithm, the algorithm has terminated and found the shortest paths.

Comments on Distance Vector routing:

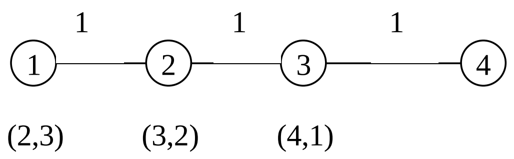
Distant vector routing was original routing protocol for ARPANET. It is the basis for the RIP (Routing Information Protocol) routing protocol used in the Internet. Variations of distance vector routing are used in some Appletalk and Cisco routers as well as in Novell's IPX, and DECnet.

Two shortcomings of distance vector routing are:

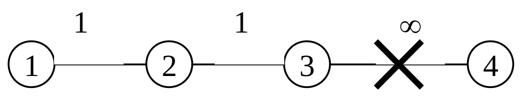
- The convergence rate decreases as the size of network grows.
- The "Count-to-infinity" problem described next.

“Count-to-infinity” problem

The Bellman-Ford algorithm can react very slowly to bad news such as a link failure. For example consider the network shown below, where each link has a cost of 1. Consider the shortest paths from each node to node 4 after running the Bellman-Ford Algorithm. These are listed below the nodes in the figure, using the same notation as above.



Now suppose that the link between nodes 3 and 4 fails, in this case the distance between 3 and 4 becomes ∞. The progress of the algorithm after this failure is shown on the next page. The algorithm may take many iterations before it realizes the link has failed, this is called the ***count-to-infinity problem***.



Before Failure	(2,3)	(3,2)	(4,1)
After Failure	(2,3)	(3,2)	(2,3)
	(2,3)	(3,4)	(2,3)
	(2,5)	(3,4)	(2,5)
	(2,5)	(3,6)	(2,5)

Several heuristic approaches have been proposed to help with this problem, one is called ***split horizon***. With this variation, nodes do not forward their distance vector to the next node along their shortest path. For the above example this speeds up the convergence, but other examples can be found where slow convergence is still an issue.