**CAPITULO 5**

**5.2 Routing Algorithms**

The main function of the network layer is routing packets from the source machine to the destination machine. In most networks, packets will require multiple hops to make the journey. The algorithms that choose the routes and the data structures that they use are a major area of network layer design.

The **routing algorithm** is that part of the network layer software responsible for deciding which output line an incoming packet should be transmitted on. If the network uses datagrams internally, this decision must be made anew for every arriving data packet since the best route may have changed since last time. If the network uses virtual circuits internally, routing decisions are made only when a new virtual circuit is being set up. Thereafter, data packets just follow the already established route. The latter case is sometimes called **session routing** because a route remains in force for an entire session.

It is sometimes useful to make a distinction between routing, which is making the decision which routes to use, and forwarding, which is what happens when a packet arrives. One can think of a router as having two processes inside it. One of them handles each packet as it arrives, looking up the outgoing line to use for it in the routing tables. This process is **forwarding**. The other process is responsible for filling in and updating the routing tables. That is where the routing algorithm comes into play.

**5.2.2 Shortest Path Algorithm**

The idea is to build a graph of the network, with each node of the graph representing a router and each edge of the graph representing a communication line, or link. To choose a route between a given pair of routers, the algorithm just finds the shortest path between them on the graph.

However, many other metrics besides hops and physical distance are also possible. For example, each edge could be labeled with the mean delay of a standard test packet, as measured by hourly runs. With this graph labeling, the shortest path is the fastest path rather than the path with the fewest edges or kilometers.

By changing the weighting function, the algorithm would then compute the ‘‘shortest’’ path measured according to any one of a number of criteria or to a combination of criteria.

\*\*inserir algoritmo de Dijkstra aqui

**5.2.3 Flooding**

When a routing algorithm is implemented, each router must make decisions based on local knowledge, not the complete picture of the network. A simple local technique is **flooding**, in which every incoming packet is sent out on every outgoing line except the one it arrived on.

Flooding obviously generates vast numbers of duplicate packets, in fact, an infinite number unless some measures are taken to damp the process. One such measure is to have a hop counter contained in the header of each packet that is decremented at each hop, with the packet being discarded when the counter reaches zero.

Ideally, the hop counter should be initialized to the length of the path from source to destination. If the sender does not know how long the path is, it can initialize the counter to the worst case, namely, the full diameter of the network.

Flooding is not practical for sending most packets, but it does have some important uses. First, it ensures that a packet is delivered to every node in the network. This may be wasteful if there is a single destination that needs the packet, but it is effective for broadcasting information.

Second, flooding is tremendously robust. Even if large numbers of routers are literally blown to bits, flooding will find a path if one exists, to get a packet to its destination. Flooding also requires little in the way of setup. The routers only need to know their neighbors.

**5.2.4 Distance Vector Routing**

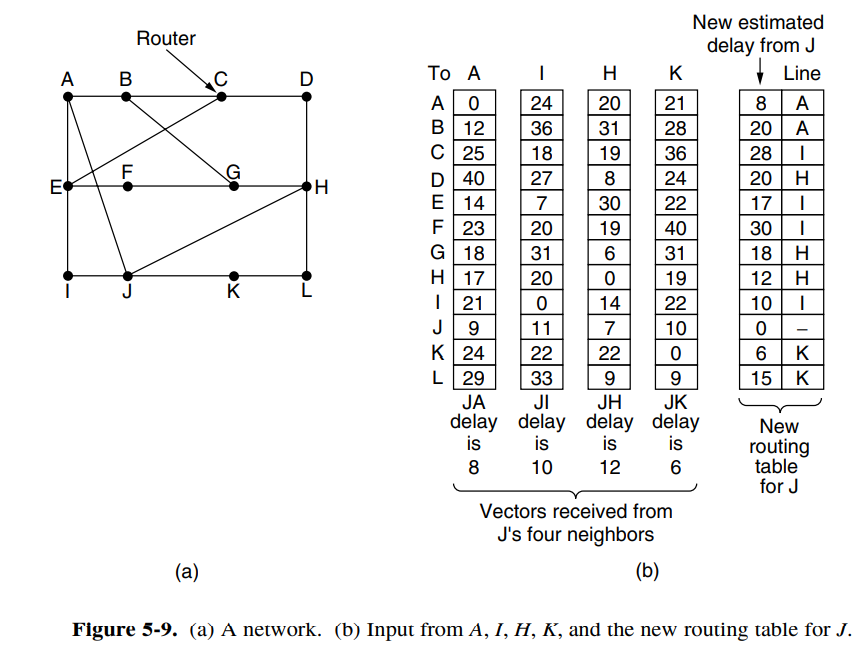
A **distance vector routing** algorithm operates by having each router maintain a table (i.e., a vector) giving the best-known distance to each destination and which link to use to get there. These tables are updated by exchanging information with the neighbours. Eventually, every router knows the best link to reach each destination.

The distance vector routing algorithm is sometimes called by other names, most commonly the distributed **Bellman-Ford** routing algorithm.

In distance vector routing, each router maintains a routing table indexed by, and containing one entry for each router in the network.

This entry has two parts: the preferred outgoing line to use for that destination and an estimate of the distance to that destination. The distance might be measured as the number of hops or using another metric, as we discussed for computing shortest paths.

As an example, assume that delay is used as a metric and that the router knows the delay to each of its neighbours. Once every T msec, each router sends to each neighbour a list of its estimated delays to each destination. It also receives a similar list from each neighbour.



Imagine that one of these tables has just come in from neighbour X, with Xi being X’s estimate of how long it takes to get to router i. If the router knows that the delay to X is m msec, it also knows that it can reach router i via X in Xi + m msec.

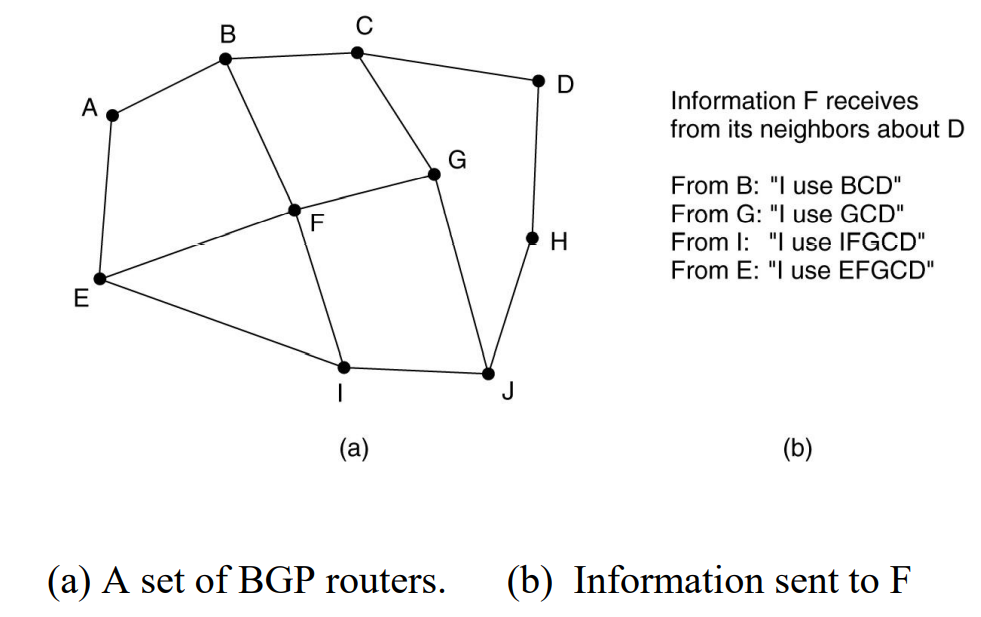
By performing this calculation for each neighbour, a router can find out which estimate seems the best and use that estimate and the corresponding link in its new routing table.

* **Routing Information Protocol (RIP)**

Distance Vector Protocol: Nodes send distance vectors every 30 seconds or when an update causes a change in routing. RIP is limited to small networks.

* **BGP – The Exterior Gateway Routing Protocol**

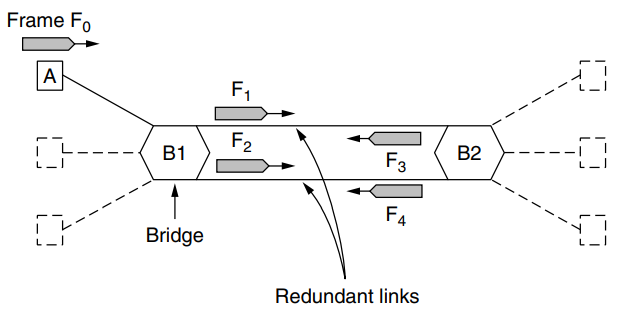
Each router sends to its neighbours their chosen path to reach a given Destination Router.



**CAPITULO 4**

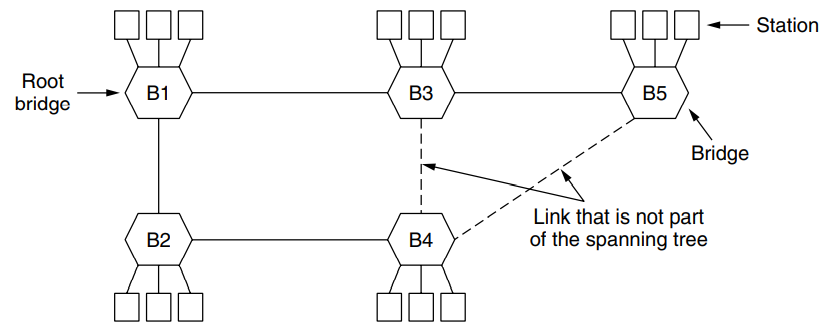
**4.8 Data Link Layer Switching**

**4.8.3 Spanning Tree bridges**

To increase reliability, redundant links can be used between bridges. This design ensures that if one link is cut, the network will not be partitioned into two sets of computers that cannot talk to each other.

However, this redundancy introduces some additional problems because it creates loops in the topology.

The solution to this difficulty is for the bridges to communicate with each other and overlay the actual topology with a spanning tree that reaches every bridge. In effect, some potential connections between bridges are ignored in the interest of constructing a fictitious loop-free topology that is a subset of the actual topology.



The pictured topology can be thought of as a graph in which the bridges are the nodes and the point-to-point links are the edges. The graph can be reduced to a spanning tree, which has no cycles by definition, by dropping the links shown as dashed lines.

Using this spanning tree, there is exactly one path from every station to every other station. Once the bridges have agreed on the spanning tree, all forwarding between stations follows the spanning tree. Since there is a unique path from each source to each destination, loops are impossible.

To build the spanning tree, the bridges run a distributed algorithm. Each bridge periodically broadcasts a configuration message out all of its ports to its neighbours and processes the messages it receives from other bridges, as described next. These messages are not forwarded, since their purpose is to build the tree.

The bridges must first choose one bridge to be the root of the spanning tree. To make this choice, they each include an identifier based on their MAC address in the configuration message, as well as the identifier of the bridge they believe to be the root.

MAC addresses are installed by the manufacturer and guaranteed to be unique worldwide, which makes these identifiers convenient and unique.

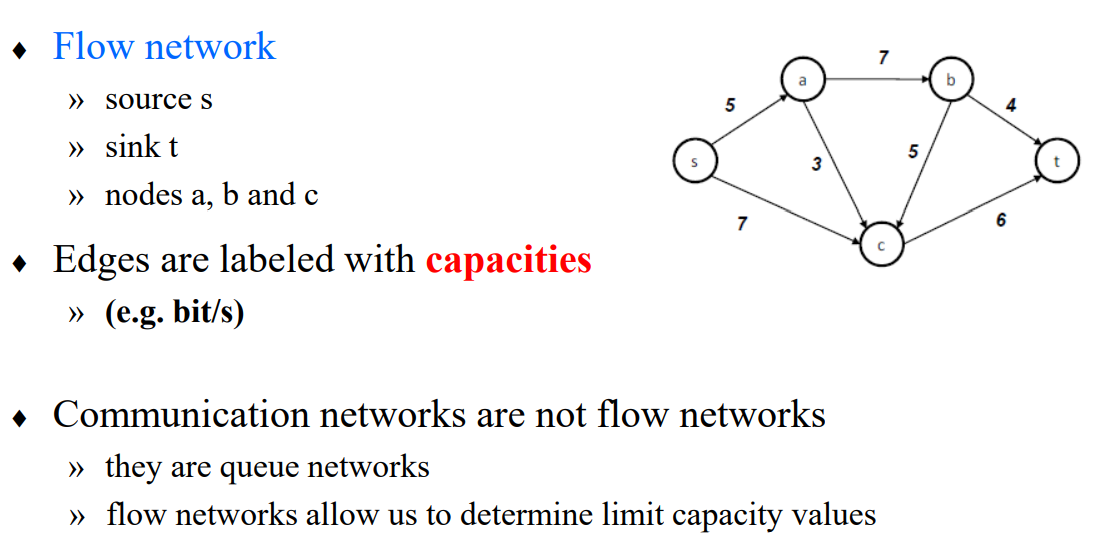
The bridges choose the bridge with the lowest identifier to be the root. After enough messages have been exchanged to spread the news, all bridges will agree on which bridge is the root.

Next, a tree of shortest paths from the root to every bridge is constructed. To find these shortest paths, bridges include the distance from the root in their configuration messages. Each bridge remembers the shortest path it finds to the root. The bridges then turn off ports that are not part of the shortest path.

Although the tree spans all the bridges, not all the links (or even bridges) are necessarily present in the tree. This happens because turning off the ports prunes some links from the network to prevent loops.

Even after the spanning tree has been established, the algorithm continues to run during normal operation to automatically detect topology changes and update the tree.

**SLIDES POWERPOINT**

**Maximum flow of a Network**

\*\*inserir aqui max-flow min-cut

* **Sidebar (algo que nao é, de todo, importante para o exame)**

Radia Perlman’s was to solve the problem of joining LANs without loops. She was given a week to do it, but she came up with the idea for the spanning tree algorithm in a day. Fortunately, this left her enough time to write it as a poem (Perlman, 1985):

*I think that I shall never see*

*A graph more lovely than a tree.*

*A tree whose crucial property*

*Is loop-free connectivity.*

*A tree which must be sure to span.*

*So packets can reach every LAN.*

*First the Root must be selected*

*By ID it is elected.*

*Least cost paths from Root are traced*

*In the tree these paths are placed.*

*A mesh is made by folks like me*

*Then bridges find a spanning tree.*