Level 2 Routing: LAN Bridges and Switches

Norman Matloff
University of California at Davis

September 6, 2001

1 Overview

In a large LAN with consistently heavy traffic, it may make sense to break up the LAN into **segments**, i.e. smaller LANs, and connect them in some manner, with the goal of improving performance. For instance, breaking one Ethernet into two smaller ones means that two simultaneous transmissions are now sometimes possible. Or, if we break one large token ring into two smaller ones, the time between visits of the token is reduced.

The easier way to do this is via a **bridge**. We will describe how simple bridges work first, and then discuss an advanced form of bridges called **switches**, which today are in common use.

The segmented LAN which results is called an **extended LAN**. Nodes on the LANs then in essence see the extended LAN as one large LAN, with the bridges or switches hiding the fact that there are actually many individual LANs; in other words, the nodes do not "know" that the LAN is segmented.

Bridges and switches perform functions somewhat similar to **routers**, but they operate at the data-link level (Level 2 in the seven-layer model), in contrast to the network level (Level 3) used by routers. A bridge or switch will be more efficient; a message will go through fewer layers (in the seven-layer model sense) when routed by a bridge or switch than it would with a router, and other efficiencies (especially in the case of a switch) may be attained.

On the other hand, all the nodes on a segmented LAN must be under the control of the same administrator, say all within the same company. When crossing administrative boundaries, we are forced to use routers. Also, routers can connect two very different kinds of LANs, say an Ethernet and an FDDI network, whereas bridges need the two LANs to at least have the same address structure (e.g. the 48-bit addresses of Ethernet and FDDI), and switches typically need the two LANs to be the same kind of networks.

(For simplicity, we will assume Ethernets in the remainder of this unit.)

2 Bridges

In its simplest form, a bridge can consist of just a PC and two Ethernet cards, with some software monitoring both cards. In more complex forms, the bridge would be implemented fully, or almost fully, in hardware, for speed, and may be connected to several LANs, not just two. However, you will usually find that it will easier

to understand the concepts if you use the PC case with two Ethernet cards as your model while reading the material below.

Suppose LANs A and B are connected by a bridge. Each of the Ethernet cards of the bridge is called a **port**; if there are k of them, we will assign them ID numbers 0 through k-1, in this case here 0 and 1, for LANs A and B, respectively.

Whenever a bridge sees a frame on port 0 whose destination address[1] it knows to belong to a node on LAN B, the bridge will simply copy the frame to port 1. On the other hand, if the bridge knows that the destination is also on LAN A, the bridge will ignore the frame. If the bridge does not know which LAN the destination address is on, the bridge will copy to LAN B anyway, just in case.

Of course, if the source node is on LAN B, a similar sequence of events will occur to those described above.

Keep in mind the central role here of the broadcast nature of the Ethernet. The source node in the preceding paragraph simply put the frame on LAN A, which resulted in every Ethernet card on that LAN seeing the frame. This includes the Ethernet card at the bridge. If the destination node happens to be on LAN B, when the bridge copies the frame to that LAN, every node on that LAN will see the frame, including the destination node, so that the transmission will be complete.

Restating this in slightly more general terms, each bridge will maintain a table, with each entry consisting of two items, a destination node address and a port number. If the bridge sees a frame come by on port x, the bridge will check the table. If the frame's destination has an entry in the table, the bridge will copy the frame to the indicated port (unless the port is x, so that no action is needed). Note that the destination node may be several **hops** away, in which case the indicated port in the table will be the site of the first of the hops. If the destination node is not in the table, then the bridge simply copies the frame to all of its ports (again except for x), i.e. **flooding** is used.

When the network first is brought up, the tables are empty. They are then expanded (though typically never completed) gradually via **address learning**: In the scenario described in the last paragraph, let s be the source node address of the frame which was just observed on port x. If there is no entry for s in the table, the bridge will add one, specifically the pair (s,x), even though s is not the destination for the current frame. This is for future reference, with the entry in the table meaning "If I ever do see a frame addressed to s in the future, I know now that I must copy the frame to port x." (Note that this means tht either x is on the LAN at port x, or is on some other LAN which through one or more hops connects to the LAN at port x.)

Note another difference between a bridge and a router, at least with the type of bridge we are discussing here: The bridge is transparent to the nodes on the various LANs. When one node sends to another, it places the Ethernet ID of the destination in the frame, not the Ethernet ID of a bridge on the sender's LAN. By contrast, in the case of routers, the sending node would consciously send to a router on its LAN, by placing the Ethernet ID of the router in the frame.

2.1 Spanning Trees

If the segmented LAN consists of many LANs and bridges, then it might include loops, in the graph-theoretic sense. This would cause problems. For example, a frame may circulate endlessly, never reaching its destination.

[1] Again, this will be an Ethernet address, consisting of a 48-bit serial number on the node's Ethernet board, not an Internet IP address.

Typically a **spanning tree** is used to eliminate these loops, as well to set up paths which are in some sense of minimum lengths. The **Spanning Tree Protocol** is an official standard, IEEE 802.1D, developed by Radia Perlman at DEC.

The main idea of the spanning tree is to turn off some ports at some bridges, with the result that the graph of connections among the bridges has the form of a tree, which guarantees that there are no loops. The root of this tree is called the **root bridge**. Note that there is nothing special about this bridge; we merely have to choose one bridge to play this role, and for the purpose of eliminating loops any bridge would work.

2.1.1 Terminology

Each bridge will be assigned an ID number, typically chosen to be the lowest-numbered Ethernet address of all the cards on that bridge. (Some systems allow the administrator to set bridge ID numbers, which is useful for example if one wants the minimal spanning tree to choose the fastest bridge at the root.)

We define the following terms:

- **root bridge**:
  The bridge having the lowest-numbered ID is called the root bridge.
- **root port**:
  For any given bridge, its root port is the port through which the path to the root bridge, in number of hops, is shortest.[2]
- **designated bridge** and **designated port**:
  For any given LAN, the designated bridge is the bridge through which a frame on this LAN can reach the root bridge in the shortest number of hops. The port through which this LAN connects to the designated bridge is called the designated port.

(Though we have defined "cost" here in terms of number of hops to the root, we could also defined it other ways, e.g. in terms of link speed etc.)

After the spanning tree is determined, any port which is not either a designated port or root port of some designated bridge (or of the root) is placed in a nonforwarding state, meaning that it will no longer process message frames.[3]

The extended LAN then enters its address learning process and then operates as described earlier.

Each message frame will travel through the tree as follows: It starts out toward the root. At each hop from one LAN to another, it goes through the designated bridge of the first LAN, entering via that bridge's designated port and exiting via that bridge's root port. It may reach its destination before reaching the root. If it does reach the root, then from that time onward it will follow a reverse pattern at each hop, entering via root ports and exiting via

[2] In the case of a tie, the port with the lowest-numbered ID is chosen.

[3] It may, however, still monitor the network, watching for changes such as bridge failures.

2.1.2 Bridge Protocol Data Units

Certain nodes will send out frames called Bridge Protocol Data Units (BPDUs) periodically, typically every 1 to 4 seconds.[4] The Ethernet cards recognize them through their destination address, which is 0x0180c2000000. When the network first comes up, all nodes will be both originating BPDUs and relaying the BPDUs originated by other nodes.

The format of a BPDU is:

```
sender ID    claimed root ID    distance to root    age
```

2.1.3 Election of the Root

At first, all bridges are sending out BPDUs claiming to be the root (with distance of 0). When a bridge A receives a BPDU sent (originated or relayed) by bridge B, A does the following:

```
if (B's claimed root < A's claimed root or
    B's claimed root = A's claimed root and B's distance < A's distance or
    B's root and distance = A's and B's ID < A's ID) then
        A changes its claimed root record to B's
        A adds 1 to B's distance, to produce A's distance
        A relays B's BPDU (with updated distance) to all of its ports
           except the one B's BPDU came in on
        A stops originating BPDUs
else
    A ignores the BPDU
```

In the first iteration, A and B above both will be claim to be the root with distance 0, but whichever one has the smaller ID will "win." As time passes, fewer and fewer bridges will be originating BPDUs, until only one bridge is doing so. That bridge then becomes the root.

Each time a BPDU is processed by a bridge, the bridge will increment the **age** field by 1. If the age reaches an agreed-upon value, say 20, then the BPDU is discarded, on the grounds that it is no longer up-to-date.

2.1.4 Election of the Designated Bridges

Remember, there will typically be several bridges attached to a given LAN. (But not all nodes on the LAN will be bridges.) Once the root of the extended LAN has been determined, as shown above, then within each given LAN all the bridges on that LAN must elect one of them as the designated bridge for that LAN. To do this, all the bridges on a LAN will now again exchange BPDUs. They will all have the same claimed (and now actual) root, but their distances to the root may vary. If a bridge receives a BPDU with a distance

[4] Note that these are ordinary Ethernet frames, e.g. with no special "privileges."

value less than its own,[5] that bridge will give up. Eventually only one bridge will be left, and it becomes the designated bridge.

The effect of all this is to eliminate loops in the graph consisting of all bridges and LANs. Remember, there is nothing special about the root; we just want to find *some* set of bridges which will make the graph of LANs connected, without loops.

2.1.5 Subsequent Operation of the Network

After the root and all the designated bridges are determined, the root will continue to originate BPDUs, and each designated bridge will continue to relay them.

Each table entry at each bridge has a timer associated with it. It during stable operation the entry times-out before a root BPDU arrives, usually due to some bridge in the network having gone down, the observing bridge will start originating its own BPDUs, so that the network can reconfigure itself.

3 Switches

Switches are an advanced form of bridge.

3.1 Hubs

To explain the idea of switches, we first discuss Ethernet **hubs**.[6] A hub consists of a central repeater, together with ports to which unshielded twisted-pair (UTP) lines are connected.[7]

Each UTP pair will be its own Ethernet, to which network nodes are attached as usual. Furthermore, other hubs may be attached to UTP pairs too. In this manner, a large network consisting of many LANs is connected, with the bridges now being hubs.

The repeater does just that—it repeats Ethernet signals. If one node starts sending, its signal travels along the UTP to the port, where the repeater copies that signal to the UTPs connected to all the other ports. CSMA and CD operations are handled in the usual manner.

A hub is convenient, because its "central star" topology is similar to the wiring using for telephone systems in office buildings; this way the hub can be located in a telephone wiring closet, and the UTPs can be laid in the same passageways as the phone wiring.

3.2 Switching Hubs

As the price of microprocessors and special-purpose chips dropped, there began to be **switching hubs**, now more commonly called simply **switches**. Here the idea is that the hub knows (in the "learning" manner a bridge uses) which UTP/port each node is on, or in the case of many interconnected LANs, which UTP/port will lead to the given node. When the center of the hub sees a frame from a source s on a given port, with

[5] Or of equal distance but smaller ID.

[6] The reader should be cautioned that the term *hub* connotes an entire class of devices, with considerable variety within that class.

[7] The twisting in a UTP is braidlike. The idea is that electrical noise will largely "cancel out" on the two wires.

destination address d, the center will then copy that frame only to the proper UTP/port, instead of to all ports. In this manner, more than one frame can be transmitted at a time, as long as the frames' destinations are on different ports. Again, one uses ordinary Ethernet cards in the nodes.[8]

Each UTP/port might be its own mini-Ethernet, with more than one node on that UTP. Thus within a UTP, there may still be collisions in the usual sense,[9] which are handled in the usual way. However, as noted above, there will be no collisions between frames headed for different ports; we say that each port is a separate **collision domain**. Moreover, even if two frames have the same destination port, a typical switch will buffer the one while delivering the other, sending the second frame later, thus avoiding a collision and wasted backoff time.

Of course, something must be done if the switch runs out of buffer space. Some designs handle this by placing a fake collision on the UTP which generated a buffer overflow. The source of the frame will then wait a random backoff time and then try again. The switch will generate a fake message to a port when the buffer space for that port is near depleted, which will prevent nodes on that port from trying to send; as soon as buffer space for the port becomes available, the fake message is terminated. These approaches help in some ways, but also have their costs, since the fake collision or message will prevent other nodes on the same UTP from sending to each other, when actually it would be perfectly safe to do so.

Some ports on a switch may be configurable as **full duplex**. Here a pair of ports are dedicated to communication with each other; no other ports can send to or receive from these ports. The two full duplex ports can also send to each other simultaneously. The reason for this is as follows. In an ordinary port, there are actually two UTPs, one for send and one for receive. A node which is currently sending on one UTP is listening for collisions on the other. But since there will be no collisions at all for a pair of ports which are dedicated to communication with each other, this frees up the receive UTP while a node is sending. Thus that node can be receiving from its partner on that UTP at the same time.

3.3 Internal Design

A switch is likely to have a microprocessor (or more than one) inside. The designers of the switch write a program which controls the movement of a frame from one port to another, checks for collisions, manages buffers and so on. There is also likely to be a network inside the switch itself, consisting of paths which frames take from one port to another.

For example, an Ω network can be used. Suppose there are n ports in all. Then the network will consist of $log_2 n$ stages. At each stage, there will be a "fork in the road," giving 2 choices. After $log_2 n$ stages, that will generate

$$\underbrace{2 \times 2 \times ... \times 2}_{log_2 n factors} = 2^{log_2 n} = n$$

paths—one to each of the ports, just as we want.

Because of the special hardware, a switch will be much more powerful than a hub. (Actually, we can think of a switch as an "advanced hub.") For example, it can route multiple frames (to different destination LANs)

[8] This was a big attraction to consumers, which is one of the reasons Ethernet has managed to remain the dominant LAN technology in spite of its age of 20 years or so.

[9] Thus, for best performance, one should avoid having more than one node on a UTP, if it is economically feasible.

simultaneously. Also, as we saw above, it can buffer frames, thus avoiding collisions. We pay a price in terms of increased complexity—for instance, the switch must constantly monitor all ports for idleness, and when one becomes idle, the switch must check its buffers for frames which need to go to that port—but the increase in performance is worth it.