Csc72010

Switches and Bridges

Initial Algorithms

OSI Layers

Seven layers
The data link layer

Switches and Bridges

Interfaces

Forwarding

My intent with these algorithms was to illustrate simple Ethernet LAN algorithms that bridges and switches use. Links in these networks are bidirectional. When I discussed these algorithms today, I was thinking bidirectional edges, as I think most of you were (otherwise, there should have been a few more questions).

There are networks that have unidirectional links (SONET, wireless), but the properties of these algorithms are much more complicated for unidirectional links, and they're unlikely to be used in that environment. So, using in-nbrs and out-nbrs is misleading, and I have changed the model to collapse in-nbrs and out-nbrs into nbrs. I also assume that the network graph is an undirected graph for this particular purpose. I encourage you to think about what happens in a directed network graph, but make sure you understand the bidirectional case first.

Forwarding algorithm idea
Each message received on an interface of a bridge is sent out every other interface.

Forwarding algorithm messages
I distinguish between the initial end-to-end messages sent by hosts and the collections of end-to-end messages sent by switches.

Let D be the set of all possible end-to-end messages. Members of D are uninterpreted symbols, and a switch can't do anything with them but copy them and collect them in vectors.
Let $M=\otimes D$ be the set of all finite vectors having components in D[1]

We want to consider both hosts and switches as nodes of the network graph, without differentiating between them. To make this work, the type of message sent on a host-to-

[1] In class, I said sets. However, we really need to allow the same member of D to appear in M more than once, because we can't tell whether two messages that we receive at different times are the same message or different ones.

switch link is the same as the type sent on a switch-to-host link. This means a host must encapsulate a message d in a vector <d> when sending it. Also, the switch sends vectors of messages to each connected host. We assume that the host can figure out which messages are addressed to it.

Forwarding Algorithm

```
states
    inbuf: Array[Int, M ∪ {null}], initially constant(null) [2]
    outbuf: Array[Int, M], initially constant(< >)

msgs
    for each i ∈ nbrs
        if inbuf[i] != null then
            outbuf[j] = outbuf[j] + inbuf[i] for all j ∈ nbrs with j != i
    for each i ∈ nbrs
        if outbuf[i] != < > then
            send outbuf[i]

trans
    for each i ∈ nbrs
        inbuf[i] = message on link i
        outbuf[i] = < >
```

Correctness

Correctness conditions that we would like to have are:

Each message eventually arrives at its destination.
No message continues being forwarded forever.

The first correctness condition will hold for this algorithm if the network graph is connected (i.e., there is an undirected path from each node to every other node) and there are no changes to the graph during the execution. The intuition for the proof is that after round r, the message will have reached all nodes whose distance from the source is r. If the distance(src,dst) is n, then after n rounds, the message will have reached the destination.

An invariant that you can use to prove this is:

If source host i sends message m at the beginning of round 1, then for every node j of the graph such that distance(i,j) = r, inbuf[j] contains m at the end of round r.

The claim is that the invariant holds for all r; you can prove it by induction on r. Where do you use the connectedness of the graph?

The second correctness condition will hold only for graphs without cycles, that is, trees. An invariant that you could use to prove the second condition is:

[2] I distinguish between null (no message was sent) and < > (a message containing no end-to-end messages was sent)

At any round r, if node i is on the path from the source of the message to node j, and i != j, and if m has appeared in inbuf[j], then m is not in inbuf[i] in round r.

In other words, the message never goes backwards. Once you show this, then you can show that once m has appeared in inbuf[j] for a leaf j of the tree, it cannot be in inbuf[i] for any interior node on the path from the source.
Again, use induction on r to prove the invariant.

Complexity:
Communication: In a connected graph, each message traverses every edge of the network graph at least once. If the graph is a tree, the message traverses each edge exactly once. So communication complexity is |E| in a tree and infinite if there are cycles in the graph.
Time: The time complexity for a message to go from src to dst is distance(src, dst) rounds from the time the originating host sends the message until it arrives at the destination host. Worst case time complexity for a message to go from src to dst is diam(G).

Learning bridges

Learning bridge idea
The bridge associates the source address on an incoming message with the interface; sends messages destined for that source out that interface only. Note that this is not much use in an arbitrary directed graph. It requires bidirectional links.

Learning bridge messages
Let D be the set of all possible end-to-end messages, as above.
Let A be the set of addresses
Let M = ⊗(tuple of src:A,dst:A,msg:D) be finite vectors over tuples src:A, dst:A, msg:D
Let this designate the index of the node running the algorithm

Learning bridge algorithm

```
states
    inbuf: Array[Int, M ∪ {null}], initially constant(null)
    outbuf: Array[Int, M], initially constant(< >)
    mac-address-table:Array[A, Int ∪ {null}], initially constant(null)
    this:Int
msgs
    for each i ∈ nbrs
        for each m ∈ inbuf[i]
            if m.dst != this then
                if mac-address-table[m.dst] != null then
                    add inbuf[i].msg to outbuf[mac-address-table[m.dst]]
                else
                    add inbuf[i].msg to outbuf[j] for all j ≠ I
                end if
            end if
    for each i ∈ nbrs
            send outbuf[i]
trans
    for each i ∈ nbrs
```

```
inbuf[i] = message on link from i
for each m $\in$ inbuf[i]
    if mac-address-table[m.src] = null then
        mac-address-table[m.src] = i
    end if
```

Correctness

For Thursday, February 17, prove:

1. In any connected graph, every message eventually gets to its destination.
2. If the network graph is a tree, no message continues being forwarded forever.