4 Scaling the Data Link Layer

4.1 Goal of this section

The goal of this section is for readers to understand how we scale up the Data Link Layer, what problems we hit, and how we handle them, including a deeper dive on Spanning Tree Protocol. We also touch on the Ethernet Ring Protection Switching Protocol, principally to compare and contrast how different protocols solve the same overall problem while making different trade-offs.

4.2 Scaling your Layer 2 Network – Moving from Hubs to Switches

4.2.1 The problem with scaling Ethernet

As described in the previous section, when using Ethernet any frame being sent from one endpoint is broadcast to every other endpoint (and the endpoints that don't match the destination then drop the frame). Scaling up, adding more endpoints to the Ethernet network creates an n^2 problem – as there are n endpoints all attempting to broadcast frames to n other endpoints. Remember, if any two endpoints attempt to transmit at the same time, that causes a collision and both have to stop and try again. As we scale up, we rapidly hit the point where the Ethernet collisions occur continually, and the entire network clogs.

Switches

To mitigate these scaling problems, we enhance our network by adding intelligence to the hubs. A hub explicitly refers to an unintelligent layer 2 repeater; a layer 2 hub that has some form of decision-making over what is sent where is a **switch**. These days, basic switching hardware is so cheap that any piece of layer 2 kit that you buy will be a switch, not a hub.

Switches do two key things – collision domain splitting and MAC learning.

Collision domain splitting

Hubs blindly repeat frames as they come through the Hub, and so a collision anywhere on the network must be handled by the endpoints. Switches can store and handle retransmission of frames, meaning that once a frame has successfully reached a switch, the switch will handle retransmission. This splits the Ethernet network into sub-areas (called collision domains) and a collision in one domain does not affect traffic/retransmission in other domains. Note that switches have limited buffer space, and if they receive more traffic than they can process through, they simply drop the additional frames.

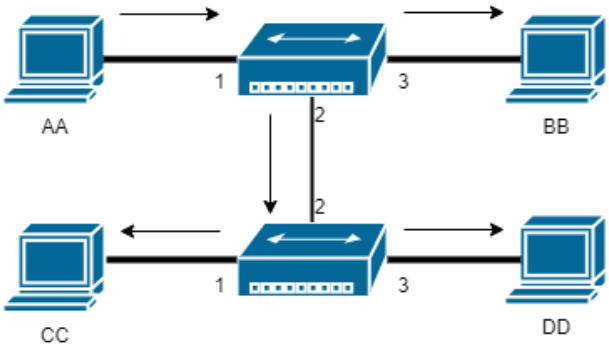
In extremis, with every node in the network being a switch, this eliminates collisions. However, note that while this solves the collision problem and reduces retransmissions, it does not solve the n^2 frame delivery problem.

4.3 Source MAC learning.

Looking back at the Ethernet header in the previous section, you'll notice that in addition to the MAC address of the destination, the header also includes the MAC address of the source. Switches can use this information to learn where in a network each MAC address is situated, called source MAC learning, and use this to ensure delivery of a frame without having to broadcast it. When performing source MAC learning, a switch maintains a lookup-table that matches from MAC addresses to an associated physical port. When a frame arrives, it first associates the physical port the frame arrived on with the source MAC address. It then looks up the destination MAC in the table. If the destination is in the table, it *only* sends the frame out to that port; if not, it broadcasts out to all ports. Over time, the switches build up a full picture of the network, and only send frames where they need to go.

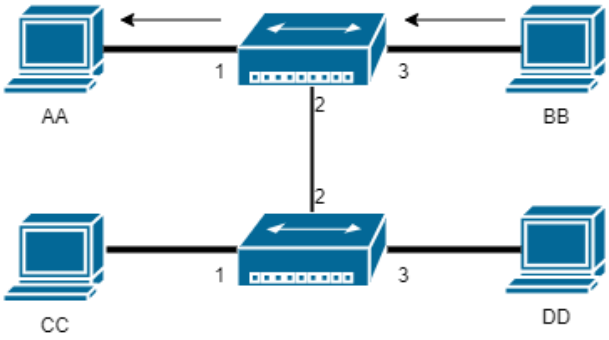
Example

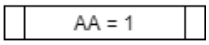
The diagrams below show 4 frames being broadcast through a simple network with two switches.

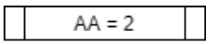


1. Frame AA to CC.

It is broadcast everywhere (but both switches add AA to their MAC tables.

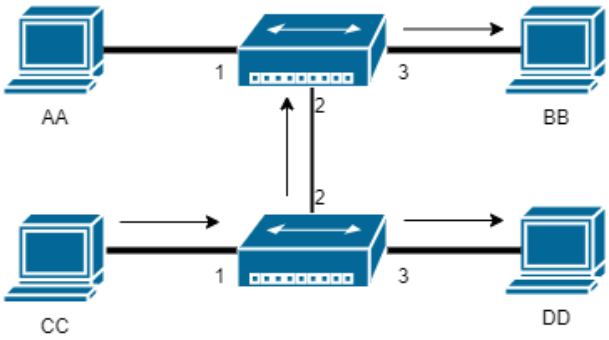


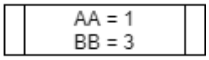


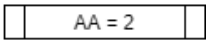


2. Frame BB to AA.

The upper switch sends it directly to AA, and adds BB to its MAC table. The lower switch never sees this frame.

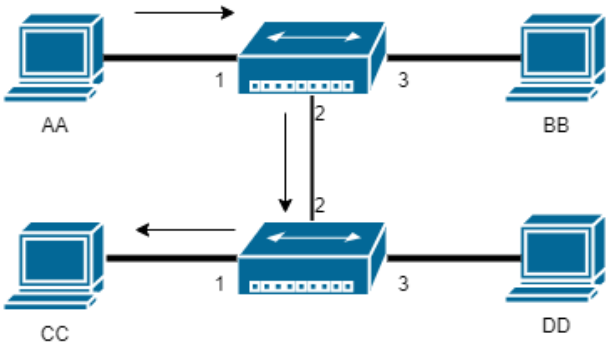


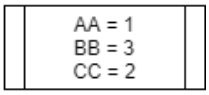


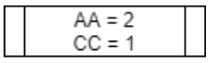


3. Frame CC to BB.

It is broadcast by the lower switch (BB is not in its MAC table) , but the upper switch switches the frame directly to BB. Both switches add CC to their MAC tables.







4. Frame AA to CC.

This is switched directly by both switches.

What needs to happen before a frame can be switched directly to DD?

4.4 What can go wrong?

In a fixed unchanging network, source MAC learning works just fine. However, network changes invalidate the MAC tables and lead to the switches sending traffic to the wrong place such that it will never emerge and reach its destination (often referred to as **black holing**). Consider what would happen in the example above if CC was moved to connect directly to the upper switch.

There are two parts to resolving this.

- Firstly, MAC tables time out (typical timeouts are a handful of minutes), which means that if a switch has not seen a frame from a particular source for a while, then it falls back to broadcasting. As long as each endpoint is sending occasional frames, this rarely happens.
- Secondly, when an endpoint is plugged into a network, it can broadcast send some **gratuitous** frames round the network. These are pointless frames for data transfer and are *purely for the purpose of advertising its MAC address.* Some endpoints can send regular gratuitous frames to prevent MAC tables timing out.

4.5 Scaling your layer 2 network – Adding Redundancy

So far, all of our layer 2 networking has considered running on networks that are graphs that are trees (they have no loops in them). This means that there is a single path between any two points, and any failure along that path results in those two points having no connection between them. We could add redundancy to our network graph by adding in additional loops. However, given our switches broadcast frames onwards, adding loops means we get frames forever looping around the network.

Switching loops are bad. In practice, accidentally adding a switching loop into an Ethernet network with moderate load renders the network flooded full of looping frames and inoperable in less than a second. It also floods the physical network, so any remote login to your switches over your network is now likely inoperable.

4.6 Spanning Tree Protocol (IEEE 802.1D)

A spanning tree is a tree (a network with no loops in) that spans the network (connects every node in the network graph). Spanning Tree Protocol (STP), and its faster relative Rapid Spanning Tree Protocol (RSTP), are protocols designed to allow switches that are linked in a network graph with loops in it to identify and close off the loops, so that Ethernet forwarding only happens along a spanning tree. The switches also monitor the active spanning tree for link failures, and if one occurs, they unblock other links to create a new spanning tree. (R)STP relies on the same Ethernet links that it is controlling in order for the switches to communicate, so it is very important that (R)STP does not allow switching loops. We'll look at STP in more detail below. RSTP works in a similar way, but is designed to update the network to a new stable state much faster after a failure.

4.7 Turning a mesh graph into a tree, and recalculating the tree

In abstract, STP turns the mesh graph into a tree using the following algorithm.

1. The switches choose one of them to become the "root" node in the graph.
2. The switches form a "Shortest Path" Forwarding tree to the root node (where the "cost" of any particular link is inversely proportional to the bandwidth of the link – thus preferring to build the tree over high-bandwidth links).
3. The switches prevent forwarding of traffic into or out of any other links. Note that these other links are not closed to traffic (the switches can still send and receive frames on these links). Switches will not forward frames between these closed links and other links.

STP detects failures of a link by sending Hello (keep alive) messages along each link. If a switch does not receive one of these over a link for a while, it assumes that the link is down, *prevents forwarding on and off that link* (just in case it isn't really down!), and recalculates the next best SPF tree links.

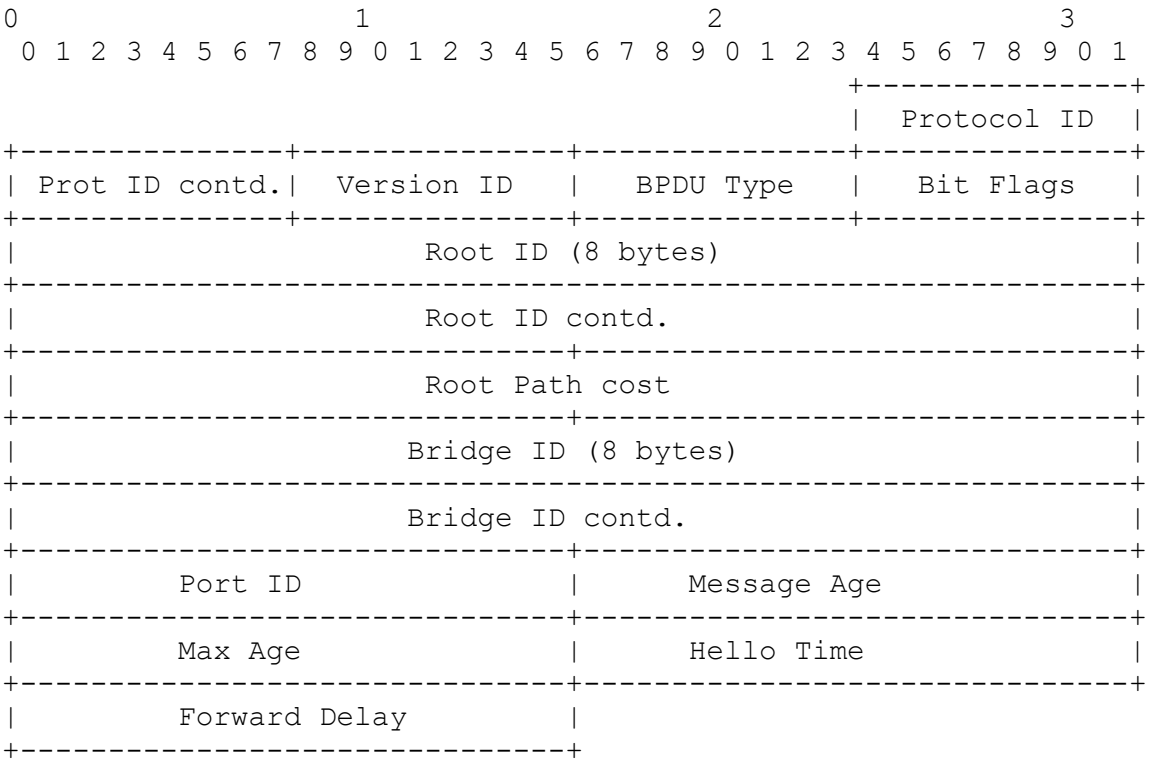
Let's look in more detail at how STP actually does all that.

4.7.1 STP Frame Details

STP uses the source and destination MAC addresses in the Ethernet Frame header to provide addressing, using the special multicast destination MAC address (01:80:C2:00:00:00) to indicate to switches that this is an STP frame that they should look at rather than forward on, and using the source MAC address of the physical port that the frame is sent from.

Note that this implies that all switch ports have MAC addresses (which isn't true of Hubs). Yes, they do – but these are generally kept hidden from endpoints and only used for inter-switch communication.

The payload of STP Ethernet Frame (and other similar protocols) is called a **Bridge Protocol Data Unit** (BPDU). The STP BPDU is below. (Note that the start of the BPDU is offset by 3 bytes in the diagram, simply to make the large fields line up nicely and be easier to read.)



We unpack the contents of this BPDU below.

The first set of fields are all about identifying what the BPDU is, which then allow the rest of the BPDU to be parsed and read.

- The protocol ID (2 bytes) is **0x0000,** indicating that this is a Spanning Tree BPDU.
- The Version ID (1 byte) indicates the actual protocol within the "Spanning Tree" family of protocols. **0x00** is Spanning Tree, 0x01 is Rapid Spanning Tree, and there are others.
- The BPDU Type (1 byte) gives the type of message. Spanning Tree Protocol has two messages – Config (**0x00**) and Traffic Change Notification (**0x80**). These are covered below.

Config message

The fields in the Config message allow STP to run the algorithm above in the following manner.

To enable the switches to determine which switch is the root bridge (switch), each switch has its own unique ID (formed by concatenating a human-configurable "Priority" with the lowest MAC address on the switch to provide a guaranteed unique tie-breaker). The switches all choose the lowest ID in the network to be the "root" switch. To do this, they advertise their own ID and the lowest ID switch that they've heard about. **These are the Root ID, and Bridge ID fields.**

Only the root bridge originates Config BPDUs (and other switches forward them on with updated fields). Of course, when a new switch comes up, it has the lowest Root ID that it knows about and so it originates BPDUs until it receives a BPDU from elsewhere and discovers that it is not 😊!

To enable the switches to build an SPF tree to the root bridge, the switches advertise to their peers the shortest distance from themselves to the root bridge. **This is the Root Path cost field.** When a switch receives a BPDU over a port, it knows the bandwidth on the port, and can thus calculate the distance to the root bridge *going out that port* by incrementing the cost based on the bandwidth of the port.

Obviously, the root port is part of the SPF tree, but which other ports should be included as part of the tree, and which should be blocked? On any *link* the port with the lowest cost path to the root bridge becomes the **designated port** responsible for forwarding traffic on/off that link, and all other ports on that link are blocked. As with switches the lowest MAC address acts as a tie breaker for ports that have the same cost path to the root bridge.

In order for STP to detect network failures, the switches run Keepalives over each link. But how often, and how long should a switch wait for before assuming the link is down and taking action?

- The root bridge sends out a BPDU based on the **Hello Time** (default 2 seconds) – and each switch that receives it, updates and forwards on (as per the above).
- It also sets the **Max Age** (default is 20 seconds) to how long extra a switch should wait before assuming the link to the root bridge is down.
- The **Message Age** of those BPDUs is initiated to 0 at the root bridge, and incremented by 1 by every switch that sends them on – indicting how many hops a switch is from the root bridge. If **Message Age** is ever greater than **Max Age** then something has gone wrong (probably a dreaded switching loop) and the BPDU is dropped.

The packet capture alongside these notes, **04_STP.pcap** contains some STP BPDUs.

Additional note: Many protocols – even simple ones – while simple once you have them conceptually in your head, can have tricky corners to get your head around – and different folks find different bits tricky. For many of them, there are excellent tutorial videos out on youtube (or similar) - typically made by networking hardware manufacturers or consultants and aimed at network administrators. The bit-rate of these is often quite low, but if you're trying to get your head around how a particular part of a particular protocol works, I find that searching those out often helps.

Traffic Change Notification (TCN)

It is useful to warn the whole network when there has been a topology change. That allows switches to take wider action (for example to age out their MAC tables – see near the start of this chapter for why that might be a good idea!)

When it detects a network change, a switch running STP generates a Topology Change Notification (TCN). This is an STP BPDU with the BDPU type set to Topology Change (0x80), and contains no other data. Each successive switch up the path to the root bridge generates a TCN until the root

bridge has received it. The root bridge then sets the Topology Change flag (in the flags field of the Config BPDUs that it sends out), which in turn alerts all the switches in the network of the change.

4.7.2 STP Port States

Preventing switching loops is critical, so while the above algorithm talks about "blocking" and "unblocking" ports, things aren't quite that simple. To unblock a port, a switch running STP puts the port through several stages, from Blocked (no forwarding or anything) to Listening (paying attention to incoming BPDUs on the port) to Learning (reading the source addresses from frames and updating its MAC table), to Forwarding. The time spent in Listening and then Learning is defined by the **Forward Delay** advertised by the root bridge in its BPDUs (default 15 seconds – which is 15 seconds in *each of* Listening and Learning states before progressing to Forwarding).

4.7.3 STP Convergence Times

Convergence times for STP after a change are slow. By default adding up the Max Age (20 seconds) and two lots of Forwarding Delay (15 seconds * 2) to go from Blocked to Forwarding on a new port, means nearly a minute to converge after a failure.

Rapid Spanning Tree gets this down to <10 seconds by lowering the Max Age, and changing how ports go from blocked to forwarding (not covered in this course), but this is still "a handful of seconds". Such is the price of having a simple protocol that switches can implement easily, that works for arbitrary meshed networks, and fanatically avoids the failure mode of switching loops. For a local computer lab's network, "Oh the network's down. Oh, now it's up again 5-10 secs later" isn't usually an unreasonable problem.

4.8 Ethernet Ring Protection Switching Protocol (ITU-T G.8032)

The ITU has a series of standards that enable running phone services over Ethernet, and these networks look very different.

- The networks are huge rings around large cities. Rings, because there is always a redundant path between any two points on the ring.
- The down-time allowed (the time by which the network must have reconverged to a new working network after a failure) needs to be small enough that phone users don't notice any glitching if there's a failure. The spec calls for <50ms, including the time taken to detect the issue and including travel time for any messages on 30km of fibre.

Ethernet Ring Protection Switching protocol solves this. It is much simpler than (R)STP, and works only for rings (and subrings).

We won't go into the guts of the protocol in this course, but it can behave much more simply, as it doesn't have to calculate and cope with determining a SPF – because simply blocking any link turns the ring into a (very simple) tree. This lets ERPS run a very simple algorithm.

1. Choose (or configure) a switch to be the Ring Protection Link owner. It's neighbour on the link is the Ring Protection Link neighbour. Both block that link.
2. When a link somewhere else fails, the switch each side of the failed link blocks that link, and send a message around the link to the RPL owner/RPL neighbour to unblock the Ring Protection Link.
3. When the failed link comes back up, the switch each side of the link detect that the link is back up, and inform the RPL owner/RPL neighbour. The RPL owner/neighbour block the ring protection link again and then OK those switches to unblock the failed link.