So just a brief announcement, on this Friday you guys have a writing tutorial.

It's at 2 p.m. there's only one tutorial this week at 2 p.m., and it's going to be in this room.

So make sure you come.

We're going to talk about issues involved with preparing and presenting your DP2. And it's really important that you come to pay attention.

So today we're going to continue our discussion of networking and network layering.

If you remember last time, we talked about the three layers that are in any typical network stack.

And these three layers we said were the end-to-end layer, the network layer, and the link layer.

And we went to the example of how these three layers interact with each other as, say, a message is sent through a network.

So, on a typical sender node, we said there are these three layers -- And there might also be a receiver node.

And then there could be several.

Each time a message is sent through the network, it might pass through any number of intermediate gateway nodes, or intermediate switches.

So when a packet gets sent in, it gets sent through the end-to-end layer in through the network layer, down into the link layer.

The link layer chooses the next link to send the packet out over, since it's one of these switches.

The switch looks at the packet, sends it up to its own network layer, which is in charge of determining the next link that the message will take.

On the next hop, the message goes up to the link layer.

The link layer determines yet another link for the message to take.

The network layer determines yet another message, link, for the message to take.

And then finally, the message reaches the receiver or the message propagates up through the link layer into the

network layer, and then finally to the end-to-end layer, and out to the user.

So we talked a little bit last time about various things that happen in this architecture.

We said that there's this process of encapsulation that happens at each step along the way.

So the end-to-end layer may attach headers on to the packet, a header or trailer onto the packet; the network layer may attach a header and trailer, and the link layer may attach a header and trailer.

But at no point does any layer look at the data that a higher layer sent.

And you also notice that in this architecture, what we've shown here is that only the link layer and the network layer of the switches that are forwarding packets are actually processing the packets.

So the end-to-end layer, by definition, is not involved in the forwarding of the packet.

The end-to-end layer is only involved when one of the endpoints of the communication is involved.

So what we're going to talk about today, we're going to finish very briefly our discussion of the link layer.

And then we're going to turn and focused mostly on the networking layer.

So do you remember last time?

We got as far as saying that the link layer is in charge of a number of sort of important issues with the transmission of data across one link of the network.

And we talked for awhile at the end of class last time about this analog, to digital, sorry, got that backwards digital to analog to digital conversion.

We are going to talk about the other issue, though, that I said we needed to talk about in the context of the network layer or the link layer is the issue of framing.

So the idea with framing is when you're sending a message out over a network link, the receiver on the other end needs to have some way of knowing that a packet is starting or a packet is ending, right?

So as we call these packets when they are at the link layer, frames.

So, the issue with framing is to identify the sort of beginning and end of every one of these frames as it transmits over the network.

And there is a sort of fairly obvious way to do this is, well, attach some special symbol, but some special symbol at the beginning and end of every packet.

So, for example, if we were looking at Ethernet, the payload of an Ethernet packet might contain the destination address, the source address, the type.

So we'll talk more about what the type field means in a minute, the data, and some checksum information that can be used to detect errors.

And the preamble is a special code that is attached to the beginning of every one of these messages.

And this is used to make the Manchester encoding, which you remember we talked about last class.

This is used, sorry, to allow the phase lock loop to lock into the message, which we talked about at last class.

And it might be, in the case of Ethernet, it's a well-defined sequence: one, zero, one, zero, one, zero, one, zero.

But remember that with Manchester encoding that the data that's actually transmitted looks a little bit different.

And then following the preamble, there is this start of frame symbol.

And then at the end of the message, there's this end symbol.

So, one thing we might be concerned about is, say, for example what if the network layer tries to send a message that contains the end symbol in it?

that would be a problem, right, because then the end layer would have inadvertently terminated the message, even though this wasn't really the end of the message, this is just something that happened to be the same code as whatever the link layer had chosen for its end of code symbol.

And the reason that this is a concern is, remember, we don't want the network layer to have to understand lots of details about how the link layer operates underneath it, right?

The network layer shouldn't have to make any assumptions about what our valid symbols for it to transmit, and what are invalid symbols for it to transmit.

So the way that we're going to solve this is through one of two techniques.

One: so the first technique that will talk about for a moment is this idea of bit stuffing.

I'll get to that in a minute.

Another simple thing that we could do would be to simply use a code that can't possibly be generated by, say for example, the Manchester encoding scheme.

So if the network layer sends a message like one, one, one, one, the link layer is going to convert that into some sequence of ones and zeros once it applies Manchester coding.

So if the link layer on the receiver sees a message like one, one, one, one, one, that can't possibly be a valid code.

That can't possibly be a valid message that could have been generated by the network layer.

Only the link layer can actually send that sequence of bits out.

So we could tell that would be a terminating symbol.

Another simple way, though, to send one of these end codes is using this technique called bit stuffing.

And the idea is pretty simple, and it's kind of a neat technique that can be used in general when you have to do this kind of encoding a lower layer.

So the idea is, suppose we set our end code was equal to some bit string like one, one, one, one.

And, the solution in bit stuffing is very simple.

What it says is that when you receive a sequence of bytes or sequence of bits from the network layer at the link layer, it says that you should convert any sequence of bytes that look like one, one, one, three ones in a row, into three ones in a row followed by a zero.

So this happens at the center.

And then the receiver just reverses this, says any sequence of one, one, one, zero, gets converted to one, one, one, one.

OK, so you could see that this means, and this transform is only applied to the payload of data packets that it's coming from the network layer down into the link layer.

OK, so you can see that there is no way for the link layer, now, to send a sequence, or for the network layer to actually cause four one's in a row to be sent out over the wire because of this transformation.

So if the network layer tries to send a message that contains a sequence with four ones, what the link layer is

going to send is three ones followed by a zero followed by a one.

Notice, however, that this also means that if the network layer tries to send a sequence of three ones followed by a zero, the link layer will transmit three ones followed by two zeros, OK?

And then at the receiving end, we can just trivially apply this reverse transformation.

So this idea of bit stuffing allows us to guarantee that any time that the link layer on the receiving side actually sees four ones in a row, this is really the end symbol as opposed to the network layer trying to transmit four ones in a row, OK?

So that's all I want to say about the network layer or about the link layer.

And what I want to do now is to move on to a discussion of the network layer.

So the network layer -- -- has two primary functions that we're going to talk about today.

The first was forwarding.

And the second is routing.

So the idea was forwarding is as follows.

So the idea was forwarding is basically to allow nodes to decide, given a particular packet to allow them to decide what the next hop for that packet should be by looking at the destination address deaths in the message that they are trying to transmit.

So they're going to look at the destination address and make some decision about where to send this packet next.

And they're going to do that by keeping a table that basically maps every address into the next link to send to.

So let's look at a really simple example.

Suppose we have five machines, A, B, C, D, and E -- -- connected as follows.

And let's number these links.

Let's number this link from E to B.

We're going to call it L1. We're going to letter this link E to D L2. And then, I'm going to number all the links sort of in the same way.

So, B's link to A.

I'll number L1. B's link to D I'll number L2, and B's link to E I'll number L3. So, notice that this link, I've given two names to this link depending on whether we're talking about it from a perspective of B or E.

OK, so now we can do the same numbering for all of the other links.

So, call this L1. Call this L2. Call this L3. And, this one is L1, L2 from C's perspective.

And this is L1 and L2 from A's perspective.

So now we have this labeled graph here.

And now let's look and see what the forwarding table for one of these nodes might look like.

So, for example, if we look at the forwarding table for node A, what we'll see is one entry for each of the other nodes that are in the network.

And this entry will tell us, given a message destined for this, whatever address is in, this is the destination address.

So given a particular destination address, it will tell us what the next link we should use is.

OK so, and this is the forwarding table for a particular node.

So in this case, we're looking at the forwarding table for node A.

So, if A sees a node destined to node A, what should it do with it?

Well, it's destined to its local mode.

So it's going to do the obvious thing, which is send it up to the end to end layer.

So I'll just write E to E.

So if it receives a node destined for B, it's going to, presumably it wants to send it along whatever the shortest path is.

And we'll talk about how the next step, routing is actually deciding which thing should be there.

But, for example, it might have L2 as the next link.

OK, and if it wants to send to C, it might have L1 as the next link.

So, A to B is using, it's using L2, and A to C it's using L1. OK, now, node D is, say if it wants to route to D, it's going to have to route through either B or C.

And so, for now, let's just say it routes to L1. And so, it routes to C, and then it's going to allow C to go ahead and forward the packet onto D.

If it wants to route to E, well, again, it can either route through B or through C.

But let's say maybe it wants to route through B because it has a shorter path.

So, we'll write L2 here.

OK, so this is just a very simple example.

Now you can see that any time that A wants to send a packet, it has a next hop that it should use for every destination address within the network.

So, this is very simple.

And this is sort of a high level the way that forwarding works.

Of course, there are some other details associated with getting forwarding to work properly.

In order to sort of talk about how forwarding actually works, the interaction between these layers, what I want to do is just give you some examples of what the packet headers look like for different kinds of protocols that are used in the real world.

So we're talking about here, this is the IPV4 header.

So, remember, IP is a protocol that runs at the network layer.

And this header has the following information in it.

So, what I've shown here down the left side is a sequence of words.

So these are 32-bit words.

And I'd just broken up the bits by number along the top.

So, the first sort of word has information about the version of the protocol that's running in it.

So in this case the version would be four because this is IPV4. There's a new IP protocol that is in the process of

being deployed called IPV6, which you'll sometimes see referenced.

There's a header length field.

It just specifies the length of the header.

There's a TOS field.

This is type of service.

It's typically not used in most packets.

And then there's the packet length which is the entire length of the whole packet, not just the length of the header.

So this should be fairly straightforward except for the type of service field which you guys don't need to pay any attention to.

So the next field, so there's this identification.

The next [word?] has identification, flags, and fragment offset.

Let's just look at the next one.

And the next one has time delay of end to end protocol and checksum.

So the important fields, I'm blanking on what the identification field contains, which is why I'm stalling.

So the identification field, so let's just talk about the other fields and we'll come back to identification if I remember it.

So the flag's field simply contains some information about whether this packet has been fragmented or should be fragmented.

So fragmentation is something that this packet can be split into multiple sub packets.

You don't need to worry about it too much.

And the fragment offset as if this packet has been split into these sub packets, these fragments.

Then this is the number of the fragment that's being transmitted so, the interesting ones now come in this next row.

So, we have the time to live flag, the end to end protocol, and then the checksum.

So, the time to live field is sometimes abbreviated TTL.

And time to live is a little bit of a strange name for it.

Basically what this says is that as this packet is being forwarded through the network, we're going to decrement the time to live by one on every step that it's forwarded through the network.

And if the time to live reaches zero, we're going to stop forwarding this.

So the reason that we care about time to live is that there can sometimes be loops and are fording routes.

So suppose that we had set this up so that A sends messages destined for, say, A sent messages destined for E through C.

But C sent messages destined through E through A, right?

So that would be a loop, and that would be a problem.

So, we're going to try and avoid forming these loops when we do our routing protocol.

But they're going to be certain situations when those fail, for example, that loops can occasionally occur.

And we're going to use the time to live field to eliminate those.

The next thing is the end to end protocol.

So the end to end protocol is the specification of which end to end protocol we should forward this message onto.

So I'll talk about that more in a second.

And then there's this checksum field, which can be used to determine whether the message was fully formed, although it turns out that in most cases in IP, this field probably isn't used.

So, and then there is the source address and a destination address.

So these are IP addresses that identify the endpoints of the packet and then, finally, there is some additional, optional information which can specify a huge range of different things.

It can be up to 44 bytes.

In this case, you don't need to worry about it.

And then finally there's the payload, which is the actual message that was sent from the end to end layer into the IP layer.

So what you see happening here, if we look at a diagram of what the interface between the end to end layer, what the interface between the network layer and link layer is.

So let's just look at our three layers again.

We have our end to end layer.

We have our network layer.

And we have our link layer.

And one of the things that we can pick out here is that notice that we have this end to end protocol that's actually in our IP packet so what this means is that if I have a network protocol like IP that's running here, when it receives a packet destined for its local address, say, destined to itself where it's supposed to forward it up to the end to end layer, it can actually dispatch this packet to a number of different end to end layers.

So, for example, we talked last time about the TCP protocol.

But there are a number of other end to end layers that can be used in the Internet.

So we'll talk in this class a little bit about a different protocol called UDP.

We'll talk about this more.

We'll talk about the end to end layer more next time.

But the point is that this up call to the layer above us is controlled by the contents of this header that we have.

So similarly, if we look an Ethernet header, what the Ethernet header has on it is the destination address, which is 48 bits, and the source address which are 48 bits, followed by a link protocol.

So the link protocol says, and so the Ethernet, remember, is a link layer protocol.

And what happens is, so if we have Ethernet here, it has a protocol ID that specifies the protocol that it's supposed to send messages back up to.

OK, so these fields, one question you might ask is, well, how does the Ethernet layer know which link protocol it should send messages to?

Or how does the network protocol know which end to end protocol to send its messages to?

So it's kind of interesting and it's worth looking at the pseudocode for one of these protocols just quickly in order to understand this a little bit better.

So let's look at the pseudocode for the function which we're going to call Net Handle that accepts messages either from the end to end layer to be sent out across the network, or from the link layer when a message is sort of received as it's forwarded through.

So we can use this procedure both for down calls and up calls, and it accepts a packet which, say for example, has this format that I've shown here, which is the same as the format that I've shown on the other page.

So the first thing this is going to do is it's going to check and see if the destination of the packet, for example, is the local address.

And if the destination of the packet is the local address, then we know what we should do with it, right?

We're going to pass it up to the end and layer.

And we're going to send, so what we're going to do is we're going to de-encapsulate this packet, right?

We are going to strip the header off of it.

So we're just going to send the payload up to the layer above, but we're going to need to dispatch to the appropriate layer, which we're going to do by sort of saying and naming what the end to end protocol is that we would like to dispatch to.

And there may be some other options that we pass along with this as well.

So that's what the three dots there mean.

So now, if this isn't for the local packet, for the local node, then we need to send this packet out again, right?

So what we're going to do is we're going to check the time to live on this packet.

And if the time to live is greater than zero, then we are going to decrement the time to live.

If the time to live is less than or equal to zero, then we know that we're not going to forward this packet anymore.

We're just going to drop it.

So the time to live gets set initially by the first guy who sends the packet out.

He initializes it to some value, which is the maximum number of hops that he wants this message to be propagated for.

And so, after we decrement the time to live, we're going to look up in our forwarding table the next link that we want to use.

OK, and what I've shown here is that we're going to look up the sort of protocol to use and the name of the next link.

So, for example, this IP layer might have Ethernet underneath it, which might be one protocol.

But there might also be a WiFi layer that's here that we could also send messages out.

So we could send out through either one of these protocols.

So we look up the one we want to send out, and then we call this link send routine, which actually does the transmission out over the appropriate named link.

Notice now what I've specified, I have this [NETPROT?] in all caps here.

So in this case, this should be, for example, IP.

It's the name of the network protocol.

So when the network protocol sends the message to the link layer below it, it specifies what the network protocol is that wants to have returned, that it wants to be called on return.

So it says, please link layer; when you have received a message that needs to be sent up to the network layer, dispatch it to the network layer named NETPROT.

OK, and then for example the TTL we can't transmit the message out.

TTL has expired.

If we've sent it too many hops already, then we're going to need to do some kind of error handling.

In this case, error handling might be, send a message back to the source address of this message, and tell them

that the TTL ran out on this message.

That will be one thing you could do.

Another thing you might do is simply drop the packet or ignore the error.

OK, so this is just a simple example of how forwarding might work.

And what I wanted to use it to illustrate is the fact that there can be multiple different link layers at the bottom that can be dispatched to.

And these link layers can, in fact, dispatch up to multiple different network layers as well.

So in today's Internet, it's uncommon to see, you'll rarely interact with anything that's not using IP at some layer or some network layer.

But there have, in the past, been a variety of different network layers that have been proposed.

You guys may have seen an old protocol from Novell called IPX, or you may have used AppleTalk, which is an Apple protocol that in some varieties also runs at the IP layer.

Finally, I just want to point out that the last little layering detail that I want to get to is that if you think about the link layer from the perspective of the network layer, the link layer is simply, the link just looks like one hop to one more connection.

But in fact, these links can, themselves, be networks.

So it's just kind of a weird statement.

But here's a simple example of what I mean.

Suppose that this link layer, that one of the link layers that the IP layers can send a message to is a modem connection, OK?

But if you think about what a modem connection is, right, it's a connection out over a phone line.

And a phone line is, itself, right, as we talked about at the beginning of class, a phone line is, itself, a kind of the network.

Right, it's this kind of circuit-switched thing that goes through multiple ones of these circuit switches as opposed to going through these.

So underneath the modem, the modem layer, which looks like a link to the network layer, there's actually, these link layers can actually themselves be composed of a number of links, and they can, themselves, be a type of a network.

So it's sort of worth realizing that there is a kind of hierarchical relationship, one, but the link layer may actually itself be a network consisting of multiple links, but from the point of view of the higher-level link layer here, it's just a single link, a modem connection to some endpoint that we can send a message over.

OK, so what I want to do now is take you through, we're going to switch gears a little bit and talk about the sort of next piece that I said we needed to address, which is this issue of routing.

So I said we're going to talk about forwarding, and then we're going to talk about something else called routing.

So routing is the process by which we build up our forwarding tables.

OK, so what I showed you here was I just sort of told you what the values that should go in these forwarding tables should be.

But I didn't tell you how they were derived or where they came from.

So in this case of the simple network, it was relatively easy for us to, by hand, sort of come up with a set of possible forwarding paths that seem reasonable.

But you can imagine that if this network had scaled up to a million nodes, that's not something that we want any individual to have to do.

No person should have to configure all these routing tables or all these forwarding tables.

So instead, what we're going to do is we're going to use this process called routing in order to build these forwarding tables automatically.

And we really want our routing protocol to be three things.

First, we wanted to be scalable.

And the obvious way in which we want it to be scalable is the way that I just said.

We don't want to have to have people, we want this thing to scale up to, say, a million nodes or several million nodes and be able to continue to work.

We shouldn't have to have people sort of involved with configuring, building up these forwarding tables at every step of the way.

We also want it to be robust.

So by that, I mean it should be tolerant of faults.

If a node fails in the network, the network should eventually discover that that node has failed and be able to forward packets around the failure or be able to compensate for the failure if at all possible.

Finally, we want this routing protocol hopefully to be distributed.

So we don't want to have to have one machine that's responsible for setting up the forwarding table on all the other machines.

We don't want to have one machine that needs to contact all of the other nodes and collect information about what all of their links are, and assemble all of those links together into one global forwarding scheme.

And this really gets at the scalability again.

So what I want to do is to take you through the process of routing.

Before I do that, I just want to take a quick digression.

We're going to start off with very tiny networks on this, and just talking about a very simple baby network.

It just so that you guys don't feel like this is completely unrealistic, I want to show you that in fact the Internet at some level started out like this, too.

What we're going to do it in the course of this class is sort of build up from these very simple networks that maybe look the way the Internet did at first, the schemes that are more like what is actually used in today's production Internet.

So, sorry I went back on you guys.

OK, so this is a picture of what the Internet look like in 1969. So you may not be able to read the labels on these things, but we're looking at three nodes here.

This is UC Santa Barbara on the coast of California.

It's southern central California.

This is Stanford Research Institute, SRI, which is the Bay Area.

This is Utah, and this is UCLA.

So, this was the ARPANET, which was a precursor to the Internet, and was sort of one of the very first of these large wide-area networks that was ever developed.

And each of these little square nodes here is a machine that's on this.

So there were four machines on the ARPANET in 1969. And there were these four routers that were being used.

So, this is 1971. So by 1971, there still is a cluster of these machines in California.

But you notice that Illinois, Carnegie Mellon, and Boston have suddenly appeared on this map.

So, MIT is here, Lincoln Labs is here, Harvard is here, BBN, which is another large company that does a lot of networking research in this area is here.

So, the network has started to evolve.

And in particular, you notice that there are now these sort of two clusters, these two regions, one on the West Coast and one on the East Coast that have a number of nodes.

So, it just keeps growing and growing.

So, by 1980, you see the network has gotten substantially larger.

And one of the interesting things about this is you are starting to see a diversity of links.

So notice that Hawaii is now connected into California by way of a satellite connection, as is London.

So, we now have not only just these wires that are running, but in fact we have wireless links.

But at the same time this was happening, all the protocols we've been talking about were being developed.

And one of the whole goals of this was to be able to support these multiple different kinds of links on top of the standard networking protocol.

OK, so by 1987, this thing has really become, turned into a number of decentralized networks.

There's this large network called the ARPANET.

There is something smaller called the NSF backbone, and a number of other networks that are military networks, and so on.

These are all connected together, and they're all being, by 1987 they were all running this TCP/IP protocol that was being used to exchange information between all of these things.

So roundabout a few years after this, right, the sort of World Wide Web suddenly happened, and there became this huge commercial interest in the Internet.

And that has really just sparked this explosion of nodes, and made the network just huge and incredibly vast.

So it's hard to see this, but in the middle of this you notice that there is a little orange node here.

This bar on the left side is showing the out degree of the nodes in the network.

So this number is 2,977. So that means there's a node at the center of the Internet that has 2,977 links to other nodes in it.

OK, so this is a really incredibly large network.

And you notice that all of these 20 or 30 bright pink red nodes here have hundreds to thousands of outgoing connections on each one of them.

So there is this core of the Internet that is very highly connected to a very large part of the network.

And now, down or out around the edges are these much smaller networks that have much lower collectivity.

And these are things like service providers, for example, that consumers might pay some money to get a connection from.

So this is the sort of core of the Internet.

These are the people who are not so much the end-users on the Internet, but the service providers that are providing connectivity for the end-users. Each one of these little nodes represents one of those service providers.

This was as of 2003. OK, so what I want to do now is to start as I said.

We're going to sort of start off with a baby version of a network.

In particular, we're going to look how forwarding, or how routing might work in this very simple network that I've shown here.

So let's see what happens.

So we're going to use a protocol that we call path vector routing.

And the idea behind path vector routing is that we're going to build up the paths from, that is, the sequence of nodes that we should forward messages through in order to reach a particular destination.

And the way this is going to work is we're going to have two steps.

So, let's put it over here.

So routing is going to consist of two steps: an advertisement phase or an advertisement step, and an integration step.

And the idea is that during the advertisement step, each node is going to advertise what other nodes it knows how to reach.

And then during the integration step, each node is going to take all of the advertisements that heard during the previous advertisement step, and integrate them into a new set of routes that identifies the new set of nodes that this node can reach.

OK, so this'll be very clear when I show you the example.

So let's just look at the case of all the nodes, figuring out how they can reach node E.

So what's going to happen is that first node E is going to send out an advertisement that says that node E knows how to reach node E, right, which it obviously does.

And the way that it reaches node E is simply by forwarding a message up to its end to end layer.

So it says, to reach node E, come this way.

And it sends it out over the two links that it has to node C and D.

So when node C and D hear this advertisement, during this integration step what they are going to do is to add this information about this connection to node E, and they're going to store which link it is that they send that message out over.

They should send messages out over in order to reach node E.

So node C is going to store link one, and node D is going to store link two.

But in addition to that link, they're going to store the path that they use.

But now what's going to happen is that each of C and D, during the next advertisement step, C and D are going to also advertise the information that they have about the nodes that they can reach in the network.

So, I'm only showing the advertisements for node E here.

But of course, at the same time, C and D are also advertising the fact that they can reach themselves, and maybe their ability to reach other nodes in the network that we haven't shown.

So we're just looking at E, but bear in mind that all the advertisements are being done for all the nodes at the same time.

So, C sends out an advertisement that says I can reach node E, and the way to do it is to send, and I can reach node E via C and then E.

And similarly, D sends out a message that says I can reach node E by a D and then E.

So, OK, up to this point we haven't really seen anything very interesting.

But now during the next integration step, we see that these two nodes, B and A, now have heard an advertisement that gives them a path that allows them to reach node E.

So now, every node has a path that allows them to reach node E.

But this advertisement and integrations process is going to keep going on.

So, for example, nodes A and B are going to advertise that they can, also, reach node E to their neighbors.

And in this case, it's probably unlikely that any of the nodes would like to switch to a new route.

So, for example, it's not clear, there's no reason, for example, that A would want to forward its message.

It's probably unlikely that A will want to forward its messages through B to reach E, right, because that's going to be a longer path than for A to send its messages simply through C and then to E.

But B doesn't actually know whether or not A knows about E yet or not.

So it needs to continue to broadcast this information out.

So this is pretty simple.

It's pretty clear how this works.

And once this process has been running for a while, you can see that the network is going to converge into a state where every node has, as long as the network is connected, it will converge into a state where every node has a path to node E, right?

And the amount of time that it'll take for that to happen is equal to the maximum number of hops that any node is away from node E.

So, once this node has converged, now we can trivially build up our forwarding table simply by pulling out the link number from each one of these nodes.

So, for example, we can see that E's forwarding table simply says: to reach node E, you send it to my end to end layer.

D's forwarding table would say, to reach node E, you send it over my link, L2. C's forwarding table would say, to reach node E, you send it over L1. B's forwarding table would say, to reach node E, you send it out over my L1, and so on, OK?

So, once we've done this path vector routing, at the end of this process we will know which links, we'll have built up the forwarding table that we can use for sending our links.

OK, so this is a very simple process.

But now, let's look at what happens when something, so what we said here is that each advertisement step, and after each advertisement, we're going to go ahead and do integration.

Integration, basically what we're going to do is we're going to try and pick the best path in some way.

What we've shown here is just simply picking the shortest path.

So we've picked the shortest possible path for every node to reach node E.

And in case it wasn't clear, I didn't explicitly state this, it's important to realize that nodes are going to ignore advertisements with their own address in the vector.

OK so if, for example, when node E hears node D advertising that it can reach node E, node E is going to say, oh, well, I am node E.

I don't actually need to pick up this path.

I don't need to send my packets to myself through node D.

That would be a silly thing to do.

That would create a routing loop, which is something that we presumably don't want to do.

So, this is a simple way using these path vectors we can use to avoid creating routing loops.

OK, so now let's look at what happens, something a little bit more interesting.

Let's look and see what happens when, for example, there's a failure.

So this is just exactly the same network that I showed you.

But, suppose for example that the link between D and E fails.

OK, so now node D no longer has a route to node E.

But remember, the way that I described this is that this advertisement process is just going to continue going on in the background, right?

So what's going to happen is that at some point node D is going to realize that its link to node E went down.

And, node D is going to cross this table out of its entry.

So, node D is going to basically stop hearing advertisements from node E.

When it stops hearing advertisements from node E, eventually it's going to do something.

It's basically going to expire that entry from its link table.

So after it hasn't heard advertisements from node E for a while, it'll expire that entry.

And then, sometime later it will hear an advertisement from node C saying I know how to get to node E.

And now, node D can go ahead and integrate this new path into its routing table.

Now, notice that this process is just going to propagate through the network.

So, once node D stops hearing advertisements for how to reach node E, it's going to stop sending advertisements

for how to reach node E.

And so, similarly, because B is also routing its information through, had previously been routing its packets to reach node E by way of D, it's going to say, oh, well, I stop hearing about this route, DE, from node D.

So I'm going to stop using this.

And instead, then sometime later, it's going to hear about this new route, DCE, and it's going to integrate that into its table.

So, it's this process where there is this sort of process whereby nodes continually advertise and integrate routes.

And there's this sort of interesting thing which happens, which is that nodes forget about routes that they haven't heard about for awhile when they miss an advertisement.

So this, forgetting about routes is an important sort of principle that's often employed in networking called soft state.

And the idea with soft state is that you should only keep information when that information gets refreshed.

So all the information that you have has some time limit on it.

And when you haven't heard that information refreshed after some time limit, you throw it out.

So that's what we're doing with the routes here.

And so we only keep routes that we have heard advertised recently, basically.

And that has this nice property that it allows us to adapt to faults within the network, right?

So we saw a failure happen.

We saw that sometime after that failure, this soft state property would cause the information about the link between D and E to disappear from the network, and then nodes would rediscover their new links that allow them to connect to node E.

OK, so what I want to do now, so this is sort of the basic process.

And, path vector routing is fairly similar to the way that routing in the Internet actually works.

For next time in recitation, you're going to talk about a protocol called the border gateway protocol.

And you're going to actually study in much more detail how Internet routing works.

But this is a nice simple model of how routing in a small network might act.

So the problem with what we've discussed so far, while we are here, is that if this is a very large network, these number of routes that you're going to have to hear about is really huge.

So suppose that this, instead of being a five node network was a million node network.

Well, now every node is going to have a table that's a million entries long, right?

That's going to be really big, and it's going to have to hear a million advertisements.

And every time there's a failure, well, that's going to be a pain because we're going to have to wait for that information about that failure to propagate through the whole network.

So in some sense, this simple path vector routing protocol we have doesn't really meet the scalability goal that we want to scale this network up to a very large size.

So we have an issue with path vector routing and its scalability.

OK, so the solution is a solution that is often used when we have a scalability problem in a computer system: its hierarchy, OK?

So let's see what I mean by that.

So, you remember when I was showing those pictures of the Internet, there are these different kind of subnetworks that were forming over time on the West Coast, on the East Coast, or there was the ARPANET, and the NSFNET, and the MILNET [SP?] that were these sort of different networks that were all a part of the Internet as a whole.

But they were these sort of different regions that were separately administered, and that often corresponded to a specific organization like the NSF or like the military.

So, oftentimes these regions, so it's very common when you look at any network to have these kinds of regions in them.

So, for example, clearly there's a network that is MIT's network, right?

And Harvard, for example, has a network that's Harvard's network, right?

And those two things are sort of logical regions that define different parts or different groups within a network.

So if you were to look at any network, you would see that sort of almost any large network is organized in this way into these regions.

In the paper next time, we're going to call these regions autonomous systems or AS's, OK, so autonomous as in sort of operating on its own, operating without being dependent on the other parts of the system.

So for example, if MIT's Internet connection went down, connection to the outside world went down, that wouldn't stop you from being able to connect to machines within MIT, right?

So MIT is autonomous in the sense that it continues to operate in the absence of its connection to the rest of the Internet.

So let's look at a simple example of a network that has this hierarchy property, and see how we would modify the routing algorithm that I just talked about.

So suppose I have two small networks, each with three nodes in them.

Let's call them A, B, C, and D, E, and F, OK?

So, these are each going to be autonomous systems, which I'll draw by drawing a circle around them.

OK, so one way we could be routing would be to do what I had shown before, which is that each node within, say, this autonomous region which I'll label one, or autonomous system one, and this one two, would have information about all the other nodes everywhere else in the network.

But that has the scalability problem that we mentioned.

So instead, what we want to do is we want to make it so that only a few of the nodes in here, so that we don't have to have information about all of the nodes that are anywhere within the network.

We only have to know about a few nodes we say are on the edge of each one of these networks.

So the idea is as follows.

Suppose these are connected in this way.

And what we're going to do is we're going to appoint one node within each one of these regions.

For example, it could be several modes.

One of these regions to be a so-called border node that sort of sits on the edge of these two regions.

And we're going to connect just those two nodes together.

So we're only going to have a small number of links between our two AS's.

And now, if we look at the forwarding table for one of these nodes within, say, AS1, it's going to look as follows.

So what I'm going to do is I'm going to write the addresses for these different AS's as hierarchical addresses.

So, we're going to call node A's address 1.A. And, we'll call node E's address 2.E, OK?

So, what our forwarding table looks like is a list of addresses, and then a link to use.

OK, so this is going to be the forwarding table, again, for node A.

So, A is going to have an address.

It's going to have an address 1.A in it.

And, the link to use, in that case, we've already said is end-to-end. OK, it's going to have an address 1.B. So, to get to B, A is going to route by this link here, which let's call this link number one, OK?

So, it's going to route to link one.

And it's going to have a connection to 1.C, which is going to route via this link here, which let's label that two, OK?

So now, we don't have any information about how to reach network two.

So what we're going to do is rather than storing information about every machine and network two, we're going to store just information about how to reach the edge of network two, and then trust that network two is going to be able to route to anybody whose address begins with two.

So what we're going to do is we're going to say two dot star, right, two dot everybody.

So, [we'll star?] the character that says anybody whose address has this prefix two dot.

So, anybody whose address has prefix two dot, we are just going to send to node C.

So, we're just going to send that over link two, OK?

And so, similarly on the other side, we're going to have a table.

Each of these nodes is going to have an entry, one dot star, that specifies that it should route messages for network one through node D.

OK, so you see in this way now what we've been able to do is to make it so that each one of these autonomous systems, sort of each node only knows how to route to other nodes within its sort of group, and that in order to cross groups, you route through one of these special sort of border routers or border nodes, OK?

And, we're going to talk about the way that this border routing protocol actually works in the Internet in recitation.

But you can see that what we've done is we've accomplished this sort of hierarchy so we can make this system, you can make these tables much smaller by including these star entries in them.

It's worth just mentioning as a final caveat that we have given up something for doing this.

OK, what we've done is we've forced every node that's within one of these regions.

Now, this node's address, in order for this node 1.A to be able to continue to operate, it can only be connected to somebody like C who can advertise information about all the nodes whose names begin with one.

So, we've basically forced this sort of network to have this.

By imposing this kind of a hierarchy on the network we've limited, to some extent, the ability of these nodes to move between networks because node A can only be advertised by way of node C.

So in the Internet today, there's actually multiple layers of hierarchy.

So you may have noticed that Internet IP addresses have the form A.B.C.D. So, it's a four layer hierarchy that we have in the Internet.

You guys will learn more about this on Thursday in recitation.

And what we'll do next time is talk about the end-to-end layer, and eliminating some of the limitations of the best effort network.