Let's go ahead and get started, you guys.

OK, so before we begin, I just want a brief note about the quiz.

So you should get the quiz back tomorrow in recitation.

The mean and median on the quiz were 69 out of 100, so look for it tomorrow.

And there should be solutions post it also tomorrow.

Does anybody or member with standard deviation was on the quiz?

OK, thanks.

OK, so the topic for today, we're going to continue talking about networking.

And just to recap where we left off last time, I wanted to review a few of the best effort network properties.

So you remember at the end of lecture last time we talked about this notion of a best effort network.

And we said that these packet switch networks like the Internet are typically best effort, which means that there are certain things in particular that they don't guarantee.

So one property of best effort networks is that they're subject to delays.

So these are delays due to the propagation of messages down our wire, the time it takes for the messages to propagate along the pipe.

Remember we talked about that analogy last time?

Time to transmit the message, so transmission time is sort of the amount of time it takes proportional to the length of the message and the bit rate of the link.

And then, there is additional delays that are introduced by these properties that we talked about at the end of the lecture last time such as queuing.

So the fact that remember we said that these networks are congested and the load is somewhat unpredictable.

And because the load of the network is unpredictable, that necessarily means that there are going to be queues in between the links on our network.

And then finally, additional sorts of delay that we talked about, but also can be an issue as the processing delay.

So this is the delay incurred by sort of the overhead of processing messages between switches, for example, in our network.

OK, so other interesting properties of best effort networks, they incur some loss.

OK, so there are going to be messages that are corrupted as they are transmitted down these links.

There are going to be losses that are caused by congestions.

Remember we talked about how one of the only responses, oftentimes the best response to congestion is simply to drop packets to reduce overload.

And there are sometimes going to be failures that are going to be switches within the network that will sometimes simply cease to function.

And of course you're going to lose the data when that happens.

And there can be reordering of packets.

So you didn't talk much about this.

But it's fairly sort of obvious idea that suppose an application has two different messages, right?

Well if you think about the way that those messages may be routed along different paths through the network.

And we'll talk about this more when we talk about how routing works next time.

But because they can be routed along different paths, there's no guarantee that the messages, because message one was sent before message two, that message one actually arrives before message two at whatever endpoint you're sending to.

And finally, there can be duplication.

And duplication is often a side effect of techniques that we'll again talk about next time for dealing with this issue of loss.

So if you have lost packets, somewhere in the network you may want to try and retransmit those packets.

And because you're retransmitting things, you're sending things multiple times.

There can be places in the network where you actually see two copies of a message.

So these best effort networks have a number of sort of properties that make using them somewhat complex.

And what we're going to talk about over the next two or three lectures basically are ways of mitigating the complexity of these best effort networks, of making the interface that these best effort networks provide to applications a little bit more usable in the sense that we're going to try and find ways to reduce the effects of things like loss and reordering are to make it so that applications don't necessarily need to be aware of those things.

So in order to start getting at sort of dealing with complexity, we're going to talk today about this notion of layering.

So -- So when we talked generically about systems earlier in the class, we said that a standard way that we're going to deal with complexity is by introducing modularity.

And the specific kind of modularity that is widely used within networks is layering.

And that's what we're going to talk about most of the time today.

The other way that we typically deal with, another common way of dealing with complexity with the networks is by using protocols.

And a protocol is really just a set of rules for how two machines on the network do communicate with each other.

So, for example, a protocol might say that if I want to tell you about, if I want to send a message to you that I will put this certain bit of information at the beginning of it, I'll put some address information at the beginning of that message.

And the message will be no more than a certain number of bytes long.

And I'll put that message on the wire, and then part of the protocol is also that you will send an acknowledgment back to me that tells me that you in fact got the message.

So these protocols are very well defined sets of rules about how communication is supposed to happen.

These protocols are going to help us to deal with complexity because they are going to tell us what we expect to have happen within the network.

And when things don't follow the protocols, then that's going to be an indication that something went wrong, and that we should therefore try and recover from it.

So, for example, if we lose a packet, the protocol is going to allow us to detect that a packet was lost and request that that be retransmitted.

We'll talk much more about protocols next time, but it's sort of worth bearing in mind that almost at any time, two things are communicating within a network that are using one of these protocols.

And a protocol is just a set of rules.

OK so I want to just quickly give you this list of five questions.

We're going to come back to this list of five questions a little bit later.

But these are sort of five key questions that we need to sort of figure out how we're going to actually resolve in networks.

And the first question is, how do we multiplex conversations on a network?

When we talked about this last time, we have these two different alternatives, time division multiplexing, which we said was used in telephone networks.

And we talked about packet switching as a way in which we can have multiple people sharing a network.

But there is a bunch of other questions that we haven't really answered yet.

And this is what we're going to get at over the next few lectures.

So how do we transmit bits on a link?

How do we actually, given a wire, how do we actually modulate that wire in the right way to transmit the message that we want to transmit down it?

How do we forward packets via switches?

So we sort of said that there are these switches, and that along any communication path, there may be sort of multiple communication hops that have to be taken through multiple switches.

But we haven't yet talked about what is actually going on inside those switches.

So we are going to look in particular at what's going on inside switches in the Internet and see how that works.

Then there's this question about, how do we actually make communication reliable?

So we said that one of the limitations of a best effort networks is that, well, it introduces loss, and it can reorder packets.

Applications oftentimes don't want to have their packets reordered or dropped, right, because they're trying to actually exchange information with each other.

So we'd like to understand what kinds of techniques we can use that can systematically allow us to avoid, or to make communication reliable.

And then, finally, how do we manage congestion?

So we said that one of the fundamental problems with these packet switch networks is that the amount of load may be unpredictable in the network.

So the question that we need to ask, then, is, OK, how are we going to manage this congestion?

What are we going to do in response to congestion in order to sort of minimize the effect that it has on the applications that are trying to run on the network.

OK, so we'll come back to these questions in a minute, and as we talk about what the layers of the network stack are, we'll sort of see how these different questions fit in to these different parts of the network stack.

But what I want to do now is really turn to this issue of layering.

OK, so in order to understand a little bit about why there might be wires within a network, or a why there might be layers within a network, excuse me, let's look at a very simple kind of a network.

So suppose we have some client, a client, C, OK, who has a number of different connections to the outside world available to it.

And, it can connect each of those connections go, say, for example, to some switch which may in turn connect to another switch which may in turn connect to some N host S, server S, that the client is trying to communicate with, OK?

So, suppose C is trying to send a message to S.

Let's think a little bit about how this might actually work.

So we have the application, which is perhaps running on C.

And, it might try and send a message by calling some routine send S message.

OK, so it says send to endpoint S this message that I have.

So let's think about how we might actually go about implementing this.

So one way that you might have the application send a message to S is that you might have this sort of the application might understand everything about what the whole network topology looks like, right?

So, it might understand, know about all the different links that are available to it, and with each one of these different links, it may understand what the topology of devices that are out there.

So if I have a connection to the Internet, that would mean that I have to understand all of the machines that are connected to me via the Internet, right, so I have this list of a million hosts, and [UNINTELLIGIBLE] every client, and I sort of just scan down this list of a million machines that I can connect to until I find S.

And then I send a message out over the next link or something, right?

So this sounds very complicated because it forces the application to understand the sort of entire functionality of the network underneath it, how the network is connected, and how the nodes talk to each other.

So that doesn't seem like a very good idea.

Right, instead we'd like to application simply to be able to just send its message out.

And we would like some lower layer service to take care of the details of figuring out where to send the message next.

So, in networks, we talk about that sort of service that runs underneath the application, that's in charge of figuring out what the next sort of connection, the next top to use within the network, we call that the network layer, OK?

So, I'm going to abbreviate as NET.

And so, what the network layer, for example, suppose this client had three connections available to it.

It has a modem connection.

It has a WiFi connection.

And maybe it has an Ethernet connection.

What the network layer is going to do is it's going to look at this name, S, that the client has specified.

And it's going to try and decide which one of these links is the next link to use in order to forward the message on to S.

OK, so it's just going to make a decision from amongst the available connections.

So, it's going to basically pick the next link, OK?

And if this is the Internet, you're going to have these switches or these routers that may have links to a bunch of other networks.

And so in the Internet, the network layer runs actually on all of the routers within the Internet, and is making these decisions about how to forward packet after packet.

So we'll see again in the next lecture and in recitation how the Internet actually does packet forwarding.

But so we said there is this network layer.

But we have this thing that's picking the next link to send the message out over, right?

But we don't necessarily want that thing to have to understand the details of how you actually communicate over each of the available physical connections, right?

So suppose that the only thing that was here was this network layer.

And the network layer had to understand how to communicate over Ethernet and WiFi and the modem.

Right, well then the network layer is going to be very complicated because, of course, sending messages out over a wireless radio is very different than sending messages out over an Ethernet.

So what we have is one layer that sits underneath the network layer, which we typically call the link layer.

And the link layer is responsible for managing the physical connection for the transmission of data from a long just one of these wires.

It's the thing that actually moves bits from C to whatever the next top within the network is, OK?

So, these are these two layers.

You notice now, I've left this hole here.

You might be wondering what goes into that hole.

So what we said is so the network layer is responsible for picking the links.

But remember that what we talked about so far in this best effort network abstraction is sort of affairs all these problems, decent sort of with delays, reordering, duplication, and so on.

And it might be that neither of these layers really is responsible for dealing with those problems.

What we said is, what we want, to be able to provide an abstraction for applications where some of these problems with the best effort networks are hidden from the network.

OK, so typically networks introduce a third layer, which in this class we call the end-to-end layer that's in charge of addressing these kinds of issues.

The end-to-end layer may seem a little bit fuzzy, the details of it when we talk about it because the end-to-end layer can do lots of different things for different applications.

OK, so some applications may be concerned about, for example, the possibility of messages being lost, right, whereas other applications may not be as concerned about messages being lost, but may be very concerned about delay.

And we'll see as we talk to the class that delay and loss trade off with each other which makes sense.

If I lose a message and I have to retransmit it, that's going to increase the delay on the network.

But, so it's possible to sort of trade these things off for each other.

So the end-to-end layer is the thing that's responsible for trying to make application environment sort of more pleasant for the application.

And that can mean dealing with trying to eliminate loss or trying to minimize delay, for example.

We'll talk about different kinds of end-to-end layers in just a few lectures.

OK, so these are kind of the three.

So we're going to decompose our network into these three layers.

And in order to sort illustrate this to you a little better, what I want to do is just walk you through a simple example of how the letters might look in the Internet with a simple web application.

And I'm going to use some of that terminology from the Internet here, and we're going to return to some of this terminology, introduced its older more carefully in recitation.

But I'll try and explain it as much as we go.

So suppose we have a laptop, say my laptop here, that wants to connect to a Web server, MIT.edu. And, the way that the Web works is it uses a protocol called HTTP, which is used for making requests for specific webpages, and for returning the results of those webpages.

So these are typically called requests and responses in the HTTP specification.

Now, if you look at, so this is sort of from the user's perspective; the application is a browser that knows about the http protocol, or a Web server that knows about the HTTP protocol.

And it's running on these two remote machines.

Underneath each of these things, of course, there is a set of layers.

And each of these things has layers underneath it.

And these layers correspond to these three things we just talked about, the end-to-end layer, the network layer, and the link layer.

So one thing you may notice here is that the link layer is different between these two things.

So the laptop is perhaps communicating over WiFi, and the Web server is communicating over the Ethernet.

But otherwise these two things have the same; their Ethernet layer and their network layer are TCP and IP.

So, TCP is going to be in charge of basically providing this reliable abstraction for us.

We're not going to talk about it anymore today, except to say that, that it makes communication reliable.

We'll see how it does that later.

What IP is responsible for is choosing the sort of next top to make along each connection of the way, each connection within the Internet.

So, IP is the protocol that runs the Internet.

And, you may have seen IP addresses.

So, IP addresses are the sort of names for the endpoints in the Internet, and takes an IP address and basically gives you this next link that you should use in order to transmit a message out over it.

OK, so suppose that the browser generates a request for some page.

What it's going to do is it's going to call send on the end-to-end layer.

OK, so I've just written this as E to E send to make it clear that this is a send request to the ETE layer.

And it's going to pass some message.

In this case, the message is simply going to be the contents of this request.

And it's going to specify the underlying protocol that it should use as well as the destination address.

So this address is one of these Internet addresses and IP address followed by this colon 80. So, what colon 80 does is it identifies what's called a port number.

And it identifies the application we're running on the remote server that we want to communicate with.

So, typically Web servers run on port.

Now, you say they run on port number 80. That just means that the TCP layer on the other side knows how to communicate, knows that the Web server on the other end is connected on this port number 80. So it gives us a way to communicate, to identify applications that are running on the remote host.

So now what happens is this request is going to be, the TCP layer is going to take this request, which I've shown in blue, and it's going to attach what are called headers and trailers to it.

So these headers and trailers are the information that TCP needs, the TCP layer on the other side is going to need in order to deliver this message to the application.

So in particular, this thing is going to contain this port number that we already mentioned.

So this is going to be used on the other end in order to send the message to the Web server.

It's also going to have a sequence number, which as we will see later, we're going to use in order to, for example, reporter in order to detect reordered messages or detect lost messages.

OK, so the TCP layer now is just going to repeat the same thing.

It's just going to take this packet, and it's going to call some request, say, NetSend [SP?] on the IP layer that sits underneath it.

And again, it's going to specify the set of data that it built up, which we called a segment.

So that's what SEG is, which was the purple and the blue blocks.

And it's going to pass that on to the IP header using this NetSend message to the IP layer.

And it's going to tell the IP layer what IP address it wants to send this message to.

Now, the IP layer is going to take this message, and it's going to put the header on it.

So IP doesn't use a trailer, although it could, in principle, use a trailer.

And this IP header is going to sort of just contain the IP address of the next hop, or of the destination.

OK, now the IP layer is going to do exactly what we said.

The IP layer is our network layer so it's going to do exactly what we said it does before.

It's going to look at all the available links that it has to it, and it's going to send this message out over one of those links that it believes is the correct next top in forwarding.

So it's going to call link send.

It's going to pass this packet on, and it's going to specify the name of the link that it wants to use.

And it may have to specify some address information, for example, what I've shown here is just [cool?] in one just to say whatever machine is one the wireless network at wireless address number one.

So you guys saw sort of a similar addressing scheme being used in Ethernet last time.

You can sort of think of that the same here.

So what happens now is of course the same process.

The link layer attaches its header and trailer.

So, I've called WH and WT for WiFi header and WiFi trailer.

And now at this point we called this thing a frame.

And this frame is now ready to be delivered out sort of along the next top of the network.

So suppose that the network layer identified one particular switch as the next destination of this packet.

It's going to send this out over the network to the WiFi interface of this switch.

The switch is going to receive this message.

What it's going to do is it's going to take the WiFi header and the WiFi trailer, and it's going to peel them off of the message, and it's going to pass the message with just the network header, no more WiFi header on it up to the IP layer.

Now the IP layer now has an exact copy of the sort of message including the IP header from the laptop.

And, so what the IP header layer will do is look at its IP header labeled NH here, and it'll decide what the next appropriate hop to use, to send this message out, it is.

So it's going to then pick the next link to send a message out over and, say, in this case, it decides to send the message over an Ethernet connection.

The Ethernet connection is going to receive the message.

It's going to attach its header and its trailer to it.

And then it's going to send it out to the next link, OK?

So this process just repeats.

The message gets sent to the Ethernet link on the other side.

The Ethernet link forwards the message onto the IP layer.

The IP layer decodes the message, decides what the next link to use is.

In this case, it decides again to use an Ethernet link.

And it sends the message out over the Ethernet.

Finally we get to MIT.edu. And once we get to MIT.edu, we just start forwarding this message up the layers, OK?

So, we do what are called up-calls from the lower layers to the higher layers, notifying them that a message has arrived.

So, the Ethernet layer peels its headers off, sends them up to the IP layer.

The IP layer peels its headers off, sends them up to the TCP layer.

Then remember, we said the TCP layer, so the TCP layer has a port number that's associated with it.

The port number is used to identify the application that should receive this message.

So, the TCP layer pulls out the port and sends it up to the web server, which finally receives our request, OK?

Now the Web server does whatever it does.

It chews on this request for a while, and say, for example, generates a webpage, generates some HTML that it's going to send back to the client.

And now this process just repeats all over again.

The MIT.edu sends a message back down to TCP identifying the client as the endpoint that it wants the message to reach, OK?

So this is sort of the basic way in which we use layering.

What I want to do is now kind of step back and look at, I sort of presented this as a very quick example.

But what I want to do is step back and look at some of the sort of rules that we are following as we use these layers, and to sort of talk about why we have sort of constructivist thing in the exact way that we have constructed it.

So the first rule that we are following here is called encapsulation.

So what encapsulation is, is it's simply this way in which you notice that when we send messages out, each layer associated its own header and trailer with those messages that were sent out.

And it didn't modify anything about any of this sort of data that wasn't associated with the header and trailer for that layer, OK?

So, encapsulation says that each layer may add or remove depending on whether we're sending a message down or sending a message up.

Its own headers are trailers.

OK, but that layer doesn't touch, doesn't look at or use the payload from higher layers.

OK, so the link layer doesn't look at anything that the end to end layer sends it.

It simply treats this as a block of data that it has to transmit, and it doesn't understand anything about what the contents of that block of data are.

It doesn't assume anything about the contents of that block of data.

Similarly, the network layer doesn't assume anything about the contents of the data that's received from end to end layer.

And similarly, the end to end layer doesn't assume anything about the format or the layout of the data that's received from the application layer, OK, or from the application.

So what this layering abstraction buys us is that it allows these things to sort of coexist without any understanding of what the other layers do.

So in particular, it means that we can, for example, change something about the format of the data that the network layer sends.

And we continue to use the same link layer.

OK, so I can send it out over an Ethernet that's not data that's been sort of packaged up by IP, right?

It doesn't necessarily have to be an IP packet to send it out over Ethernet.

And so, this sort of separation between the layers is going to be really critical for allowing us to sort of maintain and develop new networking code over time.

And it also means that the people who provide the sort of companies that build and sell software and hardware that works at these different layers don't really have to assume very much about what the other layers are going to provide, right?

So if I'm making an Ethernet card, I don't have to understand anything about exactly what, I hopefully won't have to understand anything about exactly what's sort of going on up at the higher levels of the network stack.

You have to be a little bit careful, though, because of course the individual layers do have some protocol that

there is some sort of API that they're using to interface with each other.

So, API is an application programming interface.

You have some set of routines that they call on each other.

So for example, I showed in this example up here that the networking layer is calling this link send message on the link layer below it.

So the link layer has to provide this link send interface.

And similarly, there's a comparable interface when the link layer receives a message that uses to send up to the network layer.

OK, but basically what this means is that we can develop the software that runs at the different layers in isolation without having to worry too much about what's going on at the layers above or below us.

OK, so -- Let's return back to our set of questions now and talk a little bit about how these questions map onto our three layers.

OK, so our first question is, so we said question one is, we've sort of already addressed that.

But question two is, well, how do we transmit bits on a link?

OK, so clearly that's going to be handled by layer two, OK, or by the link layer, sorry.

And, so the link layer is the thing that's going to be in charge of actually pushing the bits onto the link.

And none of the other layers need to know about this.

Question three, OK, how do we forward packets via switches?

Well, it seems that it's pretty clearly what's happening at the network layer.

OK, so the network layer is the thing that's deciding sort of which packet, what the next link that we should use is within a switch.

OK, and now questions four and five are these questions about this kind of our best network properties.

How do we achieve reliable communication?

How we manage congestion?

These are things that were going to worry about at the end to end layer.

OK, so just to sort of illustrate a little bit more about sort of how the commercial, and how these things are separated in sort of a commercial world, I thought I'd just show you the slide that sort of illustrates that there are lots of different vendors, both hardware and software, that run at each one of these different layers.

So, for example, at the end to end layer, first there are clearly a bunch of applications that run up there.

And each of those applications perhaps has some different sort of set of requirements about how data is delivered.

And those applications, typically the sort of things like the TCP protocol are provided by, say, the operating system vendor.

So Microsoft Windows provides an implementation of TCP; similarly Mac OS and Linux also provide this.

At the network layer now we have sort of this huge variety of people who are building these network switches.

And these network switches are the things that have sort of coded into them the rules for how you should forward messages around on the Internet.

And so these are these companies like Cisco and Alcatel, in sort of these big companies that hear mentioned in the news all the time.

And then finally at the sort of lowest layer, there are these sorts of link layer things.

And again, there are a number of link layer technologies like WiFi and Ethernet.

And those link layer technologies are different than the technologies that are used to actually decide which hop we should next use in order to transmit data around in the Internet.

OK, so if there's a big diversity of applications.

And part of this diversity of applications is enabled by this separation of the layers because Microsoft Windows doesn't really have to know anything about how the network layer works.

It simply passes sort of messages on for the network to transmit.

OK, so and the point is that the vendors are generally different at each of these different layers.

OK, so given this sort of high-level overview of layering, what we're now going to do throughout the next few lectures is to pay some attention to how each of the different layers works.

So we're going to start off talking about this class about the link layer.

And then we'll move on to the network and ends to end layers in later lectures.

OK, so we've sort of seen some of the things that touched on some of the things the link layer needs to provide at a high level.

But let's make a list and talk about what these things are.

So the first thing clearly that the link layer does is manage the sort of transmission of bits along this physical wire.

OK, so there's going to be some digital to analog to digital conversion that happens in sort of as a part of using any one of these links, OK?

And this is going to be one of the main functions of the link layer is it's going to decide how this sort of digital to analog to digital conversion is done, OK?

Another thing to link layer does is framing.

And so, framing, well you remember we talked about at the link layer, we sometimes call for messages that are transmitted around, we call these things frames.

Framing is simply separating the frames, is deciding how we should separate the frames that are on the wire.

So it says, how does the software that's running at the link layer decide that one frame has ended and another frame has begun?

That's what framing is about.

And we'll talk about these two issues briefly today.

The other kinds of things that happen at the link layer we're not going to talk about as much about today.

One of them is channel access.

And so this is how somebody who wants to send a message actually is able to physically use the wire or the air that it's transmitting out of without interfering or something on top of something else who's transmitting at the same time.

So you guys have read the Ethernet paper, and you saw one way in which that's done and Ethernet, which is basically by listening for a carrier, right, which is called carrier sense, and only when the channel is not used, there's not a carrier on the wire, does somebody try and send.

And then you use this notion of collision detection and Ethernet in order to actually sort of detect whether or not you are able to successfully transmit your message.

So these are the kinds of things you worry about in channel access.

Last time with the phone network, we talked about time division multiplexing as another way in which you can sort of share access to a physical wire.

You can carve it up into a bunch of little units of time and assign each sender one unit of time.

OK, so now the last link layer issue which sometimes is done in the link layer is error detection and correction.

And I don't want to talk at all about really how error detection or correction works except to say that some link layer is included in some link layers don't include it.

The idea here is suppose we are transmitting a message out over a wire.

Of course, there's some probability that that message will become corrupted or garbled it is being transmitted, either because it interferes with somebody else who is transmitting at the same time, or as the message propagates, it decays somewhat and we can't decode it anymore.

So sometimes link layers include a facility for doing this error detection and correction.

And error detection and correction is one of these things that can sometimes be included at the link layer, and is very often included at the end to end layer.

And so, as you read, the reason I mention this as being a part of the link layer is that as you read the end arguments paper for recitation next time, you should sort of think about how the end-to-end argument relates to whether error detection and correction should be within the link layer or should be within the end to end layer.

OK, so let's talk about these sort of two issues that I said we'll address briefly here.

So the first issue I want to talk about is how the conversion from digital to analog to conversion works.

So -- So the way to think about, suppose that we have some sender which has some sequence of bits, say, one, zero, one, zero that they want to send out over the radio channel.

So if you think about what this is going to look like in terms of an analog signal, they are of course are many different ways you could represent it.

But a simple way to represent it might be to say that this is a digital line, and we'll make the line high when we are transmitting a one, and we'll make the line low when we're transmitting a zero, OK?

So we would send this message out as high followed by low followed by high followed by low.

So this would be one, zero, one, zero.

OK, you might ask the question, OK, how do we decide when to sort of push the next bit out on to the wire?

Well, typically the way we do it is we have some clock signal that tells us when the next bit should be, we should start sending the next bit on the wire.

So we might have some protocol that says something like, every time there is a rising edge in the clock signal, we'll start transmitting the next bit.

So, what does that mean?

So a rising edge -- OK, so suppose this is our clock and this is our data.

And what I've just shown here is that every time a clock signal goes high, we start sending in new bit.

OK, so every time we go from low to high in the clock signal, we start putting a new bit on the wire.

So, now it's going to happen is oftentimes the way the network connections are connected is what's a so-called serial connection.

So we have one wire that's transmitting the data.

And we don't have a separate wire that includes the clock signal.

We just sort of transmit the data out over the wire.

So if you were to look at this data as it comes down the serial connection, at the receiver, what it would look like is something that was sort of, it's not going to look exactly like these nice square pulse was that we have here.

It's going to have decayed somewhat.

So it might look something like is each of these nice square waves might have sort of become a somewhat

decayed version of their former selves, OK?

And now the question we have is, OK, how are we going to decode this, right?

So we don't have access to the original clock signal that was used.

But we may know what frequency, would rate this data was encoded at.

So we may be able to generate a comparable clock signal.

So suppose we generate a clock signal that's about the same frequency, and then try and use that to decode the message, say, by looking again at the rising edges.

If we're not careful, we're going to get something that's wrong.

So in this case, I transmitted one, zero, one, zero, but now I'm decoding something that's shifted from that somewhat.

So I'm decoding zero, one, zero, one.

So, I've sort of become offset because the two clock signals aren't actually identical, right?

Even though they're at the same frequency, they are not specifically lined up with each other in time.

So we say that those signals are out of phase.

So one signal is essentially a time shifted version of the other signal.

OK, and this time shifting leads to these kinds of problems where we don't properly decode the message because we're not sampling the channel at the right point in time.

We're not looking at the channel to see whether it's high or low.

OK, so how are we going to fix this?

It turns out that there's sort of a simple and elegant way that's often used to fix this.

And it's what's called a phase lock loop or a PLL.

So a phase lock loop, a very simple way that you can implement a phase lock loop is as follows.

So the idea here is that what we want to do is we want to figure out, we want and make it so that the receiver has

essentially a lined up version of the transmitters clock.

So we need to figure out how much we need to shift the receiver's clock in order to make it line up with the transmitter's clock.

And once we do that, then hopefully we will properly decode the message instead of being shifted when we decode the message.

So the idea with a phase lock loop is kind of as follows.

Suppose we have our signal like this.

A simple way to implement a phase lock loop is to take this signal and to sample it not once per clock, but some multiple number of times per clock period, OK?

We call this oversampling.

We might, say for example, do eight times oversampling on this.

OK, so what that means is if we were perfectly lined up in time on this oversampled signal, and we were encoding a one as a high, and a zero as a low, then what we would see if we were perfectly lined up is alternating sequences of eight zeroes and eight ones, right?

So, if instead we start decoding this and we see something that's not quite that, suppose we see three ones followed by eight zeroes followed by five ones.

OK, so we decode two bits worth of information off the wire, and we see that it's sort of shifted in this way.

OK, that suggests that I need to shift the signal to the left some amount.

OK, and so this is exactly what the phase lock loop does is it observed the signal as it's coming in, and it computes an amount that we need to shift the signal in one direction or the other.

OK, so if you wanted to make a sort of schematic for how a phase lock loop works, we have our sender and the receiver.

The idea is simply as follows.

You have some data at the sender, and a clock at the sender.

Those go into some encoder box.

They are transmitted out over a line to the receiver, which has a decoder box.

OK, but the decoder box needs a clock signal that's been lined up with the sender's clock signal, OK?

So we're going to do is use our phase lock loop to do that.

So the phase lock loop is going to take in the incoming signal as well as the unaligned clock from the local machine.

And it's going to input into the decoder an aligned clock signal.

OK, so basically what's going to happen is this PLL is going to be able to reconstruct the clock signal from the sender.

Of course, it takes a little bit of time for the PLL to reconstruct the clock signal.

So usually what we do is every message that we transmit over the link, we have some sort of set of synchronization bits at the beginning of it that we use in order to allow the phase lock loop to lock in to the phase of the signals being transmitted.

And that preamble doesn't carry any data bytes.

It's simply sort of overhead that's on every packet to guarantee that the sender and receiver's clock synchronize with each other.

So there is one last little detail associated with phase lock loops that you guys may have noticed.

So the issue is that the signal that I've shown being transmitted here is a one, zero, one, zero, one, zero signal, right?

But if you think about the way that this phase lock loop works, what the phase lock loop does is it looks for these transition points in the signal, right?

So it looks for points where the signal changes from a one to a zero, right?

So suppose that instead of transmitting one, zero, I transmitted zero, zero, zero, zero, zero, zero, zero, right?

Then the problem I'm going to have is I'm just going to have a very long sequence of zeroes even if I do this eight times over sampling.

I'm still just going to have a long sequence of zeroes, and I'm not going to know how much I need to shift the

signal, right?

I'm going to have no way of computing how much I need to shift the signal.

So it turns out there's a very simple and kind of elegant solution to this called Manchester encoding.

And the idea with Manchester encoding is as follows.

It says will transmit a zero as a transition from a low to high, and we'll transmit a one as a transition from a high to a low.

OK, so now if I transmit a signal like one, zero, one, if I transmit the signal: zero, zero, zero, it simply looks like this.

So, zero is a transition from a low to high followed by another transition from a low to a high.

OK, so now this is zero, zero, zero, zero, right?

And if I transmit the signal, zero, one, zero, this is a transition from a low to a high followed by a transition from a high to a low followed by a transition from a low to a high, OK?

So now we have a way that we can guarantee that every bit as it least one transition in it.

And that's going to allow our phase lock loop to look into the face of the signal that's being transmitted.

Of course, the cost of this is that we've now doubled the number, sort of, we've doubled, every bit as a transition in it, right?

So we sort of have the number of bits that we can send over the channel because instead of a one being simply a low, or a one simply being a high and a zero being a low, now everything is both a high and a low.

So we have the amount of data that we can send on the channel.

But we've sort of gotten this wind that now we can have the phase lock loop actually work.

And so turns out Manchester encoding is actually commonly used.

There are other encoding schemes that you can use that are sort of less wasteful of channel bandwidth, but operate on the same principle, trying to introduce extra transitions when possible.

So this basically wraps up our discussion of the link layer.

What we're going to talk about next time, are going to start talking about how the network layer works.

And we're going to basically talk about how Internet routing actually functions, and how these routers actually decide which link they should use to transmit the next message around in the network.

So, that's it for this time, and we'll see you on Wednesday.