So, before we begin, I just want to make sure you guys all remember about the quiz on Friday.

So the quiz is going to be in Walker at 2 pm.

Everybody goes to Walker.

The quiz is open book.

You can bring your own notes.

You can bring a class notes, bring the readings from class; you can bring your calculator.

But please don't bring any computers, phones, PDAs, things like that.

So make sure you have all your stuff printed out before you show up if you've been taking notes on a laptop.

OK, so what we're going to do today is just quickly, well, we're going to do an introduction to networking.

But quickly I just want to finish the topic of caching that we began, that sort of introduced at the very end of the lecture last time.

So, if you guys remember, we were talking about our pipeline WebServer system that consisted of these three stages: a networking stage connected up to our HTML stage connected up to our disk stage.

OK, and we did a little performance analysis of the system.

We looked at what the cost of each of these stages is, what sort of throughput of each one of these stages is.

We said that as long as we split this networking module up into ten submodules, we can get the throughput of this thing to be about 100 requests per second.

We said the latency of this was 1 ms.

So that means it can process a thousand requests per second.

We said the latency of the disk was 10 ms.

So that means it can also process 100 requests per second.

And we said, remember, that when we are looking at a pipeline, the sort of throughput of the entire pipeline is

going to be bottlenecked by the slowest stage in the system.

And so if we look at this, we see that these two stages are both running at 100 requests per second.

But we have a very simple way of making this first network stage be able to process more requests per second, right, because we simply can replicate the number of threads that are sending data out over the network.

So, for example, if we went from ten nodes here to 100 nodes, we could increase the throughput of this stage to 1,000 requests per second, which means that now the bottleneck stage that we have is simply the disk stage.

So the question is, is there something we can do to increase the throughput of the disk stage?

If you think about it, at first it may seem like, well, there's probably no way that we can do anything because the disk takes 10 ms, you know, every page takes 10 ms to read in.

So what are we going to do about that?

And there's a very sort of standard answer to that that you guys have all seen before, and that answer is caching.

OK so the simple idea is that we're going to take this IO stage, or this disk stage, with this disk that runs at 10 ms.

What I've shown here is a very simple piece of pseudo code that might correspond to what this sort of read or the get page ID function for this IO stage does.

It simply calls some read function that reads page ID off the disk and then returns the page.

If we add a cache to this system, suppose we have an in memory cache that can retrieve a page in 0.1 ms.

In that case, the way that we can use that cache is just before every time we go to the disk, we can check and see if the page that we're trying to load is in the cache.

And then only if it's not in the cache do we need to actually go to the disks.

And we get a miss on the cache.

We go check the disk to see if the page is available.

So we can extend the code in a very simple way to take advantage of this.

We simply say, we look up in the cache first, and then if the page is null or empty or whatever, we can't find it in the cache, then we go ahead and look it up on the disk.

And then we add that result to the cache.

So there's a couple of little details that we need to work through.

But let's first look at what the performance of this system is going to be, or how this is going to impact the performance of what we've been doing.

So if we come over here, if we think about what the cost of accessing a page on this system would be, well, we're always going to have to check the cache, right?

That's the first thing we do.

So I'm going to write C for cost of accessing the cache, where cost is in this case going to be expressed in something like the number of milliseconds to do a lookup in the cache, and then plus the cost of looking it up on the disk.

But we only have to look up on the disk some of the time, right, so there is some probability of getting a miss times the cost of that miss.

OK, so in this case, we said the cost of going to the cache was 0.1 ms.

The cost of going to the disk is 10 ms.

OK, and now what we're left is to figure out what the probability of this miss is.

So if you think about a system for a little bit, you might say, well, OK, what does the system look like?

What's the probability that we are actually going to get a hit or miss?

And if what the applications are doing is simply generating random requests for random pages, which might be one model of how the application behaves, then it doesn't seem like cache is going to buy us very much, right, because the cache is going to be a small fraction of the total size of the disk.

It might be in RAM, so we might have 100 MB of cache, and we might have 10 GB of disk, right?

So that means there is a factor of 100 difference between these two things.

So if the page requests are random, the probability that something is going to be in the cache is going to be very low.

It's going to be only, say, 1%. But it turns out that almost all applications have an interesting property which is

commonly referred to as locality of reference.

What this means is that if you look at which pages a program is likely to access over time, be these pages of data or parts of a program, typically a page that has been accessed recently is very likely to be accessed again.

So a simple example might be that if you look at the files that are being used in a computer system, usually there is a small set of files that a user is working with at any given point in time.

You're running certain programs; you're editing certain documents.

And there's a huge array of other programs and files and documents that are on the system that you aren't accessing.

And when those active files are being accessed, there's a much higher probability of getting a hit of looking at those active files then there is of going to any one of the other files.

So even when the difference between these things is, say, a factor of 100, in the case of a Web server, it might be very likely that we would have 90% maybe of the pages that have been accessed are already in the cache.

So in that case, sorry, this probability should be 0.9, 90%. OK, so suppose that the probability, sorry, the probability of a hit is 90%. The probability of a miss is then 10%. OK, so if you look at now what this sort of formula evaluates to, we see it's 0.1 plus ten times 0.1. So, it's 1.1 milliseconds on average, assuming that we get a 90% hit rate on our cache.

OK, so if you now come back to this diagram, that means that the throughput of the box is one over 1.1, remember, because throughput is just one over latency when we have sort of just this one module, which means that now we can process something like, so this is something like, sorry, this should be one over latency.

But this is in milliseconds.

So this is 0.1, one, sorry, the number of seconds is 0.0011, right?

So, one over 0.0011 is about 900. OK, so we can get about 900 requests per second that we can process instead of just 100. This is approximately equal.

OK, so what we've managed to do is increase the throughput at this stage by about a factor of nine.

I mean, now you can see that sort of all three of these stages are close to about 1,000 requests a second.

We have increased the performance of the system pretty dramatically by introducing the cache.

And you have to remember that this cache is only going to be a good idea when we are sure that we have this locality of reference property.

OK, so if the Web server is going to a random page, if the Web server is sort of asked to fetch a random page on every request, the cache is not going to be a good choice.

But most of the time Web servers to have this locality property.

So the last little detail that we need to talk about when we talk about caches, is the sort of question about how we deal with page eviction.

So in this diagram here, when we call add to the cache, if the cache is already full of results, we have to pick something to replace, right?

So we need what's called a page removal or page replacement policy.

OK, so you guys presumably have seen different page removal or page replacement properties in 6.004 before.

I'll just talk about two very quickly: a FIFO policy and a LRU policy.

OK, so what FIFO means is First In First Out.

So it says the thing to throw out is the first thing that we loaded into the cache.

OK, so if I access a set of pages, suppose I have a three element cache and I access a set of pages, one, two, three, two, one, four, what I'm going to do is I'm going to load this.

I'm going to access these three pages.

Page 1 will be the first one that's in my cache.

So when I try and load page 4, I'm going to evict page 1. OK, that's what's going to happen in a FIFO system because one was the first one that was loaded.

But a least recently used approach says instead of evicting the first thing that was put into the cache, we want to evict the last thing that was read from the cache.

So in this case, if I do one, two, three, two, one, and then I get four, the last thing that was read was three, and that's what I'll evict from the cache.

So, if you think about the intuition behind these two policies, you sort of realized that this FIFO approach has a

problem, which is that it's sort of contrary to this notion of locality of reference, right, because it doesn't capture anything about how frequently a data item is accessed or when it was last accessed.

It throws the sort of first access thing out of the cache even if that first access thing is read all the time.

LRU sort of captures the intuition which we want, which is that something that's been accessed recently is likely to be accessed again.

And something that hasn't been accessed very recently is less likely to be accessed again.

So it sort of captures our intuitive idea that we want locality of reference.

So the text talks much more carefully about these different page removal policies.

But anytime you see anybody talk about a cache, you should sort of immediately think to ask the question, well, what's the page removal policy that's being used?

OK, so that basically does it for our discussion of caching and of performance that we started last time.

And what we're going to do now is move on to the topic of networking, OK?

So -- So a computer network, you guys should all be familiar of what the idea of a computer network is.

It's some sort of a connection that connects multiple computers together.

So why are we interested in studying a computer network?

What is it that's relevant about a computer network for the purposes of this class?

And there's really sort of two primary reasons.

The first one is that computer networks are commonly used as components of systems.

OK, so anytime we are building a big computer system, it's very likely it's going to involve a computer network.

So it's going to be important for us to understand what the properties of computer networks are, and how computer networks affect the design of our system if you want to use them effectively in our systems.

And computer networks have a number of uses in the computer systems.

They do things like allow us to overcome geographic limits.

So what you guys are all familiar with the ability of the Internet to allow you to check the score of some sporting game that's happening on the other side of the country, or to be able to send an e-mail to your parents back home.

So obviously, being able to sort of transmit data over long distances is clearly a good thing for many kinds of systems.

They also allow us to access remote data.

And this is related to the other thing, to the other one.

But if you have data that's not on your computer that you want to get a hold of, you may want to contact your bank and ask them for your bank balance.

And then finally they can be used to physically separate a client and a server.

So we talked about the last few lectures, we spent a while talking about the fact that we can use things like threads and address spaces in order to provide a sort of enforced modularity between modules running on the same computer.

But there are lots of situations in which we want to actually physically separate the client and the server, right?

Your bank doesn't really want you to be running your copy of Quicken on their server, right, because this notion of enforced modularity we talked about isn't a perfect separation between the client and the server, whereas putting these things on really separate machines that are only connected by this network that is controlled by this piece of hardware that's this network card that sort of talks with the network, that's a good reason may be to separate these things from each other.

And then finally, the second major reason why we want to study networks is simply that they themselves are an interesting computer system, OK?

So we talked in the first lecture about some of the interesting sort of properties that big computer systems have.

Well, there's not many computer systems that are bigger than the Internet, right?

It's a perfect example of the giant complex interesting system with all sorts of complicated behaviors.

And we'll talk about some of those complicated behaviors today as we overview networks.

So the goal of our discussion of networking in some sense is going to be to develop a set of tools that allow us to

have universal communication between a number of different clients.

OK.

So what do I mean by that?

Suppose I have some collection of machines, which I'll just draw as little boxes that are scattered around.

And they are connected together.

What we want to do is to connect these guys together, OK?

And the network system that we are going to design is going to be the interconnect that allows these got to talk to each other.

And what we want this network to be able to provide at a very high level is the ability to do any-to-any communication.

OK, so if this is A and this is D, we want to A to be able to send a message to D or anybody else who's in this network.

And that should be true for any of the pairs that we can find.

And so, in some sense, what we're going to do over the course of the next couple of weeks is sort of see how we design this cloud that sits in between all these different computers.

If you like, you can sort of think of this cloud.

Often times, the Internet is represented as a cloud.

It's just some black box that you sort of send messages into with an address.

And on the other side, the message pops out at the other end.

So, we're going to dive into this cloud and see what happens inside of it.

So, in order to start to understand some of the challenges of building this cloud, we are going to start looking top down.

We're going to look at the biggest issues that we have to deal with, and then we're going to break up those issues into some smaller pieces.

So - - OK, so what are the issues that sort is immediately pop out when you first start thinking about the properties of a network?

Well, so there's some technological issues which have to do with things like the physical medium over which you're transmitting messages, or the properties of the links over which you're transmitting them, the rates at which you can send data, the lost rate, the percentage of messages that get lost by the link as it's transmitted.

So those are the kinds of things we mean by technological concerns.

And so one kind of technological concern is this, is sort of the different kinds of technology that we might use for transmission.

And we saw this notion of D technology over DT before.

It's just this idea that technology, just like the technology in your computer system has been evolving dramatically over time.

The technology of networks has been evolving dramatically.

And we're going to sort of study some of the range of different technologies that we have to deal with.

There also are a set of limits, technological, physical, fundamental limits that are associated with networks.

So, these are things like the speed of light, OK?

We simply can't send messages faster than the speed of light, right?

These are messages that are being transmitted [down wire?].

And they propagate at some speed.

And that's going to be fundamentally a bottleneck in the design of a computer system.

If I have to send a message from here to California, that message isn't going to get there any sooner than the amount of time it takes for light to travel across the country.

And the amount of time that it takes to transmit a message across the country is nontrivial in computer time.

It might be a couple hundred milliseconds.

And that's a long time as we saw before it when we have a processor that connects one billion instructions a

second.

OK, so the other interesting issue that's going to come out about networks is that networks are a shared infrastructure, OK?

What that means is that there are multiple users who are simultaneously using a network.

You guys, of course, all know this from using the Internet.

And all the networks that we study in this class are basically a shared infrastructure.

And we'll talk in a minute about why the networks are fundamentally almost always going to be sort of shared.

They're going to have to transmit data for multiple different users at the same time.

And that's going to present a number of challenges about how we ensure that the network is fair, that everybody gets to send their data when they want to send data, how we sort of allow multiple people to access the same physical wire at the same point in time.

OK, so what I want to do is I'll talk about these two things now in order.

I'm going to start off by talking about technology.

So -- So the first interesting technological problem, which I've already mentioned is that these networks are extremely diverse.

They are heterogeneous.

OK, networking technology has evolved a great deal in the past 20 years or 30 years since networks first started being designed.

And that means there is a huge sort of range of devices that we might have to transmit data over.

So let me show you what I mean.

This is just a graph showing the rate of transmission, the sort of number of bits per second that you can send over different networking protocols.

And the thing to take away from this first is to note that on the Y axis here, this is an exponential scale.

So the technology at the very bottom, things like early telephone modems, could maybe send data at 10 kb per

second, whereas these sort of high-end router class devices that are running the Internet or sort of very fast desktop Ethernet kinds of connections that are coming out today can send more like, say, in this case we have 10 Tb a second, right, so one times ten to the tenth.

So we are talking about a factor of ten to the seventh or ten to the eighth difference in performance in terms of the number of bits that these different kinds of devices can actually transmit.

So, that's pretty dramatic.

And that's going to make it difficult to design computer systems.

OK, and these kinds of variations occur not only at the sort of number of bits that we can send per second, but also in terms of the propagation delay of the links, OK?

So if I have a local area network that's connecting two computers in my office together, I may be able to transmit a message from one computer to the other in an order of microseconds, OK?

But as we said before, to send a message all the way across the country might take 200 ms.

To send a message across the Pacific Ocean by way of the satellite around the United States might take a second.

To send a message to Mars might take tens of seconds or a minute.

OK, so there's this huge range in terms of transmission times that some of these different links have.

So, fundamentally these two things, bit rate and propagation delay are going to tell us how long we would expect it to take for a message to travel from one place to the other.

So for example, the amount of time, so we can think of a communication link, if you like, as being a piece of pipe, which has some width that is the bit rate.

OK, so the width of the pipe is the number of bits that I can cram down this thing.

So, a wider pipe I can shove more data down it.

But no matter how much data I can shove down it per second, there still is some link to this pipe, right?

And that's the sort of separation between the sender and receiver.

And that's bottlenecked, as we said before, by the speed of light.

And that's going to affect how long it takes for a message to get from, say, one endpoint to the other.

So if you think about the total time to transmit a message from one guy to the other, it's going to be the amount of time it takes to pack the bits into the beginning of the pipe, right, and then the amount of time that it takes for all those bits to come out the other end.

So it's going to be some combination of the propagation delay plus the transmission delay.

The propagation delay is just going to be equal to length divided by, say, for example, the speed of light.

And we're going to need to add to that the transmission delay, which is simply going to be the number of bits we have to send divided by the bit rate, the number of bits per second.

OK, and so this is the time to send along one link.

OK, so it's worth remembering that this time to send equation, this is only to send along a single link.

If we have to send more data, if there are multiple people, for example, who are waiting to get access to a link, that may increase the delay that it takes to send a message.

Or if we have to relay a message over multiple hops and there are some intermediate node in between each of those hops, then that sort of processing at each intermediate node may take some additional time as well.

OK, so if we look at a graph of some of the properties of networks, it's interesting to sort of look at the rate of change of these different technologies.

So this is just showing what Moore's Law is.

So we saw this before.

Processor speed doubles every 18 months according to Moore's Law.

If you look at the size of the Internet, so the number of things that we have to connect together, well, that's been doubling every 13 months.

So this has this sort of exponential behavior that Moore's Law has as well.

So we are talking about sort of the number of things that we have to connect together is just growing at this massive rate.

And that's a real challenge for designing the kinds of protocols that we are going to need to use in these networks.

The other thing that we see that's been going up at the same rate or even faster is the amount of traffic that's being sent over the Internet.

So more and more people are sending data.

And again, this affects the sort of, this is the challenge for designing protocols that are able to scale up and up and up to these greater and greater rates.

The largest link capacity in the past few years has actually started to go up as fast as double once every seven months.

So this is the size of the largest pipe that's on the Internet, say, connecting lots of these big service providers on the Internet together.

And this is just been getting larger and larger and larger as well.

And so that's, again, means in order to exploit all this additional link capacity, we need faster computers.

And the rate of change of technology is just extremely high.

Interestingly enough, of course, as we said before, the speed of light never changes.

So this is a fundamental constraint that we're always going to have to deal with when we are building these things.

That is also the case that if you look at, there's another number that you can look at.

And in the context of the Internet, this number is, this is bits per second per dollar.

OK, so what does that mean?

This is the amount of money that it costs to, for example, send one bit per second over the Internet.

And this number just has not been scaling up very fast.

If you look at this, it's a factor of ten cheaper today than it was in 1980 to send the same amount of data per second out over the Internet.

So, why is that?

That seems strange, right?

I mean, if you look at the fundamental technology, the technology itself has gotten much faster.

If you look at the Ethernet network that you might have in your home or your office, this thing has gotten, in 1980 you might have had one megabit a second that you could transmit over this Ethernet.

You can now, for $100 go to Best Buy and buy something that can transmit a gigabit a second over an Ethernet.

OK, so we are talking about a huge scale up in technology in terms of the performance of the networks.

The issue here is that there is a human cost that's associated with sending data out over the Internet.

And if I want to send data from here to California, there's a physical wire that that data has to transfer over, right?

And somebody owns that wire.

And it cost somebody a lot of money to cut the holes in the ground where that wire runs and to knock holes through buildings and to get rights from all the cities that that wire runs through to dig holes in their roads, right?

So, there is some very large human cost associated with setting up these sort of very wide area internets.

And you see the same kind of thing not just in wire but in wireless technology with wireless companies paying huge amounts of money for access to the rights to use a particular part of the wireless spectrum.

So this is these bits per second per dollar is growing slowly, OK, and the reason for that is human costs.

OK, and what this argues for, what this suggests that we are going to need to do when we are building these internets or building these large networks is to share these network links.

So, if every time we added a new host to the network, we had to run a wire that connected it to every other host in the network.

Obviously, that would be hugely expensive, so we want to avoid that.

And so, we are going to end up because of these costs of physically running the wires having to share network connections.

And that brings us to sort of the next topic that I want to talk about, which is this topic of sharing.

OK, so just that everybody is clear, this notion of having pair-wise links, a connection from every host every other host is just not practical.

So suppose I have them set of nodes like this, if I want to connect all of them together by their own set of independent wires, I don't know if I got all of them there, but then the cost is going to be, if I have N hosts, the cost is going to be N squared wires.

OK, and that is just too expensive, right?

If you have a million hosts, you're going to have a trillion wires.

So we don't want to be building anything that has a trillion components in it.

It's going to be really, really expensive to deploy.

So what we end up doing is multiplexing.

OK, so what do I mean by that?

I mean this notion of sharing links.

So let's look at a very simplified kind of an architecture for something like the Internet.

So suppose I have two hosts, two nodes, two computers, A and B that I want to be able to communicate over a network to some other pair of nodes, C and D.

OK, well in a multiplex network, what we're going to do is we're going to have a set of these boxes that are going to connect these nodes together.

So I just sort of shown one possible network topology.

But with these boxes are, they are sometimes called switches.

And they are the connection points that are scattered in various locations around the Internet that transfer data from one location to the other.

So in your home, you may well have what's sometimes called a wireless router.

That's the thing that transfers data from the air onto a wire network.

You may have a box from your cable company that transfers data from your home out over the cable.

And then the cable company itself has a box that transfers data to some other service provider, which may be transfers data across the country, which transfers data into the home of whoever you're trying to send, or the

business of whoever you're trying to send a message to.

So you have this set of what we are calling switches here that connect all these various hosts together.

And these hosts clearly, there are connections here that are sharing data.

So if I look at this link here, if both A and B are trying to send data to C, clearly there's going to be some data for both A and B traveling on this wire.

And so, what we end up with is, sorry, so there is a set of issues that are brought up by the fact that we are sharing this data, and that we have this kind of switched infrastructure which we are running data around in.

So the first issue that we need to talk about that obviously comes up is this notion of routing.

OK, so when you see a graph like this and you ask the question, well, OK, how does A get its data to C?

We have to pick some route which A is going to take along this path, through this network in order to get to C.

And, there are a couple of different alternatives, right?

So, any time you're sending a message, there has to be something that plans out how that route is going to be transmitted.

And commonly what happens is that each one of these intermediate points along the network has some table which says how to get to every other location within the network.

And we'll talk about how those tables work, how these tables that talk about where to forward your messages to work.

But when a node has multiple sort of possible locations where it can send data out to, typically that node is called a router.

OK, so if you hear the term router, just think of it as one of these switch things that gets to make a decision about where some kind of traffic is going to be sent to.

OK, so now what we need to do in order to think a little bit more about how these switches work is to resolve the issue of how we multiplex traffic over one of these links.

So we said there may be traffic for both A and B running on this link going into D.

Well, so how are we going to interleave the traffic from A and B in order to make this work?

And there are two techniques.

We call this technique for interleaving data, we call it switching or multiplexing.

And we're going to talk about two techniques for switching.

We're going to talk about circuit switching.

And we'll talk about packet switching.

So circuit switching is pretty easy to understand at an intuitive level.

The idea with the circuit switch is that we want to set up some reserved channel that is the communication channel between two endpoints, say for example, A and C.

So, a simple way if you want to think about circuit switching is to think about the way that phone networks used to work.

So you may have seen these old movies where the operator would plug wires into a switchboard.

What an operator is doing there is establishing a physical circuit, a physical wire that connects two endpoints together.

OK, so that's circuit switching.

Each operator has a bunch of lines coming in and a bunch of lines coming out.

Somebody connects to the operator and says I want to talk to line A.

And the operator establishes the physical connection.

So, of course we don't use that technology for doing circuit switching anymore, but this notion of circuit switching is still used in the telephone system.

And the way that a common digital telephone system might work is as follows.

So the idea is that suppose I have some set of nodes A, B, and C, all of which want to send their data out over some line.

So they are connecting up to some switch.

And they want us their data out over this switch, out over this wire that is on the other side of this switch.

OK, so what this kind of switching that we are going to talk about does, which is commonly called TDM for Time Division Multiplexing -- is to give each one of these nodes, A, B, and C, a little unit of time in which only its traffic is being transmitted on this wire.

So if you think of this wire as having one additional piece of information put on at each unit of time, and this is the wire stretching out away from the host so that this is sort of time is going this way as the signal propagates down the wire.

And if you were to look at all the different messages that are on the wire, what you see is sort of a repeating pattern of time intervals.

And each of these time intervals would be carved up into a number of slots.

OK, and each slot is allocated to exactly one of these nodes.

And each node gets the same slot and each one of these larger intervals.

OK, so you'd see a pattern like A's traffic, B's traffic, C's traffic, and so on.

And then in the next interval we would have A's traffic, B's traffic, and C's traffic again.

So there would be this repeating pattern where each node is associated to the same slot in every one of these intervals, OK?

And we call the data that, say, for example, A puts into a slot a frame.

The frame is sort of the set of data that is allocated to a node.

And, so this is basically the way that circuit switching works.

And it's almost exactly the way that the phone network works now.

So a phone network carves up the wire into a bunch of these little slots where each conversation is allocated one of these sort of frames in each interval.

So in a typical phone network, a frame might be eight bits.

And there might be these intervals.

There might be 8,000 of them per second.

OK, so we have 8,000 intervals per second.

OK, and so what that means is that if each frame is eight bits and there is 8,000 intervals, each sender can send 64,000 bits per second.

And, it turns out that 64,000 bits per second is about enough to be able to encode voice at a sort of acceptable level for people.

So we can talk about the capacity of this channel by basically looking at the number of frames that are in one interval.

So if we say there are N max frames in one interval, then that specifies the number of conversations that we can simultaneously maintain on this wire.

OK, so the behavior of one of these circuit-switched systems is as follows: everybody who's having a conversation has bandwidth allocated to them, and has a frame allocated to them in every interval.

So, up to N max conversations can be held simultaneously and there's no problem.

But as soon as we get to the N max plus first guy tries to initiate a call, the network will simply refuse to accept that connection because there's no slot available for this guy to do his communication.

And the last little detail about how these circuit switches work is how do we actually establish a connection in the first place?

And we need some protocol for actually resorting to a slot for a particular connection between a pair of endpoints.

And so typically what you will do is reserve some frame or set of frames to carry information that's used to configure the connection between the endpoints.

So, A can request that an interval be assigned to its connection between C, and then it will be by communicating during a special frame.

OK, so that's circuit switching.

But, circuit switching has a problem.

And we're going to need to introduce this notion of packet switching to deal with it.

OK, so the reason that circuit switching has a problem is as follows: because although circuit switching works great for voice where every conversation is essentially a continuous stream of audio that can always be encoded at about 64 kb per second, Internet traffic just doesn't work like that, right?

People who are communicating on the Internet can have very variable demands for the amount of traffic that they want to send.

And even taken as an aggregate, taking all the people who are, say, for example at the community of MIT over time, there still is a lot of variation in the network demand.

So this is just a plot over the course of the day with samples taken every five minutes of the amount of traffic that was being sent out of one of MIT's labs.

OK, and so you don't need to worry too much about what these different lines show.

But the point is that there are these periodic spikes.

And these spikes represent spikes of traffic demand a different machines on the network.

So you can see that at any given point in time, there can be anywhere from, at its lowest, these numbers are about 40 Mb per second to numbers as high as 100 Mb a second.

So there's a lot of variation in the demand at MIT's network.

Here's another way of looking at the same traffic.

This is an interesting graph.

This is showing the amount of traffic that different protocols use.

This is a real graph by the way.

This is showing the amount of traffic that is transmitted over MIT's network for different types of protocols.

And what you see here is that it's really pretty amazing.

You see that Kazaa, this file trading network, is using up a vast majority of the traffic that we send out of MIT.

But besides that, there's another interesting point which is that noticed during the months of October and November, the two leftmost lines here, we are sending a lot more traffic than during the month of December.

So, if you were at MIT and are trying to plan how much traffic you expected to be transmitted during a month, this makes it hard because there was a 50% drop in the amount of traffic from one month on this Kazaa network.

So we need to introduce this notion of packet switching to deal with this problem of unpredictable rates.

OK, so in many kinds of data networks, it's simply not possible to predict exactly how much traffic you're going to need to send at any point in time.

And so the idea is to use something called asynchronous multiplexing instead of time division multiplexing.

And the idea is as follows.

Suppose we have our three nodes, A, B, and C all transmitting data through some switch out over a wire just like we did before.

If you look at the traffic that's being transmitted on this wire, what you'll see is that it's not necessarily going to have the same regular pattern that we had before.

The idea is that any node is allowed to try and start talking on this wire any time that it wants.

It's asynchronous.

It's not timed.

There's no regular slots when nodes are required to transmit.

So this might be traffic for A.

This might be traffic for A.

This might be traffic for B, A, A, and C.

OK, so what this suggests, and noticed that these packages of data that are being transmitted are of different sizes.

Typically these are called packets.

And that's what this is packet switching because we are sending the full packages of information that are of a variable size as opposed to these fixed size frames that we were transmitting before.

And notice that there is no requirement that A transmitted any specific time.

A is allowed to try and transmit as much as it wants.

But this does introduce a complexity because it makes it hard to plan for the capacity of the network.

So suppose that each of A, B, and C have a 1 Mb per second link, and suppose that this link is also 1 Mb per second.

Now, clearly if A, B, and C are rarely sending any traffic, then everybody may be able to get all its data onto the network.

But if A, B, and C all decide that they want to send at the same time at 1 Mb per second, then there's going to be a problem because they're going to be trying to send more data over this link them the link has capacity for.

And so what that immediately suggests is that in these boxes, in these packet switch networks, there is some kind of a queue.

OK, so we're doing queuing within the network.

But it also, so you might ask the question, why don't network designers design these packet switch networks so that you never have a link that's connecting multiple endpoints together that doesn't have sufficient capacity to transmit all of the information from all of the endpoints that may be transmitting data over it?

But if you think about this for a second, that's going to be a really bad idea.

So suppose we are on the Internet.

And suppose that I am designing simple website.

I want to set up a website for students of 6.033. If I needed to make sure that that website had sufficient bandwidth that could support connections from everybody who is on the Internet just in case everybody who's on the Internet decided to access that website at exactly the same time, I would be in big trouble, right, because I would have to have hundreds of terabits of bandwidth available to me in order to be able to support that.

So clearly I'm not going to do that.

Instead, what I'm going to do as a network designer is simply try and ensure that most of the time my network has enough bandwidth available for the users were going to be accessing it.

But some of the time, there may not be.

I may sort of under-provision the network, may not provide enough resources.

And so during those times when the network has more data that's trying to be sent over it than it could possibly send, we call those congestion times.

And when a network is subject to long periods of congestion, these queues that are sitting in the network may fill up.

So these queues are some fixed size.

They are stored typically in RAM on a device, and they can over-fill. And when they over-fill, we have to do something about it.

OK, so this problem of congestion in networks leads to all sorts of kind of interesting trade-offs in the design of these packet switch networks.

And typically the bottom line is basically, and we'll talk about this in much more detail, but the bottom line is typically these packet switch networks have to be able to drop data some of the time.

If a queue over-fills, the network has no choice but to refuse to transmit data for somebody.

But because these packet switch networks are composed of multiple routers along the way, it may be that the data is actually dropped deep in the network by some router not when I first try and send the data, but five hops down the line the data may be dropped.

And so these kinds of things complicate the design of these systems.

Queuing means there's variable delay.

OK, and congestion means there may be data that's lost.

And to deal with lost, we may have to retransmit packets.

And so this gets the sort of fundamental notion that's going to come up in the design of these packet switch networks, and that's that the extraction that these packet switch networks provides is so-called best effort networking.

And that means at the lowest level, these networks don't guarantee that the data that you send over them will actually be delivered.

All they say is the network is sort of promising that it will try hard to get the data to be delivered.

But you as a user, the applications that are using this run network interface need to understand that this variable delay, they need to understand that there's congestion that can happen which means that packets can be dropped.

Congestion also means that typically packets may be retransmitted.

And so that means that all of your data may not arrive at the other end.

And so this is the best effort network abstraction, and we're going to talk about these kinds of best effort network abstractions exclusively for the next several weeks as opposed to these circuit switch networks.

And what we're going to do is we're going to see how we can build up a set of software that runs on top of these best effort networks that allows us to provide things like, for example, some reasonable guarantee of reliable delivery, that data won't be lost as data is transmitted over the network.

So we will see how we build up layers on top of this best effort network stack that allow us to provide certain guarantees to the user.

So that's it for today, and tomorrow we will talk about, starting Monday we will talk about the link layer interface for these things.

Don't forget about the exam.