MITOCW | 17. Packet switching

PROFESSOR: So my name is Hari Balakrishnan, and I'm going to take you through the rest of 6.02, doing the remaining lectures in the class. So far in 6.02, what we've looked at are ways in which we design a single communication link. So we know how to take two computers or two nodes and design what a link between them might look like. And this link might be an actual wired link. Or it might be a radio link. Or it might be an acoustic link.

There's some medium over which these two guys communicate. And the main ideas we've looked at have to do with coding, in particular, channel coding, which is a strategy to combat noise and errors that might show up on the channel. And then in order to match what we communicate to the characteristics of the channel-- for example, the ability of the channel to deal in sinusoids-- we studied modulation and de-modulation.

So those are the two main elements that we studied. And in both of these, we looked at both how you do this to achieve reliability-- because ultimately, we want to communicate information in a way that's reliable-- and do it efficiently. In particular with modulation, we look at a scheme to share a medium amongst multiple conversations, frequency division multiplexing, which is the topic of one of the tasks on this lab.

And with coding, we looked at ways in which you do this coding in a way that isn't just replicating every bit but involves some linear algebra operations that allows you to gain efficiency. So the rest of the class is really about taking for granted our ability to design communication links and putting them together and composing them to build networks.

So the basic problem is actually very, very easy. The problem is, you're given a set of nodes-- let's say computers-- and the problem you want to solve is to come up with a way by which you can allow any computer or any phone or any device on this network to communicate with any other device on the network. And that's the problem.

So you're given n nodes. And you want all-to-all communication. Now, this is a little different from the kind of-- there are other networks you could design. You could design a network where you're given n nodes and you have 1 to n communication. There's one transmitter, many, many receivers, and you want to design a network for that purpose.

What's an example of a network where you have one transmitter, many receivers, and you just want to build something that makes that work? Radio is one example. Television is another example. And those are good examples. In fact, for those kinds of one-to-many networks whether you have-- or k to n networks where you have k sources of information and n receivers and k is a lot smaller than n, it'll turn out that the basic frequency division approach makes sense. I mean, that's how radio stations or TV stations work.

Someone in the US-- the Federal Communications Commission has decided to allocate different chunks of frequency to different TV stations and different radio stations. And the assumption is they're always going to be using it. Turns out that assumption may or may not be true. But under the assumption that they're always going to be using it and you have many, many, many receivers, you just divide up frequencies and allow them each to transmit in their own frequencies.

And then you have a receiver that's capable of tuning to different frequencies. And you get the information or the channel that you want. We'll actually come back to that problem a little bit in the next two lectures. But for today, the design problem-- and going forward, the design problem you should have in mind is you want a network where you have all-to-all communication and you want to be able to support any application.

This is a big deal. We're not just designing a network to allow telephone calls to work. Or we're not just designing a network that allows you to do video conferencing. We're trying to design a network where any application can run on it-- in particular, applications that you might not have envisioned. This is the reason why the internet works really well is because when they designed the internet, they designed it under some set of assumptions. But they were really, really smart to design a network that made minimal assumptions about the application.

So it's a network that's good enough for almost any application, though it isn't perfectly optimal for any application. It's just good enough for everything. And that's a really good characteristic of a well-designed network is if it can work even for things you didn't even dream of. When they build the internet, they suddenly didn't dream that the web would exist. They didn't dream that people would be tweeting and telling people they're going to the bathroom or whatever they do on Twitter. I mean, they designed a network.

And it just kind of is amazing that all these applications can work. So the question is, what did they do correct? What did they do right? And what are things that-- what general lessons can we learn from it? And the general high level lessons you learn actually apply to any system you build. It'll turn out that whenever-- if you're confronted with a real world problem in an industry or research or whatever, very often you're trying to make decisions on what you need to be doing.

And it's very tempting to make decisions based on what you think it's going to be used for. But very often, what you end up eventually using it for is very different from what you thought in the beginning. So it's good to have applications in mind. But it's good not to embed too much about those applications in the design of network. So the high-level principle here is how you can do something that works well enough without making too many assumptions about what's running on top of it.

There are two big themes. They're the same two themes that we studied before that we're going to keep coming back to. The first is efficiency. And the second is reliability. The same two themes we come back to over and over again. There's a third important theme about network design, which has to do with scalability. I mean, how can you make it work so this network can work for millions or billions of devices and billions of computers?

That's a topic we're not really going to talk about. I'll get to it in the last lecture. But 6.033 and 6.829 we'll talk about those issues. So let me start first with efficiency. If I tell you how to build a communication link that can communicate between any two devices or any two computers, it should be pretty straightforward to now design a network that allows all-to-all communication. Something out? That's a mouse. Great.

One way you can design this network is to simply take your communication link that we know how to build and do this. Just connect every pair of computers or every pair of nodes to each other. I'm probably missing a few of these. But this is a great network design because it's composed of bunch of links to build a network.

So why don't we do this? Or maybe we should do this. What? It's too expensive. Why is it too expensive? Sorry? You know, how many of you are in Professor [INAUDIBLE] recitation? Great. I understand he gives you guys money to answer or if he makes a mistake. I'm going to do the same thing. Whenever I make a mistake, Professor [INAUDIBLE] will give you some money.

[LAUGHTER]

So I actually-- I mean, don't hold me to this. But why don't you guys answer? This is pretty straightforward. How many links do you need? n [? choose ?] 2. It's about n squared, right? So n squared, depending on the context, it's either too big or too small. But it's about n2 to n squared lengths.

It turns out that's actually a pretty large number of links, because-- and the notes talk about some of the reasons why this is too expensive. But the other reason it's a problem is that it's one thing to design a network where every computer in this room can talk to each other. And conceivably, we might get tangled up in all these wires. But we could imagine laying wires between every pair of our computers and communicating.

But there are two reasons this is a big problem. I want to communicate with computers in California or China or wherever. And individual links going across the world and my computer to China and your computer to another computer in China just doesn't scale. It doesn't work very well. And the second problem, the reason why this issue matters is that not all communication links are wires.

In fact, right now, the most dominant mode by which people gain access to the internet, including right now in this room, is through radio, is through wireless. And this is a shared medium. So it's not like we can somehow put these wires together. We're going to have to share this communication medium. We're going to have to share this communication network. And somehow we have to come up with a strategy to do this efficiently.

There's a few different principles involved in how you design networks. But the main one is that we're going to construct a special computer called a switch. And a lot of what we're going to be doing has to do with what we do in the switch. The other part of what we're going to be doing is what we do in the computers itself.

So our network is going to be designed using a set of rules that are obeyed and implemented and followed by the computers. OK, a special set of rules that are implemented by these computers called switches and a special set of rules that are implemented by the end computers, by the devices on the network. And together, they're going to make our communication work.

So the high level plan is going to be that we take these computers and rather than put wires between every pair of them, we're going to connect them together into-- perhaps there's lots and lots of computers and many of them get connected to one of these boxes, which is a switch. And a switch may connect to other switches.

And some of these switches may have other computers attached to them. And then eventually you might get to other end computers. And when you build a network like this, a structure like this, this kind of a picture is called the network topology. A switch has one or more links attached to it. These links could be wires. They could be shared things like-- like this thing here is a switch.

It has no visible links, but it probably has one wire link connecting it via ethernet to the rest of the MIT campus. And out here lots of computers right now are connected to it. It gives the illusion that each of your computers has a separate link to the switch. And we look at how that illusion is maintained and done next time, next lecture.

But this is an example of a switch, probably the world's-- you know, this thing is made, I think, by Cisco. So they charge $600 or $800 for it. But really, you can buy it for $40. When you put the word "enterprise" next to anything you sell, you can mark up the price. But anyway, the world's cheapest switches are on Wi-Fi access points.

So you connect the stuff together into a topology. And the job of the switch is to look at messages that come in from these links and figure out what to do with those messages and make sure that together they coordinate to get messages to the destinations to which you wish to send those messages.

So here's a picture that I got today from MIT's ISNT, which is a picture of MIT'S network. So I just want to give you a sense for what this looks like for a campus like MIT. So the first thing to notice is that this is actually-- it's got some redundancy built in. You don't see it in the picture.

But really what's going on here is that we have these two routers here. In the context of the internet, these switches are also called routers. It's taken me 10 years to pronounce it router because where I was brought up, they pronounced it "rooter." And many people say that. But in the US, they say "router."

So anyway, these routers here, there are two backbone routers. And they're actually-- each of these guys, these other routers in these different buildings, are connected, actually, to both of these. So the idea here is that if one of those links were to fail or if one of these routers were to fail, the other guy would take over and handle this traffic.

Under normal conditions traffic is kind of balanced between these two different routers. So some of these computers-- some of these other routers are connected to one of them. Some of the other routers are connected to the other. And together they work to provide connectivity.

These backbone routers get connected to these things that are called external routers, which are routers that connect to various other networks and internet service providers that MIT uses. MIT is extremely well connected. The amount of bandwidth coming in and out, as you might have noticed doing, I don't know, bittorent or whatever the cool people do these days with networks, is phenomenal.

MIT commercially uses Sprint, which is an internet service provider, uses level three, which is probably the biggest internet service provider in the US. This thing called [INAUDIBLE] tech is-- I found is that-- so MIT now does telephony through the internet. So it's voice over IP as opposed to the old telephone system. So a lot of that voice traffic goes through that network service provider.

Other things here-- this NOX is-- I think it stands for the Northeast Crossroads or something like that. It connects to a network called the internet two, which is a network connecting many universities in the US. And it's a very, very high bandwidth network. And so if you were to communicate with, say, a Stanford or something like that, it wouldn't go over the public internet. It goes over a network that's essentially not commercially paid for but is a private network connecting different universities.

And it has a connection to Comcast. So many people who have Comcast in their homes in this area tend to have good-- or are supposed to, in theory, have good delay, low delay to MIT. Out here on this side, MIT is connected to other research and education networks. It has high connectivity to fermilab and to [INAUDIBLE], because I'm assuming there's a huge amount of data flowing because of things like the LFC experiments. They send terabytes or petabytes of data back and forth. So you need high bandwidth. So they have their own network connection to do that.

This NLR is something called the National Lambda Rail, which is another high speed network connecting a bunch of East Coast universities. And then out here on the edges, you have MIT connecting to other-- out here-- other internet service providers. This thing here is funny. It's called Big Ape, which is actually-- its called the Big Apple peering exchange.

It's this place in New York City where a lot of people-- a lot of companies and internet service providers have gotten together. And you can just connect to other networks. So MIT connects to, I think, 13 other networks on a non-payment basis. Whereas to internet service providers you have to pay money, you can peer with other networks essentially on a bilateral agreement. So I carry your traffic, you carry my traffic.

So it turns out that out in New York, there is this building where a lot of these different networks have gotten together. And MIT is one among those networks. So it has extremely good connectivity. But you can see that already MIT is a tiny campus. And already it's got such rich connectivity to the rest of the internet. I guess as far as college campuses go, it's a big campus.

But still, in the grand scale of the internet, it's a tiny thing. And you can already see that there's so much complexity and so many things going on inside the network. So the question is, how does this network get designed?

And the main idea that I want to get at today is this idea of packets and packet switching. So the design principle that's used in communication networks is this idea of packets and packet switching. There are some special rules-- simple, special rules that you have to follow to allow these switches to send messages back and forth. And in fact, these are fairly obvious rules.

But what's remarkable about them is how simple they are. And they can work. The main idea is that you take your message and you have to decide who it needs to be sent to. And you have to decide who it's coming from. So if I decide that I want to send a message to you in this network, my computer and your computer have to somehow have names associated with them. And in the context of packet switched networks, these names that we associate with-- ideally, these names should be associated with computers. But they turn out to be names that are associated with the link that you use from your computer to send these messages.

These names are called addresses. So very concretely, if I have a computer here, my computer may have a name. But this computer here has two or three different links coming out of it. If I connect this-- even this thing here or this ethernet link to the USB port here and I connect a cable to it, that's one link. The Wi-Fi on this is another link. If I turn the Bluetooth on and use that, it's a third link.

Each of those links has a different name. The name here is equivalent to an address. Each of these things is an address. So when I send a packet, I have to tell you my address. And similarly, if I want to send someone else, some other computer, a packet, I have to specify the address that I wish to send it to. So that's the first rule of packet switching. It's specify an address-- in particular, specify a destination address. And you specify a source address.

Now, the idea is once I specify the addresses and I construct a message, my message has some bits in it. Maybe it's a file. Maybe it's a piece of video or whatever. I add something to that message, which I call the header. The header has a bunch of fields in it specifying something about what should be done with the message.

But the only two important things here-- there's three or four things that you need. But the non-negotiable part that you need is a part of this address-- part of this header should specify the destination address. There's other parts of it that specify the source address, as well.

The basic structure is very simple. I send a message in which I specify a destination address. And the job-- and my job is done as the source for the time being. I send it to some switch. I'm connected to a bunch of switches. My computer picks a switch to send it to. The switch it picks is typically the switch that that link is connected to.

So if I am connected right now through ethernet and Wi-Fi, there's some rule on my computer that decides whether to use ethernet or Wi-Fi. And let's say it decides to use Wi-Fi. It sends this thing, this message, with this destination address to that access point. And that's the first switch it goes to.

And then it becomes the switch's job to figure out how to get this message to the actual destination. This combination of a header that includes the destination address and some number of bits that corresponds to the message, this entire bag of bits is called a packet. For something technically to be considered a packet, it needs to have an address on it. Or it needs to have something that's equivalent to an address on it that then allows the rest of the network to decide how to send that packet onward.

This is a lot like the way the post office works. When you deliver-- you write your letter. You write who it's from. And you write who it's to. You put it in the mailbox. Your job is done. And maybe at some later point, if it's registered post, you get an acknowledgment that the other guy received the message.

Packet switch networks are very much like that. They just work a little bit faster. Now, why is this idea good? Now, the reason this idea is good is that it's extremely robust at dealing with failures, at least in theory, because it becomes the job of the switches and the network to talk to each other and run some sort of algorithm between each other that allows them to always construct and maintain some information that allows them to always, no matter what the failures are, as long as there is some path that takes you from here to there in the network, regardless of failure, as long as the underlying topology allows you at least one path to get between one place to another, the switches figure that out.

And if you want to make a network more reliable, you add more switches and more links. And you figure out how to make it reliable. The end points and nothing else have to really bother with that problem. And you can take portions of the network that are unreliable and add some redundancy to it, add more pads to it, and run some other algorithm that allows the switches to figure out how to divert, how to route packets, or how to move these messages across.

And this idea is a brilliant idea. It looks completely obvious in retrospect, like all brilliant ideas. But it's actually quite recent. I think they celebrated its 50th anniversary quite recently. In 1959, Paul Baran, who was at the RAND Corporation at the time, wrote one or two-- you know, it's not often you can call a paper seminal. This is seminal. This is really important.

It just changed the way communication worked. His paper is called on distributed communications, introduction to-- the first one was "Introduction to Distributed Communication Networks," where he looked at various ways you could design these network topologies and completely theoretically argued that this design would allow you to build a network that could withstand various kinds of failures-- in particular, even adversarial failures caused by enemy attacks.

And the second part of the story with these messages that are in packets is he said that if I want to communicate a large amount of data, what you should do is break it up into smaller pieces. So you take a message-- if you have a big file to transfer, don't put it in one big packet. But instead, you break it up into smaller pieces and send each piece into the network.

So a big file gets broken up into many packets. Each packet becomes an independent, atomic unit of delivery. Packets could be sent along very different paths, in principle, between any point in the network and any other point in the network. And at the other end, packets could arrive along different paths. And as long as there's some working path, it's the job of the network to figure out how to get those packets through. That's the basic idea.

So the first one is this idea of using an address on messages. The second one is the idea of breaking it up into packets. And in particular, these packets could all take arbitrary paths. The sources and the destinations don't determine the path. The switches determine the paths that you have to use, using some algorithms that we're going to be studying.

So is the idea clear? Does everybody understand kind of what a packet switched network is? The textbook-- the notes also talk about other ways of doing it. The other big way of doing it, which predates this, was what was done in the Bell Telephone network. It's called circuit switching. It's a different idea. I'm not going to talk about in lecture. You can read about it. It's important stuff to read about, but mostly cultural at this point, because almost every network is packet switched today.

So any questions about this idea? It's pretty simple. OK. So here's an example of the world's simplest packet header. This is the 6.02 reference design. So for the labs and everything else, this is the packet header we're going to be using. It has just four fields-- a destination address, which specifies where the packets should be sent.

It has something called the hop limit, which I will talk about in a couple of lectures from now as to why we need it. It has a source address, mainly because when I receive a message-- when this computer receives a message from someone, it often wants to send a message back in response. And having the source address allows it to send a message back to the person who sent the message. It's just for two way communication.

And it has a length. And the reason for having the length is convenience. You kind of know once the header is done, how many bits do you need? Or how big is the actual data corresponding to the packet? It's also called the payload. How big is the payload in the packet?

Now, real world packet header is a little more complicated. Just for concreteness, this is what IP version 6, which is the version of IP everybody's trying to move to, the internet protocol, looks like. It has the destination and source addresses. It has the hop limit. It has the length. And it's got a few other things that we're not going to worry about. They have to do with allowing switches to prioritize certain kinds of packets so that-- I guess things like if you were doing Skype or voice telephony, you might want to schedule those packets differently in the switch so you get low delay.

Or if you were-- maybe the CEO's packets get higher priority, whatever. You could come up with policies on deciding how you switch these-- how you schedule these packets. So that's the main idea in packet switching.

For the rest of today, I actually want to talk about two performance metrics that people use to evaluate how well a packet switching network is doing in terms of how properties that users care about. And I want to also explain to you why this idea works-- like, this idea that nodes just send data. All these nodes are sharing a communication medium-- I'm sorry. Sharing resources in the switch.

So this node can send packets. This node can send packets. This node can send packets. And the switch must have a plan in mind to-- let's say that all these packets are going to some destination and have to go on this link. This switch must have a plan in mind for deciding how to take all these packets that are coming in and sending them along this link.

I mean, for example, what happens if packets come too fast for the switch to handle? The speed of these links when they all simultaneously send packets could be bigger than the speed of the link going this way. What does the switch do with that? Does it just drop the packets? Does it hold onto them for some time? What does it actually do?

And I want to do this first with a very simple picture that tries to get at why this idea really, really actually works. This idea that makes packet switching work has a fancy name. It's called statistical multiplexing. So let me explain what that means.

Let's take it with a very simple picture. So let's say that you have a switch with one link coming out of it. And let's say that the speed of this link-- I need to get into some metrics here. So links are measured in terms of how quickly-- how quickly is the wrong word. In terms of the rate at which they can send data.

And there's another metric, which is the delay of the link. So I'll get to both of these more carefully in a bit. But the important thing right now I want to keep in mind is the rate of the link. This is the rate at which it can send bits per second. It's a measure of throughput. So it's typically measured in bits per second.

So let me actually imagine that the rate of this link is one megabyte per second, which is 10 to the 6, which is a million bytes per second, or about 10 million bits per second. Let's imagine a simple network that looks like this. Let's imagine that all these links are also coming in at 1 megabyte per second.

If somebody came and told you, here's the design of my network. I have a switch. It's connected to three computers, each of which can-- is connected with a link whose maximum speed is one megabyte per second. And this switch is going to connect to something else downstream, maybe another switch, and it goes somewhere else. And the speed of this link is one megabyte per second. Is this a good network design?

How would you go about assessing that question? Is this good or bad? How would you know? Yes?

AUDIENCE: [INAUDIBLE]

PROFESSOR: Right. So let me ask this before we answer this question. Let's say this was 10 megabytes per second. Is this a good network design? It is. You're paying too much, though, because I mean, really, this link is too fast for the amount of load that is coming in. But yeah, you know, it's a reasonable network design.

But the real question is, if it's one here, is it a good network design? And the answer, as the gentleman here pointed out, is it really depends on how much traffic or how many packets per second or bits per second these different computers are going to be sending. Let's say that they all actually send-- when they send traffic, they send at one megabyte per second. And when they don't send traffic, they're quiet.

How would you determine whether this is a good network design, whether this works or not? Like, in practice, on average, how often can each of these guys be sending before you determine that this is probably not-- this network isn't going to work. Yeah.

AUDIENCE: [INAUDIBLE].

PROFESSOR: Right. And they may or may not be equal. Ideally, what you'd like is just to make sure that over some window of time, they sent slower than the rate at which this link can ship packets. Now, the reason why packet switching works is that when you build a network like this and you scan it out to bigger numbers, it turns out to be extremely unlikely that everybody using the network exercises the network at exactly the same time.

I mean, a bunch of people might have their computer on. But if you think about how it's used, you click on a link and you get a bunch of stuff showing up. And then you click on-- you read it for some time. You click on a link and something else shows up.

Or if you're watching a video stream, video is compressed. So if the scene changes very often, you end up using a lot more of the-- in terms of the bitrate. But then every once in a while, it's one of these old Russian movies where nothing's changing for 10 minutes. And it's very heavily compressed.

And then you get to Schwarzenegger and it's blowing your bandwidth limit. So I mean, it's kind of like that. Traffic is bursty. So when traffic arrives in bursts and the users are not all highly correlated with each other-- I mean, from time to time, you do get these correlations. These are called flash crowds. Presumably this happened last night. Everybody is hitting refresh on the *New York Times* website.

And presumably what's happening there, of course, is that these websites really know what they're doing. So they've actually provisioned with the expectation that starting from 8:00 PM, everybody is sitting there glued. Nothing's changing, but everybody's hitting reload.

And they've designed this network-- they've provisioned their network to allow for people to get the answers they want or the results they want to see. So here's some pictures. So what I did was I took-- I sniffed on the traffic in this room. So here's the kind of stuff that you see.

So this is the traffic in this room during lecture. Now, this is actually not this semester. But I would assume that it's fairly typical. I should also say this was during-- well, I don't want to say. The x-axis in these pictures is time. The y-axis is the number of bytes that was sent.

So you can see that what I've done on top is I've broken time into 10 millisecond windows. So initially on top, it's every 10 milliseconds I just count the total number of bits-- actually, total number of bytes that was sent. Now, you can't read the scale on the y-axis on top. But on the top curve, it goes up to 200,000 bytes in a small 10 millisecond window.

Then the curve down here does the same thing. But I've picked 100 millisecond window. Now you can see that what has happened when you've picked a bigger window of time? Has it become smoother or less smoother? What can you say about it? It's become a little smoother. But surely there still are these peaks. The bursts do become smoother. But they don't completely disappear.

And what's remarkable about network traffic is that these bursts never completely disappear, but they do get a little smoother as you aggregate over more time. Over 100 millisecond windows, that's what it looks like. Over one second window, it looks smoother. But you can actually see that from time to time, there are these big bursts that take up a lot of the-- that end up over any window of time that you expand out, there's still some probability with which you'll see a big burst of traffic showing up in that window.

That's kind of a nice and noteworthy characteristic of kind of real world data traffic. In fact, even when you go to 10 second windows, it says, look, I'm looking at 10 seconds at a time. You get stuff that looks like this.

MIT runs a website you can get access to using your web certificates. It's called mrtg.mit.edu. You can actually go to this website and you can see, for different switches, including ones in your dorm or wherever you're living if you live on campus, you can actually look at the statistics from your router. They do this on a per switch level. It's kind of interesting to see when people use this network and when they don't.

I think an interesting characteristic of MIT's networks is it turns out if you look at some of the dawn network traffic, it peaks at between 1:00 and 3:00 or 1:00 and 4:00 in the morning, which is probably good, because honestly, I think MIT should negotiate preferential pricing with ISPs, because no one else is using those ISP networks at that time.

So it would be actually-- it turns out I learned that the Amazon Kindle kind of does that. When you do your newspaper subscriptions, they actually send it through wireless networks, through this commercial 3G and 4G wireless networks. And I believe that what they do is they send it to you in the middle of the night when not many people other than at MIT are using those networks.

So you could take advantage of some of these time varying properties [INAUDIBLE] doing it. So why did I tell you this story? The same thing-- I showed you these time windows. The same thing applies when you bring many, many users together. The odds that we all are going to click on some link at exactly the same time and all of us cause a burst of traffic to happen exactly at the same time is extremely small.

Now, it can happen if there's an adversary in the network. If there are bad guys-- and how many of you have heard of denial of service attacks? Yeah, DDoS, Distributed Denial of Service attacks. I understand if you know Russian you get an edge in doing it. So these things are launched because they commandeer a whole bunch of machines and they coordinate an attack. They destroy the assumptions that make statistical multiplexing work because the normal assumption is people are not exercising the network at the same time. So you're not attacking some website or whatever at the same time.

But if you coordinate an attack, then you make that assumption not hold, causing congestion to happen, causing traffic to exceed what your network link can support. But under normal, non-adversarial conditions, the assumption is that people are randomly gaining access to the network, which means that you can actually get away with the design of a network that looks like this as long as you study statistics like the average amount of traffic.

Like, on average, the guy is not going to be sending more than-- this node is not going to be sending more than a certain amount of traffic when measured over some period of time. What happens when people send traffic in a burst? What happens when, from time to time, in fact, you see these bursts of traffic, right? You look at this picture here. You do it over a one second window or a 100 millisecond window and you see these big peaks of traffic.

Lots of bytes at 100 millisecond window. What that really means is that this switch here is going to be getting traffic from different users that probably exceeds-- you know, is perhaps the sum of all of the input links. So it's a large amount of traffic. If you have a design like this, something's got to give, because you're getting water or packets coming in at one megabyte per second times three. And you've got a link that can only send one megabyte per second.

So what can you do? What can the switch do?

AUDIENCE: [INAUDIBLE]

PROFESSOR: The easiest thing it can do is just drop it. Just say, you know what? Just drop it. You laugh, but I'm telling you, sometimes dropping it and letting the end point deal with it is a better strategy than holding onto it and simply keeping it in line. It's like, you've got to be careful, right?

I like the idea of storing it. But for how long do you store it? How much do you store? For example, if I look at that burst of traffic here and I have a network like this and I look at this big burst of traffic here, over a 10 second window, I'm seeing traffic that's probably, in this example, perhaps 10 or 100 times bigger than the average.

The average is sitting down somewhere. And maybe this is 10 times the average. The peak to average ratio might be 10 to 1 or 20 to 1. So how much should you store inside the switch? If you were designing a network and I told you, well, all right, good idea. Why don't you store the packets. You're going to put these packets into a data structure called a queue, right?

Packets come in. Packets go out. Packets go out whenever the link is able to send packets. You keep shipping packets out. In the meantime, traffic's coming faster than you can handle. You're going to put stuff in a queue. How much? Do you want to keep everything? If you did, you'd be like Disney World, because they have these lines that go forever, and nothing's moving and everybody just piles on the end of the line.

This is a tough question. We're going to answer this question somewhat. There's no single, easy answer to this question. But the rule of thumb that I'm going to have you keep in mind now is you're probably going to keep between 10 milliseconds and 100 milliseconds worth of traffic. I'll get to why later. For now, it's some small amount of time worth-- amount of traffic.

The reason why you need this queue is to absorb a burst of traffic that you're not able to immediately send. But the important principle in a packet switch network is, you need a queue, but they're a necessary evil, because the only thing that the queue is doing for you is absorbing the burst. But the only thing-- the bad thing that it's doing for you is adding delay.

Just because you have a queue, the network ain't going to move faster. The network's moving-- the link's moving at the same speed whether you have a queue or not. The only thing the queue is doing is it absorbs a burst so that whenever the network link is able to send packets, you can ship packets from the queue and you don't want to drop too many packets.

Now, if you're lucky, the size of the queue is enough to absorb all of the burst. And then the traffic eases and you get to send the rest. But if you're unlucky, the queue overflows and you drop some packets. And then the endpoints have to somehow deal with it.

So what are the things we've looked at? Packet switch networks as defined by a header, which includes the destination address. The way the network works is that the sources just ship a packet with the header that includes the destination address. The switches somehow are going to figure out how to ship-- how to get those packets to the destination.

The reason why this stuff works is because of the statistical multiplexing. And finally, the reason we need a queue in a packet switch network is to absorb these bursts. So what I want to do in the remaining six or seven minutes is to tell you about the other metric by which we're going to evaluate our networks.

The first metric I introduced already is the rate of a link. When you have links of different rates, you can also define the rate for an actual communication. When a source sends a packet to a destination, you can measure the rate at which bits are arriving at the destination. That's the throughput of the data transfer of the bit rate.

The other metric we're going to care a lot about is called the delay. The fancy term for delay is latency. I really don't know why they have two terms. But you know, from time to time, people use the word delay or latency.

And by the way, I'm going to try hard to use the word rate here or bitrate or throughput. Often you see the word bandwidth, like oh, my bandwidth is 10 megabits per second. And that's actually fine to use, except it's confusing in a real communication system because we're going to-- we've already used the word bandwidth to refer to a frequency. And so we've already said the bandwidth is defined in terms of, say, Hertz or something like that.

And it's just a little confusing to also use bandwidth for rate. So we're going to try to use words like bitrate and throughput to refer to bits per second. So delay is measured in seconds or milliseconds or microseconds. And what we want is you have a source that sends a packet or set of packets-- let's say a single packet to a receiver going through a network of switches.

And I want to ask-- if I send a packet at some point in time, let's say at time 0, when does that package reach the receiver? That's the delay for a single packet. So I just want to explain to you how to calculate this or how to measure this. So let's say that the packet has a size of L bits.

So what does the answer depend on? Let's take an even simpler example. Let's say that I have a sender. I have a receiver. I have one link between them. No switches. And the packet has size L bits. I send a packet at time 0. When does the last bit of the packet show up at the receiver? Yes?

AUDIENCE: [INAUDIBLE]

PROFESSOR: Good. So I need to define this thing here. Let's say that the bitrate of this link is c bits per second. So I have l bits and I have a link that can send packets at c bits per second. Therefore, something here should be l divided by c second. That is from the moment I start shipping these bits, from the time delay between when the first bit arrives-- first bit arrives at the receiver and the last bit arrives at the receiver, that time distance or time difference is c divided by l-- sorry, l divided by c seconds.

Because I ship these bits. These bits go back to back over the link. If I look at when the first bit arrives and I look at when the last bit arrives, that time difference is the spacing-- the time difference between any two bits showing up at the receiver is 1 over c, seconds, because if the link can send c bits per second, any two bits are separated apart by 1 over c seconds.

Therefore, from the time at which the first bit arrives to the time at which the last bit arrives, that distance is l over c. That difference is l over c seconds. This l over c has a name associated with it. This is called the transmission delay.

Now let's say I want to send just one bit. I have to now-- I send a bit at some point in time. And that bit shows up some point in the future, because it can't show up immediately. If it did, we'd probably have to change the laws of physics because speed of light is no longer valid as a finite limit.

So what is the time between when I send the first bit and when the first bit shows up here? What does that depend on? Like, let's think I want to communicate to the moon. I send one bit of information. I put it out onto the-- or even one sample. I put it out on the radio or whatever. How long before it gets to the moon?

AUDIENCE: [INAUDIBLE]

PROFESSOR: Depends on what? Does it depend on the rate at which I can communicate? No. What does it depend on?

AUDIENCE: [INAUDIBLE].

PROFESSOR: Sorry? You guys said it. What is it?

AUDIENCE: [INAUDIBLE].

PROFESSOR: Speed, and the speed of what?

AUDIENCE: [INAUDIBLE].

PROFESSOR: It's the speed of light in that medium, or speed of whatever the signal you use is. If it's acoustic, then it's the speed of sound over the medium. So it depends on the distance and it depends on a speed at which a signal can propagate through that communication medium-- for example, the speed of light. So the distance is b and the speed of the communication medium is, let's say, v.

That thing is called the propagation delay. So let me organize this properly so I'm not confusing everybody with these different terms. So so far we've hit two sources of delay. The first source of delay, which I said second, is the propagation delay. This is the time it takes for the first bit to get to the other side. It depends on the speed at which a signal propagates through the medium and the distance between sender and receiver.

So sound travels at one foot per millisecond, I think, roughly something like that. So if I'm doing acoustic, that dictates the propagation delay. The second delay is the transmission delay, which depends on this l over c. The third delay is whatever processing delays there are. That, for example, is when a switch gets a packet, it has to look at the packet's header, figure out the destination, do something to that.

There's some competition time that the switches have to work with. That delay is called the processing delay. This is purely some sort of computing delay. And it's usually very, very small. And the fourth delay is the queuing delay, because it could be that you get these packets in and they have to sit behind in a queue. And that imposes a delay in communication. So that's called the queuing delay.

And it's usually a very variable source of delay. On many networks, these other delays are constant-- not always, but generally constant. The transmission delay may or may not be constant. But usually these are more constant. The queuing delay is not a constant delay. And the actual delays that you experience when you click on a link, there's lots of reasons why the website is slow. But these are a principle, dominant factor in many, many cases.

So we'll pick up on this next week after the quiz two stuff. You deal with quiz two and p set 6. And we'll continue with multi-hop networks.