MARTEN VAN DIJK: So today, we're going to talk about communication networks. Communication networks is a great application of graph theory. So what we're going to study is, how do you route packets through networks? So you have the internet, which is a chaotic network. It's not organized. We are interested in highly structured networks and you can find them, for example, in parallel computers, where you want to route the data flow. You can find them in certain telephone switches networks and so on.

So we are going to talk about a few very special ones, binary trees, and then slowly we will figure out what all these performance measures really mean. This one has to do with latency. We have switches, their size, the number of them, congestion, and then we will slowly get down to Benes network, which is a really beautiful network with beautiful parameters. And we are going to prove those.

So let's start off with the first one, the complete binary tree, and let me draw it for you. In this network, we will have a root and let me just draw it first We have vertices that represent here the switches. So these circles-- let me explain it over here-- actually represent a switch.

And the idea is that these actually direct packets through the network. And these packets are fixed-size packets of data, so like, I don't know, say 4,000 bytes or bits or whatever the network wants you to comply to. So these are fixed-size pieces of data. So what we want is we want to be able from every terminal-- and the terminal I will denote by a square-- from every terminal, I want to be able to reach any other terminal.

So what is a terminal? A terminal is like a computer or something like that. It's actually the source and the destination of data. So what we are looking for is how can we route-- how we can find a network of switches that are connected through wires, fibers, or-- yeah? What's the question?

AUDIENCE: Can you move down a bit the top of the-- how it's getting cut off? No, the--

That one.

MARTEN VAN DIJK: Oh, sorry.

AUDIENCE: All right. Thank you.

MARTEN VAN DIJK: So what we want is we want to route packets that's come from any terminal to any other terminal. That is what our goal is and we want to make sure that that is efficient. So the first one is this binary tree. And let's see how this may work.

We may have switches that actually have inputs coming from terminals. And the switches may also output to terminals, so here at the bottom. At this site, we have a similar structure. this is the root of the tree. We have another switch over here. We go down, we go up here, and once more, like this. And again, we have-- oops. We have input coming in or an output coming out to their respective terminals.

So what is happening here is that I would like to have an input-- say input zero wants to travel all the way over to say, the output that is present over here. So let me label these. So we have the output zero, input one, output one, input two and output two, input three, and output four. So well, I can definitely reach every single output from any input so that's great.

So this looks like something that you are familiar with, right? It's just a tree. It's a directed graph, but these edges go in both directions, right? So I have an edge that goes from here to here and back from here to here. So this is the kind of layout that you could try out first to see whether this type of network would lead to good performance. So let's have a look at the different parameters and see how well this behaves. So here, we have a few parameters that we will be talking about.

So first of all, let's talk about the latency in this particular network. So how are we going to measure this? Well, we're going to look at this graph and we're going to measure it by the number of wires that you need to go through from an input to an output. So let me write this down. So the latency is the time that is required for a packet to travel from an input to an output. And how are we going to measure this? Well, we're just going to measure this by the number of wires that we need to go through.

So this you have seen before. We can measure this by the diameter of that particular graph. So here, we will define it for a network. So the diameter of a network is going to be the length

of the shortest path between the input and output that are furthest apart. So let's have a look at the graph above.

So for example, we can clearly see that, for example, input and output-- so say, input zero and output one are connected by just going up one step over here, but just going up from here to here. Then, this switch forwards the packet to this switch. This switch reroutes it, forwards it over here, and then it goes back to the output, output one. So for example, this particular path only has 1, 2, 3, 4 edges.

And what we are interested in is sort of the worst-case time that it requires to go from an input to an output. So that means that we are interested in a diameter. And a diameter is in this case, well, the shortest path that you can find from an input to an output that are furthest apart. So what are those who are furthest apart?

Well, of course, you would like to go through here, right? So if I connect the input zero to say, output four, I will need to go all the way up through the route down to the output. And how many edges do we see here? 1, 2, 3, 4, 5, 6-- so in this example, we have a diameter that is equal to six.

And in general, if you are looking at n times n networks, what does it mean? n is the number of inputs and n is also the number of outputs. So in this case, we have a four times-- well, this is actually three over here-- we have four inputs and four outputs. So this particular example depicted on the board is a four times four network.

So if you generalize this for any size binary tree, say, an n times n network, then what's the diameter of such a general network? Well, if we have n inputs and n outputs, well, we have to go all the way up through towards the root and all the way down. So we actually count the length of a leaf to the root here twice. So in general, we have a diameter that looks like this. It's 2 times 1 plus the logarithm of n.

So in this lecture, we will have n is going to be a power of 2, just to make calculations simple. And the logarithm is always to the base two. So this is a diameter of a general binary tree.

And well, what are the other parameters? So that does not look too bad. It's logarithmic in answer. That sounds pretty good. What about the switch sizes? Well, how do I measure those? It's like the number of inputs that get into it and the number of outputs that get out. So in this case, I will have 1, 2 inputs that go into this switch and there are two outputs coming

out. So this is what we call a two times two switch. So this will be a two times two switch.

But if you look at this one, for example, we see one, two, three outgoing edges and three ingoing edges. So this is actually a three times three switch. And in a general binary tree, we will see that all these intermediate nodes over here, they are all three times three switches. So approximately half of the switches are actually three times three switches. So that's the switch size.

Now, you may say, well, why don't I use a larger-sized switch? That would help me a lot, right? If I could use, say, a four times four switch, then I would be able to have more inputs coming in, more outputs coming out, and I can actually maybe use a ternary tree rather than a binary tree. In a binary tree, every note at the level has two children, right? But we could design a tree that has at every level three children. So then, they can use four times four switches.

But if you do that, then the path from the leaf up to the root is getting shorter and the diameter gets smaller. So if I increase the switch size-- so rather than three times three, we look at four times four or five times five, six times six and so on-- then the diameter will actually reduce. So what about having a monster switch, like I have just one switch and I have my input zero all the way up to input n minus 1 and then I have my outputs on the other side?

Well, of course, the switch size is n times n but the diameter is nothing, right? The diameter is reduced to one. You can immediately go from an input to an output through the switch. But this, of course, conceals the problem. So what we are interested in is, well, we're actually really interested in how to solve the problem of routing all these inputs to these outputs using smaller switches of size three times three or two times two.

What we're really interested in is, what is the internal structure in this monster switch? I sort of have concealed the problem by just saying, oh, I've got a big switch. But what we want to solve today is how do we do the routing in this case within the monster switch? So we want to use just small switch sizes and build up a network using these smaller ones, like three times three switches or two times two switches.

Now, so that brings us to yet another parameter, because here, we'd like to count the number or smaller switches that we use and that relates to the cost of the network, the amount of hardware that you need to put into it. So in this example, we have the switch count. Well, it's pretty simple, right? It's 1, 2, 3, 4, 5, 6, 7-- we have seven switches.

And in general, if we have n inputs-- so 1, 2, 3, 4 inputs-- then the number of switches that we use in the binary tree is 2 times the number of inputs minus 1. So let's write that down. So over here, we would have 2 times n minus 1, which is the number of switches that you actually use.

So how can you see that actually? So in general, we have 1 plus 2 plus 4 plus 8 and so on plus n. And it's a power of 2, according to our assumptions. And if you add them all up, I think you'll-- well, you can check for yourself that this is actually equal to 2 times n minus 1.

So now, we have the switches. So so far, this looks pretty good, actually. We use small switch sizes. The number of switches is linear in n. The diameter is logarithmic in n so that sounds good.

So what about congestion? Do you any idea-- what's the problem with this graph? What is the big, big problem here? What can happen in a very sort of worst-case scenario where the packets get routed from inputs to the outputs? If they need to go to certain locations, then they all may have to travel through the root. So you get congestion over here. We don't like that. So this root is actually then overloaded.

Actually, you can already see that say, this particular switch-- if this switch fails, then actually, we will have two disjoint trees that cannot even communicate to one another. So this brings us to the idea of congestion. And in order to define it better, you will need a few definitions.

So to start, we will define a permutation and we will use this to stipulate the requirement that we want on how inputs and outputs are related to another, which input needs to communicate to which output. So permutation is a function pi from the set 0 to n minus 1 to the same set. And it is such that no two numbers are mapped to more than once. So no two numbers are mapped to the same value. So what we really want-- to put it in mathematics, we want that pi of i is only equal to pi of j if and only if i is equal to j.

So let's have an example to plug into that picture over there. So a first example could be pi of i equals, say, n minus 1 minus i. This is a proper permutation. No two numbers map to the same value. Another one could be the identity permutation, like you map i to the same i. So that's another example.

Now, how do we use permutations to go towards the idea of congestion? So permutation can be used to formulate the permutation routing problem. And the permutation routing problem is defined like this. It's defined as follows. What we want is that for each i, we want to direct the

packet at input i to output pi of i. So you want to do that for all i.

So let's have a look at this particular example, where we look at identity permutation. So if you do that, we can easily route this, right? So I want to send a packet from input zero to output zero. So I can simply go into this direction. I just go towards this switch and it gets routed back to this one. I can go like this and this one can go like this and this one goes like that.

Now, if you look at the other permutation, the picture looks very different. Now, we want to route input zero to output three. In order to do this, I will actually need to go all the way through here and then all the way down to this particular output. And now, the picture gets into a big mess because for input one, we have to go to output two. So for input one, well, we go all the way like this, we again go through the root, and then we go down to this particular output.

And as you can see, for input two, well, we need to connect to output one. So again, we go all the way up and we go all the way down. And for this one, we will again go all the way up and all the way down to input zero. So now, you can see that this particular switch over here has to serve packets from all the inputs. All the four packets have to travel through this particular node here. So this leads us to the following definition of congestion.

So the congestion-- oh, before we continue, let me first define a path. So for i, we direct a packet at input i to output pi of i. And the path that corresponds to this route is actually denoted as follows. So the path taken is denoted by P i pi i. So now, we can define the congestion of a set of such paths. So the congestion of the path corresponding to P zero to P pi zero and so on and we go all the way up to the n minus 1 input that needs to be mapped to pi of n minus 1.

So the congestion is now defined as the largest number of paths that pass through a single switch. So in our example, we saw that in the case of the blue arrows here for the identity permutation, well, this switch only needs to transmit one packet and all those actually zero packets. So actually, the congestion here is equal to 1. And for this particular permutation, well, we had to direct all the packets through the root and it's the most accessed switch. And that switch has congestion four, right? So the congestion over here is equal to 4.

Now, this does not look so good because for a binary tree, we always have this vulnerable root that is right here in the center connecting the left side to the right side. So we can always find a permutation-- actually, this permutation over here-- that leads to this worst-case congestion.

So what we're interested in is the maximum congestion, which is sort of the worst-case scenario. And we'll define it as follows.

The maximum congestion is actually equal to the maximum over all permutations pi. So this is kind of the worst-case routing problem that I can imagine and it may occur in practice. So in the worst case, how can I solve it the best? So I want to find the minimum of the congestion of a path over here and the minimum is over these types of paths. So actually, this is our solution to this routing problem. We want to find the best kind of solution for this worst-case scenario-- so the minimum over all solutions for these paths

So well, for this particular tree structure, this permutation is really the worst-case scenario that you can have because every packet needs to be routed through the center over here. And it means that our maximum congestion for an arbitrary tree is actually equal to n. So that looks really bad, actually. So we don't like this at all.

So let's find out where we can do a little bit better and we come to look at the two-dimensional array and see what that would lead up to. And its structure is as follows. We essentially have inputs on the left and the outputs are on the bottom and they are in a grid structure. So we have input zero, input one, input two, input three. They all connect to their terminals. We have switches, four of those, and they are all connected in this grid.

And at the very bottom, we will have the outputs, the output terminals. So this is output zero and here, we will have output one, output two, and output three. So notice that my circle start to resemble my squares, but these are all the switches right here in the center.

So how does this work? Well, do we have a better parameter? So let's look at it together. So we need to first of all figure out what the diameter is. So what's the diameter of this particular network? So what's the shortest path between the furthest input and output? So if you look at that, we can see that if I go all the way from here and I go all the way down to this corner, that looks like the largest path and I need to cross all these wires. And in general, for any n, we will have that the diameter is 2 times n.

Now, what about the switch size? It looks a little bit smaller, right? Because over here, we had three inputs coming in and three outputs coming out but over here, we see that every single switch is only two inputs and two outputs. So that makes the size two times two. Now, the number of switches is pretty bad, right, because we have n squared switches. So that's really horrible. That's a lot. We would like to do much better.

And what about the congestion? Do you have any idea what the congestion could be in this particular case? We will prove a theorem on that. For any permutation, is there a way to route the inputs to the outputs in such a way that the switches get almost not congested? So in the binary tree, we had a congestion of n, which is linear in the switches.

But over here, we can do much better. We will show that the congestion of an n-input array is actually equal to 2. So that's great. So I'll prove it in a moment, but that looks really fantastic. And so it's way better than the binary tree.

Now, this is really not so good and this is also much larger, but still-- we will start to think next after we show this particular property how to combine these two and see how we can come up with another network that's able to combine in some ways these two properties. And maybe we can find a good solution that way. It turns out we will not immediately be able to do that. We will need to make another step and come to the last network. It really has good parameters.

So what about the theorem? So if you prove this, well, how do we start? You just start with any permutation. If I want to prove something about the congestion, it's defined as the maximum of all permutations. So let's take one of them and see what we can prove.

So let us define the paths for this permutation. So what we really want to do is we take any permutation and we want to find a really good solution for the routing. If that gives us a very low congestion, we are very happy. So the way to do this is well, maybe you have an idea already. So how would I route this? So I want to connect an input i, say, 1, 2, output two, for example. How can I do this? Any suggestions? So of course, I could go any path, but somehow, I want to have some uniform structure that hopefully helps me to prove that the congestion in every switch is very small.

So how could I think about this? Well, if I make sure that, say, a packet that goes from one to output two is only going to be participating in the wires off the i-th throw and the P-i-th column, then I know that every wire will only get traveled over twice by a packet. This could either be a packet that goes into this direction or-- so a switch will be accessed at most twice. A switch can either receive a packet from this direction or receive a packet from the upper part. So that will be a really good idea. So let's define that.

So we say that in our solution, we will design it such that the path from input i is actually going

to be rightward to column pi i and then downward to the output-- so downward to output by i. So this is a really good solution to the routing problem because now, we can continue our proof as follows.

We just say, well, if you look at the switch in row i and column pi i, well, this one actually transmits at most two packets because a packet can only come from the left or it's going to go from the top. So either one of the two-- at most, those two packets will go through the switch. So this shows that we have a congestion of at most two for any permutation.

And in order to prove equality, because that's really what the theorem says, we also have to show that there exists a permutation that achieves a congestion of two. And that is pretty straightforward. We can, for example use a specific permutation that maps zero to zero and maps n minus 1 to n minus 1.

Well, for this particular permutation, when we look at the picture over here, we see that input zero needs to go to output zero. We also see that this lowest input, input three, needs to travel all the way up to here. But it's clear that the packet that needs to go over here needs to travel through that switch in the lower left bottom corner. And the input three also needs to travel through that. So here, we clearly see that we you have a congestion of two.

So now, the proof is complete because we have shown this upper bound. So for any permutation, the congestion is at most two and we see that this specific permutation achieves this congestion. So this is the end of this proof. So that's great. So now, what we'd like to do is we'd like to combine these two networks and see what we can learn from both. So now, we'll be taking out a lot of chalk over here.

So the idea is to construct a butterfly network and I will draw it in such a way that you can see the recursive structure. The idea is to do the following thing. So let me see how I can do this the best. So I will just do the top line first and I have the spacing. So we have input zero, a terminal, we have a switch, we have a switch, we have a switch, and another one, and here, we have the output zero.

So the whole idea is that I'm going to combine every two outputs by using a small butterfly structure. So we have two, output three, output four-- actually, I need a little bit more space. Do it once more, output one, two, three, four, five, six, and a last one, seven. This is going to be pretty tight on the board.

So what's happening is this. So these are all connected, of course, to switches. The switches output those. And the idea is that we create the following structure. This switch can either forward it over here or it can cross it over to this particular line. And this switch can either forward it or cross it over to this line. So this is a very small butterfly structure. Here, we have two inputs and two outputs. And we will repeat this process and we'll do the same on each of these other levels. So we forward those or we cross them, like this.

And now that we have constructed all these smaller butterfly structures, we can start to combine two butterfly structures together in the bigger one. So here, we had two outputs that we combined in a butterfly structure. Now, we use two butterfly structures that we put into a bigger version. So how do we do this? Well, we have that the upper half over here can either forward those packets or cross them over to the bottom part butterfly structure.

So for these, we can either forward them straight on or we can go to the top butterfly. So you see that these two inputs, these two switches, either can forward packets to this sub-butterfly network or to the top butterfly network. Now, we'll continue this process and for these, you'll do the same. So we can either go straight or we go down. And over here, we can go straight or we can go to the top butterfly network.

Well, now we have the final part where we combine essentially these two butterfly networks. We have two butterfly networks created here now composed again of smaller ones and these two are being composed to this bigger butterfly network. Again, we take these four switches. They can route their packets forward to the top butterfly sub-network or to the bottom one.

So they can either go straight ahead or this one can connect to the first over here, this one to the second, to the third, and this to the fourth. And in the same style, these can forward them straight like this and then go up like this. And these are all connected because in this example, let's just have an eight by eight network, butterfly network. We have input zero to seven.

So this is the butterfly network. In a way, what you can see here is you can see sort of the two-dimensional structure, like we have rows and columns. At the same time, we can also see this binary sort of tree feeling we get from it, which is that a switch can forward sort of its packets to either, say, the top butterfly or the bottom butterfly. So there's a split in two. The same for this one, right? This one goes either to this butterfly network or it goes to this butterfly network. So you have this tree structure sort of embedded in this two-dimensional structure.

So what are the properties of this one? So let me first define in more formal mathematics how

the switches route their packets, so how the connections are. So in order to do that, we are going to label each switch. And the idea is that we're going to label it by its row and by its column.

So we will have-- the columns are numbered by level zero, level one, level two, level three, yes? And the rows are these integers, but we are going to represent them by binary numbers. So zero would be 000, 001, 010, 011-- oops-- 100, 101, and then we got 110 and 111.

So for example, this particular switch would be labeled by these three bits, 001, and the integer number, 1. This one would be 011 and its column is indexed by integer 2. So a switch is uniquely identified by its row and column. We will have b1 up to b logarithm of n, which are the number of bits to represent the row in digits, and to finally have an integer l and this we will call the level.

So this particular switch either directs or routes a packet to the switch that is indexed by b one up to-- and then we get b, l plus 1 and we take its complement. So instead of if b, l plus n would be 1, we would have a 0 here. If it would be a 0, we will have a 1 over here. But we repeat all the other bits and we get to b log n. And it routes us back to the next level. So we will have l plus 1.

Another possibility because there are two outgoing edges is if we have just b1 and we just copy b, l plus 1, essentially. We route a packet straightforward. We don't do anything special. We get b log n over here and then to the next level. So for example, let's see where we can see how this works.

So for example, take this particular switch. We have 010. So it can either go straight on to the next level. It would go to 010 but then instead of level one, we have level two, which is the right edge over there. The other one is if this one goes up, well, we will need to switch the first bit over here, a 1. We swap it into 0 and then we go to the three zeros over here and we go to the next level and that would be this particular rule.

So what we can do here is to-- so when we see this, we can start to figure out how we can direct inputs to outputs. So let's do this. So suppose I want to route a packet from a certain input, one of these, all the way to one of the outputs over here. So the way to do this is as follows.

We can just start-- for example, I want to go from switch x1 up to x log n comma 0. So I start

completely at the left over here and I want to go somewhere of my choice to the right. So I want to somehow move all the way to some other row, y1 indexed by y by the bit pattern, y1 up to y log n, but now at the very last level, which is log n.

Well, how do I do it? Well, this switch, I can use that rule up there and simply change x1 to y1. I can either leave x1 as it is if it's the same as y1 or I can swap it to its complement if that's the value of y1. So what I can do is I can just simply route it to y1. And then, I leave all the other bits the same, which are x2, x3, all the way up to x log n. And we will have reached the first level.

Now, this one can go to-- well, now I'm going to swap the second bit into the bit of my choice. So I leave all the other bits the same, y1 the same, x3, all the others the same. I just swap x2 into y2. So we leave all those equal and we go to the second level. And then, we go all the way to the final level and we one by one swap all these bits.

So let's have an example. Suppose I want to connect, let's say, this one to for example, well, let's say this particular output. So what's the binary for this one? This is actually 101. So if the first bit is different, I need to cross. And otherwise, I need to pass straight on. So let's do this. So over here, I'm in 011. I need to go to 101 so we need to change the zero into a one. So I need to go down. I need to cross.

Now, if I look at the second bit, I also need to change it to a zero so again, I need to cross, which is over here. Now, the third bit is equal to 1 and it's the same. So now, I can go straight ahead. I do not cross and I end up at this output. So what did I do? For every bit that is different, I cross and for the bits that are the same, I go straight ahead. So this is how I can route packets from one input to another output.

So let's look at the parameters. First of all, if you look at the diameter, well, it turns out that that's approximately equal to the number of levels, which is the logarithm of n. And to be precise, it's actually equal to 2 plus the logarithm of n. So that's great. That's a good scaling. Again, it's back to the logarithm of n. So we have the best of these two parameters.

The switches that we see have two inputs and two outputs. So we again have a two times two switch. The number of switches is the number of rows times the number of columns. The number of columns is the logarithm of n and number of rows is equal to n. And to make it a little bit, precise, it's 1 plus the logarithm of n. So that's somewhere in between those two. But if you're thinking about it, it's much better than n squared. It's almost linear except for a

logarithmic factor.

For the congestion-- and we are not going to talk about it here, but you have a problem set assignment that will ask you to solve this-- is that actually, the congestion is the square root of n or it's equal to the square root of n over 2, depending on whether n is an even power or n is an odd power. Now, we're not going to prove that here because we want to step forward to this particular network. It's very exciting. And you will prove this in your problem set.

So this one is somewhere in between, somewhere in between these two extremes. Now, it will be really fantastic if we can somehow transform this network with a trick to, again, have a really great congestion of just a constant, like two or three or whatever or maybe even one.

So for this particular network, in the 1960s, Benes, a Bell Labs researcher, had the great idea to use a butterfly network and attach to it, again, a butterfly network, back to back sort of.

So what was his idea? His idea was to do the following. So the butterfly network as we have it right now is this particular part over here. And the idea is now to start up mixing all those outputs that we got here together again using a similar rule.

So what do we do? We are going to essentially repeat this particular structure on this side. So how do we do it? Well, we go either straightforward or we start to mix them again. So it's like this output, this particular switch, can either go straight ahead or can cross to the lower part over here. It goes over here and this one goes over.

So as you can see, we have repeated this part. It's exactly the same as this structure over here. We'll do the same for this part. So we can either cross or we can go straight ahead. Oh, we also have, of course, that these switches can go straight ahead or can cross to the top. I forgot about that. So we have this-- oops-- as well. So as you can see, this particular structure repeats itself again and we slowly start to build up in mixing all the outputs again or the possibility, at least, to route them to any other row.

So how do we do this? Well, we continue this particular structure now over here. So all these can either go straight ahead. That's a possibility. Or they can go all down. So this switch can either go straight ahead or can go to the lower half. And for these, we have a similar structure. We can either go straight ahead or such a switch can cross over to the top over here.

So that's this. So this is Benes network and then over here, of course, we have the outputs,

zero, one, and all the way down to seven. So as you can see over here, the structure again has a recursive nature to it.

You can see that this big Benes network over here consists of two smaller ones that are right here in the middle, this one that goes all the way up to here-- so maybe I should put a color boundary around it. Let me check I want to do this-- right. So this particular part, is again a Benes network and the top part in the same picture, the top subnetwork is also a Benes network, this part.

And if you look within those, we again see a top part and a bottom part. And over here, we see a top part and also a bottom part. So you see this recursive nature again reappearing. It turns out that with this trick, we can completely eliminate congestion and we can get it to only one, which is really surprising. And that what we're going to prove here. So this is a great invention at the time. It's really, really beautiful.

So let me put in the other parameters. So they stay approximately the same up to that the diameter is about twice as large because we added another sort of whole butterfly structure to it. The switch size stays the same. We, again, have about two times more switches so they sort of stay about the same up to a linear factor, like a constant factor. And the congestion, however, completely dropped down to one.

So that's what we're going to prove now. And in order to get some intuition, well, let me first write down the theorem. Actually, let me put this over here. So in order to get some insight into this, we are going to use this recursive nature. So we're going to use induction and we're going to say, oh, for any permutation, I can find really good routing for say, this red subnetwork and for this blue subnetwork. So I know that.

So what I need to do is, if I have my bigger Benes network, like this one, I would need to somehow map these inputs-- I need to route them to either the top and the bottom subnetwork, one of the two, in such a way that there will be absolutely no congestion, because we want to keep this one. So a switch should only see one packet coming in. So that means, for example-- and we'll come back to that-- that for example, for this switch, it should not receive a packet from both this input and from this input.

So the intuition that we are going to create is we're going to list our constraints, the constraints that we need to satisfy, like the zero and the fourth input should not both be mapped to this top subnetwork and so on. So we will get into that and then we will gain a lot of intuition on

how to solve this.

So what's the theorem? So the theorem is that the congestion of the n-input Benes network is actually equal to 1. And we will prove this for n equal to a power of 2. We have assumed that at the start that we had with all the other networks, as well. And in this case, we will use induction on a. So that's the method that we will do because that's also the recursive structure of the Benes network itself. So we will use induction on a and we are going to define the induction hypothesis simply as, "The theorem is true for a."

Now, let us do the base case. We always start with the base case and that should be pretty easy because this is the most basic Benes network. So n equals 2 to the power of 1. We essentially have two inputs, an input zero and an input one. They are connected to these switches over here that can either forward them or can cross them over and then they go directly to the output.

Notice that in this case, we just have the most elementary butterfly network. It's the same. So we have output zero and output one. So this corresponds in this picture to these little small things over here, this one and this one and this one over here and the fourth one over here. So now, let's take any permutation. We want to show that we can route it in such a way that there's only a congestion of one. So let's do this.

So there are essentially only two permutations. Either zero is mapped to zero and one is mapped to one or zero is mapped to one and one is mapped to zero. So in both cases, we can just route them through their own switches. So we have that either pi of 0 equals 0 and pi of 1 equals 1, in which case we just direct them straight through and we go straight through and every switch only sees a packet once. So for this particular permutation, we have a congestion of one.

Now, the other permutation that we can have is if zero is mapped to one and if one is mapped to zero. Well, in that case, we just route this cross over to the bottom row and here we go from this switch to the top row. Again, every switch only sees a packet once. So in this case, in the base case, we are done. We are happy. We have shown that the congestion is equal to one.

So now, it gets to the harder part because for the inductive step, we are going to assume, of course, that it holds true for a smaller Benes network. So we assume that P a is true and well, let's try to gain some insight here. So we know from our induction hypothesis, within each subnetwork, we can solve any routing problem with congestion one and for this subnetwork,

the same. That's our induction hypothesis.

So how do we go ahead? We need to somehow map these inputs according to the permutation of our choice. So that could be for some input zero goes to output five or input one goes to output two, et cetera. So somehow, we need to choose where we are going to map this particular input to. So packet zero that comes from this input should either go to the red network or it should go to the blue network. And for each of these inputs, we can make such a choice. But we have to be very smart about it because we need to avoid any congestion.

So the intuition is that we're going to set up a constraint graph, a graph that represents all the constraints that we need to satisfy in order to achieve congestion of one. So let's do an example so that we can figure out what's going on. Actually, let me put it over here. So just take an example permutation and we'll go through this example and then see how the proof works.

So let's as an example have pi of zero maps to one, pi of one maps to five, pi of two goes to four, input three goes to seven, four maps to three, five to six, six to zero, and seven to two. So this is just an arbitrary permutation. So what do we see? We want to make sure that, for example, this switch is only seeing one packet. So it cannot see a packet both coming from input zero as well as from input four. I cannot see that. I do not want that to happen.

Similarly, for this one, I do not want to see a packet coming from one or one from five. So let me define a constraint graph that sort of represents this. So the constraint graph that we are interested in is defined as follows. If two packets must pass through different networks, subnetworks-- so in our case, the red and blue subnetwork-- then we'll actually have an edge between those two. So then, there is an edge between them. So for this example, we're going to set up this constraint graph.

So I was just talking about this particular switch. It cannot see one coming from four and a packet from zero. So what he have, we have an edge between zero and four. In the same way, we have an edge from one to five. Why? Because a packet that comes from input one and a packet that comes from input five cannot both be routed through the switch because then the switch would see two packets and then the congestion would not be one, but two, right?

So one and five also have an edge in between. And in the same way, we have two and six and seven and three. So two and six is this constraint, like two and six over here. And three and seven is the other constraint. So if I have those constraints in place, well then, I know that the routing that goes from level zero to level one will not violate my congestion of one. So that's great.

Then, I hope to be able to use the induction hypothesis and I get a proper routing within the red subnetwork and one within the blue network. And then, I need to map all these to these outputs. So I also have constraints on these outputs because, well, For example, take this particular switch. It should not see a packet coming from this particular one and one from this one. So how do I code that up?

So let me first write out what we did here and then we'll do the same for the last level over there. So-- oh no, that's not really necessary. So at the output side over here, we have similar constraints as we did over here. And in this particular example, just as an example, suppose we look at the packet that is destined for output zero.

Well, what is this packet? Well, I know that's pi of 6 is equal to 0, according to my example. So packet six is destined for this particular output zero over here and goes through this particular switch. So this packet and also the packet for output four, which is if you look at the mapping, pi of 2 is equal to 4. So that's packet number two. Well, both of these packets cannot pass through the same subnetwork.

So why is this? So let's look at this particular example. So output zero, well, comes from packet six, somewhere over there. Now suppose packet six was routed through the red network and at the same moment also, output four-- the packet that is destined for output four, which is packet number two-- suppose packet two was also going through the red network.

Well, then I notice that both of these packets must arrive at this particular switch in order for one to be routed to output zero and the other one to be routed to output four. So in order to avoid congestion in this particular switch over here, we need to have a constraint. The constraint says that the packet for packets two and six, that those two cannot go through the same subnetwork. So essentially have another edge over here-- we already had the constraint but it's just the same edge. So let's look at the other constraints that we have.

Well, let's look at a different example. So for example, if I look at this switch, well, if a packet goes through here that needs to end up at one and a packet that's goes to five, if those two

packets are routed through the same red subnetwork, they have to end up here in order to go to both here and to there. So we have congestion of two. So what are those packets? Well, what does pi map to to one and five?

Let's look over here. We see that pi 0 is equal to 1 and pi 1 is equal to 5. So packets zero and one are actually mapped to output one and five and they should not go both through the same subnetwork. So we have another edge over here. And now, we can continue this and we have five and seven. So just have a look over there. See, five and seven, they map to the outputs two and six. Again, we have two and six. If they are both mapped to the same network, this one, for example, then I will have a problem. So the other edge is over here.

So what did we do here? We started to write out the constraints on this side and we wrote out the constraints on this side. So I only looked at the red subnetwork. That's what I realize now. I could also have looked at the blue network. So let's do that also just to make the picture complete. So for example, let's look at this particular example. The packet six and two should not both be routed through the blue network because then they would both have to go through this switch, one going up to output zero and one going to the right to output four.

So in order to avoid congestion at all costs, we have this constraint graph. So now, we come to the key insight. And the key insight is to use a two-coloring of this graph. So the key insight is a two-coloring of the constraint graph, which will lead to a best solution for the routing problem. So let's do this.

So we will color this one blue. As you can see, this is an even cycle, blue, red, blue, red, and blue and red. We will make this one blue and this one red. Well, it turns out that we can now start our routing process. So for example, actually, I will draw a new graph to make that really clear. So I have my blue and my red chalk over here to demonstrate what I mean.

So what do I do? I have zero, one, two, three, four, five, six, and seven. I have the switches that correspond to those. Well, if it's colored red-- so zero over here is colored red-- I will direct it to the red subnetwork. So where is this red subnetwork? It's really contained over here and the blue one-- so this is the red one and the blue one is right here. And over here, we have the outputs ranging from zero, one, two, all the way to seven.

So input zero is colored red. We go straight ahead. We want to go to the red network. Input one is colored blue. It goes, therefore, to the blue network. So this is the only way how to do it.

Input two is colored red. Go straight ahead. Input three is also colored red. Go straight ahead. Input five-- oh, input four is colored blue-- goes to the blue network. Input five goes up to the red network and input six goes straight ahead to the blue network. It's colored blue and input seven is also colored blue.

Let's look at the outputs. So for example, well, let's have a look at output zero. so output zero-- which packet is mapped to output zero? It's packet number six. So six was mapped into the blue network and then it needs to be mapped to output zero. So there's only one edge that goes from the blue network to output zero, which is this particular one. And then somehow, this one needs to be mapped to this one over here.

Now, we can continue like this. Output one should receives a packet from-- let's look at the permutation-- from five. No, sorry, output one-- pi of 0 is equal to 1 so packet zero needs to go to this particular output. Now, packet zero is in the red network so there's only one edge that goes from the red network to this output. So we need to have a connection over here.

Now, we can continue this and note and demonstrate-- and you can test it for yourself, too-- that output four needs to receive a packet from the red network. Actually, it should be this particular one, which happens to be packet number two. And then, we have this one, right?

So let me just finish it. We have this and we have these two and we have this one. We have this one and we have this one. This one goes straight ahead. This one goes all the way up and this one goes all the way up.

So what do we see? We see that packets over here, that these switches only see a packet once and these ones, as well, these ones also and these ones also. So we have directed the packets, routed the packets to the red and the blue subnetworks in such a way that the congestion at the last level and at the first level is still equal to one.

Now, we use our induction hypothesis and we conclude that we can map the route that's going to have a routing from packets from here to here such that the congestion within the subnetworks is only one, so within the blue as well as in the red. So this is the insight into how this works and I notice I am running out of time. So the formal proof we will have to postpone until recitation, but that's actually really a very simple thing to do that right now.

So just keep this key insight and then you can easily prove the theorem. But this is the real insight. Thank you.