MITOCW | 19. Network routing (without failures)

HARI BALAKRISHNAN: OK. So today, we're going to continue talking about multi-hop networks. And in particular, I'm going to talk about a fairly fundamental problem in multi-hop networks called network routing. And that's the topic for today and through most of, half of, next week.

And today, this week, we'll talk about network routing in networks where there are no failures. So in real networks, of course, things fail. Packets get dropped. Nodes may fail. Switches may fail. Links may fail.

We'll just worry about the case when there are no failures for this week. Next week, we'll add some more complexity and talk about how we deal with failures. So the abstract problem is pretty simple. You have a set of nodes in the network. And there's some network topology.

And every source in the network-- you have sources, and you have destinations, nodes in the network. And they wish to send packets to each other. And we're going to decompose this problem, these endpoints, which actually want to communicate with each other, as we've already talked about, do so by sending packets via switches.

And the problem we're going to worry about is what happens inside the network in these switches. The switches solve the following problem. If things work correctly and if things work well, when a switch receives a packet with a destination address specifying that that packet has to be sent to a particular named destination to the network, the switch figures out how to ship that packet.

It figures out whether to send it along-- I need a better-- here. This switch, for example, figures out whether a package should go along that link, or this link, or this link, or this link, or that link, given any package that it receives. And every switch in the network performs this task.

Now, when a switch gets a packet and the packet has a destination address in it, what the switch does is some sort of a lookup. It looks up that destination in some sort of a table. And that table maintains information about which of the links to use in sending that packet toward the destination.

And this step is called forwarding. Packets are forwarded by switches. Technically, packets aren't routed. Everybody says packet routing, and I say it, too. But it's important to realize packets aren't routed. Packets are forwarded.

The forwarding process is pretty straightforward at slow speeds. You get a packet. You take the destination. You look up that packet in a table. And that table tells you what you do with the packet, which link to use.

Routing is the problem that the switches solve of constructing these tables. So forwarding is simply lookup in a table. And the table is called the routing table.

Sometimes it's also called the forwarding table. And there's a technical difference between them that's not that important for us. So forwarding is the process of looking up the destination in the routing table.

Routing is the distributed process, the algorithm that's used or the protocol that's used, to build up these routing tables. So it's the construction process, how to construct these routing tables at each switch in the network. That's done in the background.

That's usually done in software. Switches at high speeds involve a fair amount of hardware. Routing itself is usually done in software.

And so a typical switch has a lot of different processing elements in it. A lot of those processing elements are high speed hardware that deals with the process of forwarding. And then you have software sitting on the side where all of the complexity is.

And all that complexity deals with all sorts of complicated rules that you have to come up with in order to decide how to construct these routing tables. And we're going to talk about the world's simplest networks today. And you'll find that even that's reasonably sophisticated and complicated.

So just for concreteness, an example of a picture of a topology is like this. And for reasons that will become apparent as we talk this through, we'll model the network as a set of nodes and a set of links. And so far, there's nothing new about this.

But we'll model links as having a cost. And this cost might reflect, for example, the delay or latency to send a packet along that link. The cost might reflect the real dollar cost of shipping data along links.

Internet service providers might charge different amounts of money for different types of data, for example. Or the cost might just reflect your own internal preferences as to which links you might prefer based on whatever concerns. Maybe some links are slow, and some links are fast. Maybe there are higher speed links, lower speed links, that cost more or less, et cetera.

So these costs are abstract numbers. And we'll just assume that we're interested in finding minimum cost paths between senders and receivers. So every switch solves the problem of finding the minimum cost path to a destination where the cost along a path is simply the sum of the costs along the links on that path.

This is just standard shortest path routing. We'll use shortest path even though we don't mean literally the shortest number of hops, but the minimum cost path. And where that distinction is important between the number of hops and the cost, et cetera, we'll clarify.

So the routing table looks something like this. In fact, it looks like this. There's destination. Every switch has this.

So this is an example of a routing table at node B. Node B maintains all the destinations in the network. And we're dealing with small networks so far in our class. So we'll just assume that every node in the network has a unique name or a unique address, and that the routing tables contain an entry for every destination in the network.

So the routing table has three columns. It's a database with three columns. And every switch in the network or a node-- I'm going to use the word node and switch interchangeably. Every switch or node in the network has one of these if the network is working correctly.

So if the routing protocol does its job, every node comes up with its own version of this table. There is a destination. There is the link you need to use that is the next hop, if you get a packet to this destination, what link would you use, and a cost.

So in this table, if node B or switch B received a packet destined for destination A, it would use link L1. Each link is named locally. So B would have its own L0, L2, and L1.

You know, your own computer has a bunch of links. I don't know what the command is in Windows, but in all other sensible platforms, if you do IF Config, you can get it. I think it's called IP config on a Windows. But you can actually see a list of links. So switches have many, many, many links.

And so you'll find, for example, for destination A you use link L1. The cost is 18. In fact, you'll see here that, when B receives a packet for A, it doesn't use the direct link because the direct link has a cost of 19, whereas going through L1, this link has a cost of 11. And if B believes that the path toward A along this link has a cost of 11 plus 7 is 18, that's what it thinks. And, therefore, it would use that link.

OK? It's very simple. Now, routing is the process by which these different nodes talk to each other and build up these tables. When I say a route to a destination, I mean this.

I mean the route to a destination at a switch or at a node is the link that that destination would use to send packets to the destination. This is important technically for us. Because, otherwise, we will get all tangled up getting confused between routes and paths.

OK. For the purposes of our discussion, a route is the next hop link, is the link that you're going to use to get to the destination. The path is a sequence of links. So I don't want people telling me that the route from B to E is whatever, is BCE.

I'd like people to tell me that the route at node B to destination E is L1. Or if you wish to be clear, you can say, it's link L1 which takes it to next hop C. The path from B to E might BCE, OK?

All right, so that's what the routing table structure looks like. And we're interested in minimum cost paths to go from places to places. So normally, traditionally, we sit around and try to now dive into protocols.

But I think here what we're going to do is since our notes are all so nice and written up, I actually don't have to tell you everything that's there. So what we're actually going to do is something we call the routing game, which is an experiment in social networking. What it means is that each of you or some of you will act as nodes and start computing routes to different destinations.

OK, so that's what this game is going to be. So what I have in my hand is 40 slips of paper. And I hope there are 40 people here, 40 slips of paper. And each of these slips of paper has some information that looks like this.

It says you are node X, and you're connected to nodes Y, Z, W, et cetera, OK? And there's a set of rules that I will go through in a minute. First of all, these were all in perfect order. So I need to shuffle them a few times carefully without losing them. I think Perci Diaconis, the famous probabilist, said that you've got to do this seven times. Maybe this is close to seven.

OK. So what I'm going to do is I'm going to pass this around. There's no bag here. The bag's cumbersome. So what I'd like you to do-- actually, I'll put it in the envelope and pass it around.

Just kind of close your eyes or something. Don't look at the number. And just pick one up. And don't look at it until I tell you, OK?

So we'll start here and just pass it around. I'd like 40 different people to have them. You don't have to feel compelled to take it, but just pass it around. And try to do it quickly because I'd like that to happen faster than the time it's going to take for us to compute these routes.

So why don't you just pick one sheet of paper up and just pass it around, please? OK, so I picked 7 minutes. It sometimes works a little faster than 7. The best that the class has done is about 5 minutes.

Last time, they did it very quickly, but they got it wrong. So I don't think that counts. But I've been told that they've tried this experiment at Berkeley.

It's taken usually more than 7 minutes. And I figure MIT students are smarter. So let's see if you can do it faster than 7 minutes.

Now, your job is to find a path from a source node to a destination node as quickly as possible, OK? And I mean, as a bonus, if you actually find the path with the minimum number of hops, that's even better, but we'll take any path. Now, there's some rules here.

Now, there are some rules here. So you may not actually kind of get up and run around and try to do things that a normal network switch wouldn't do. So you're allowed to stay in one place. And what you're actually allowed to do is you're not allowed to pass your sheet of paper to other people because you have information about yourself. And that piece tells you who your neighbors are, the numbers of your neighbors.

So you can't actually kind of send your piece of paper to other nodes. That's not allowed. And don't let people copy what's on your sheet of paper.

Now, here's some things you can do. You can read them. Ask your friends for advice. You can shout to other participants.

We're allowing you to yell and scream, but do so in a way that's somewhat civilized. And you can wish that you didn't pick up a slip or pick up a slip or whatever. But try to act generally-- I mean, this class is recorded. So you know, what you say might be heard.

Now, if you get a slip, there's some ground rules. You can't cheat. We're not dealing with security here where a node that's 17 and connected to 27 and 29 tells people that it's 14. I mean, you don't want that.

It's hard enough to do when you tell the truth. So don't cheat. There's probably a variant of this we can come up with where some fraction of the nodes are adversarial and one can't see if this stuff even works.

But right now, don't cheat. If you've got a slip, you kind of have to really try to participate in whatever protocol you come up with. And this experiment has no human subjects approval.

OK. So who has the envelope? Is it empty? Oh my goodness, it's 7 minutes already. Let's move it around.

OK. When the envelope gets empty, we can start this thing. So we should try to move it along. Is everyone clear on the rules?

What you're trying to do is to find a path between the source node, which was numbered 1, and the destination node that was numbered 40. The source and the destination know who they each are. And then we're just going to wait.

This is the easiest lecture to prepare for because I just have to keep quiet for a few minutes. OK. And if things don't work out or things don't work, whatever it is, I'm assuming that you'll come up with some variant of a reasonable routing protocol. And odds are, it'll be a variant of one of the ones we're going to study.

You know, the gentleman here who suggested something was a reasonable idea, which is you go and you pick one neighbor and you go through. But you get stuck in a loop, and then the idea is you come back to where you start. And then you go through the next. And that principle could work, but you got to remember a lot of stuff and tends to make mistakes.

And in fact, this was an easy network where every node had one, two, or three neighbors. I mean, there's a few with more, but most of them had a small number of neighbors. What you guys were going after was a sort of better plan, which everybody yells out their neighbors. And then they yell out their neighbors and their neighbors and so forth.

And that particular protocol has a nice name to it. It's called link-state routing, where you broadcast and flood your own neighbor information, OK? That's the second of the two protocols we're going to study. What you were trying to go after was link-state routing.

Now, the first protocol we're going to study has a different name to it. It's called distance vector routing. And these are the two routing protocols we're going to study.

Almost all routing protocols in practice are variants of either a vector protocol or one of these link-state protocols. There's lots of variance. And there's hundreds of routing protocols, but these cover the main concepts.

So let me tell you how distance vector protocols or how these vector protocols work. They're a little different from the kind of approach you were going after in solving this problem. Had you gone after an approach where you started at the destination, at 40-- and 40 simply said, I'm 40.

And then the neighbors next to 40 then said, I'm connected 40. To get to 40, come through me. And the cost is, let's say, whatever the cost of their link to 40 is, right?

Then, now, initially, nodes only knew about themselves. But, now, you have a destination. And you have a set of nodes going to get to the destination.

Let's call these nodes-- let's just say n1, n2, and n3. Initially, the only thing that D says is I am D. And it says that to its neighbors along these links.

This saying has a name to it. It's an advertisement. Now, when anyone hears about destination D, it can do the same thing to its neighbors.

So let's say these are n4 and n5 here. What n1 can do is to say that, to get to D, let's imagine that this cost here is 6. So D says, I'm D. And the cost to get to me is 0 because I'm D.

n1 here's that. And it says, I'm n1. That's true. But to get to D, come through me and the cost is 6.

So it now puts out advertisements along these links. But it says, D:6, D:6, D:6, where 6 is the cost of that link. Now, let's imagine that these other links have cost.

So let's say this link has a cost of 8. This link has a cost of 2. Let's say this link has a cost of 9.

Sorry, I should draw this better. Let's put a 9 here. So the link cost is 8. The link cost is 9. And the link cost is 2. And let's say for a minute here that this cost of this link is 10.

Now, n1 advertises to everybody saying, to get to D, the cost is 6. To get to D, the cost to 6. And to get to D, the cost of 6.

Each of those guys, when they get this information, now know that they can get a route to destination D. Because now n4 knows that, if it used this link to get to D, the cost would be 6 plus the cost of that link. So it's 14.

So it would have a routing table entry that says D is cost 14. And of course, its routing table entry would have an entry saying this is the link that it should use. Similarly, n5 here would take destination D and say that the cost to destination D is 9 plus the advertisement, which was 6. And so it would say D is 15.

And n2, which previously had it a route to D, which was this link at cost 10, would look at this new advertisement coming in here saying that the cost on this from n1 is 6. Add that cost to the cost of the link, which is 2, and have now a different way to get to D whose cost is 8. And compare that cost against the cost it previously held to destination D. And because we're interested in minimum cost routing, 8 is smaller than 10.

So n2, which previously had a cost to destination D of 10, would throw that route out and replace it with a route to destination D going along this link with the cost of, now, 8, which is smaller than 10. And this process just continues through as it goes along. So you might end up in a situation quite easily, as we just went through here, where, let's say, n3 were connected to n5.

If this link had a cost of 1 and this link had a cost of 4, what would eventually happen is that n5, which ended up with a cost of 15 to go like that, would, when it hears an advertisement from this node, replace that route with a cost of 4 and use this link as its route to get to destination D. And this process just continues until everybody has initially some route to D. And then if the process continues a little bit longer, everybody will have a minimum cost route to D.

This step, where nodes evaluate an advertisement that they hear about a destination against their current route and the current cost to the destination, and replace it if they end up finding a route with smaller cost or an advertisement with smaller cost, when the cost you have to compare is the cost of the advertisement plus the cost of the link along which that advertisement came and you compare that cost against the cost of the route you already hold, if it's smaller, you replace it.

That algorithm is called the Bellman-Ford for algorithm. And you might have seen centralized implementations of shortest path routing using this algorithm. And if you have, it actually turns out it's a little less efficient than another algorithm, another one we'll the study called Dijkstra, if you're doing it centralized.

But it's very, very elegant elegant for distributed computation because the routes to different destinations are being computed in a completely distributed way. These nodes far away here have no idea what the network topology looks like. They couldn't even reconstruct the network topology.

The best they could do is to find their own way to get their linked to use. The only information they have is what they hear from their neighbors. But, yet, they're able to find an answer because all they have to do is to listen to all their neighbors and, among the set of neighbors, pick that neighbor who's advertised cost plus the cost of the link to that neighbor is minimum across all of the neighbors.

So the computation that's being done is a very simple computation. It's a very elegant computation, which is the min over all of the neighbors of the link cost LIJ plus the advertised cost from J where the minimum is done over that all J, where J is the set of neighbors of I.

So you minimize over the set of the neighbors. Each node I does this. You minimize over the set of neighbors J of I. The link cost from I to J plus the advertised cost to destination D of J.

And you take the minimum cost. That's the minimum cost. And then you take the link corresponding to that neighbor. And that's the route that you use.

Now, this algorithm gets more complicated and tricky to argue that it's correct when there are failures. But today, there are no failures. There's another wrinkle in the algorithm, which is I mentioned that what are the conditions under which a node changes its route to a destination.

So I went through one rule. I mentioned a rule that said, if the current route to the destination does not exist, in which case the cost is assumed to be infinity, or if the current cost to the destination is smaller than the cost of the advertisement plus the cost of the link along which the advertisement came, then you replace the route to the destination. But there's actually one other condition under which you should replace that route to the destination. Yeah.

AUDIENCE: [INAUDIBLE]

HARI BALAKRISHNAN: If it's equal, well, technically, you don't have to replace it. You still have a good path, right? You could replace it, but it doesn't matter.

There might be a case where you have to replace the link, the route, when in fact the cost increases. There might be a case where you have a current cost to the destination and some current route to the destination.

And you hear an advertisement, and you take that advertisement. And you add the link cost, and you find a bigger number. And you might sometimes have to replace it. Yes.

AUDIENCE: [INAUDIBLE]

HARI BALAKRISHNAN: Right. It could be that what's going on here is that the cost to the destination, a link-- I previously told you that my cost to the destination is 17. And, now, I've changed my mind. I tell you that it's actually 19. And I could change my mind for a variety of reasons usually having to do with failure.

So I guess this is a little bit of a cheating question because I told you there's no failures. But it could be that perhaps the cost of a link changed because it became more expensive or something like that. And that's the only sort of case you have to worry about where a cost of the advertisement could increase.

And if that cost increases along your current route, like you think that the route to the destination is 17, the cost is 17, but in fact, it turns out to be 24. That's the time when you have to change your entry in the routing table. You have to change the cost associated with it.

But, otherwise, it's basically that's the algorithm. And it's summarized over in this chart here. And I've gone through pretty much all of it. Does anyone have any questions?

No questions? How long does it take before every node in the network-- actually, before I get to that, the reason it's called a vector protocol is I showed you a picture for one destination. But in fact, each switch or each node does it for all destinations.

So the general form of a distance vector advertisement looks like this. It has a destination, destination 1 colon cost 1, destination 2 colon cost 2, destination 3 colon cost 3, and so forth for all the destinations to which you have a cost. And initially when you start, you don't know about any of the other nodes.

If you have a route to some destination in general, then there's some cost associated with it. If you know about a destination but have no route to it, the cost is infinity. It'll turn out next week we'll find that the value of infinity in this network has to be pretty small, but that's because this algorithm is not the world's best algorithm for big networks.

And I'll explain why infinity has to be a small number, but theoretically we can assume it's infinite. The reason it's called a vector protocol is because the advertisements are a vector of destination cost tuples. And so you send these tuples around. And this is a vector of these tuples. And, hence, this is called a vector protocol.

It really should be called cost vector protocol. But initially, they ran this thing where all of the links had cost of 1. And, therefore, they were minimizing distance. And the name stuck.

But if we wanted to be perfectly precise, we would say cost vector. But then no one else in the world would understand what you meant. So we say distance vector.

OK, any questions? All right, for some destination, how long does it take before every node in the network has a route to that destination? By how long, I mean, how many advertisement cycles do you have to go through? How many of these advertisements?

Let's say that the way our world is going to work is initially every node advertises its own advertisement to itself. Then at the next time step, every node advertises the routes that it knows about. And then it does that periodically.

So let's say that, every t seconds, a node sends out an advertisement where an advertisement basically contains this vector of tuples for all of the destination it knows about, right? So let me explain this protocol again, and then I'll ask you the question. The protocol is very simple.

Every t seconds, what the node is doing is looking at two columns in its routing table. It's looking at the destination column and the cost column. And it's just taking that information out. And it sends out an advertisement, the distance vector advertisement, every t seconds.

How long does it take? Now, let's focus on one destination D, some destination D in the network. And you have some network. How long does it take before every node has some route to the destination? Yes.

AUDIENCE: [INAUDIBLE] up to the number of [INAUDIBLE].

HARI BALAKRISHNAN: Up to the number of edges in my network?

AUDIENCE: Yeah, because if you're [INAUDIBLE].

HARI BALAKRISHNAN: All right. So let's say you have a network that looks like this. How long does it take before every node has a route to destination D?

AUDIENCE: So in that case, it only takes one--

HARI BALAKRISHNAN: Therefore--

AUDIENCE: [INAUDIBLE] worst case everything is in a line.

HARI BALAKRISHNAN: Well, I'm asking for an answer that holds for all networks, not for--

AUDIENCE: Oh, OK. What do you call the worst case, like how fast [INAUDIBLE].

HARI BALAKRISHNAN: Sure, in the worst case, it could-- well, yes. In the absolute worst case, it is true that it'll always take time smaller than the number of edges in the network. But you can come up with a much better bound. So let's try to come up-- yes, sir.

AUDIENCE: What if you did the number of nodes [INAUDIBLE] longest length chain?

HARI BALAKRISHNAN: Longest length chain, so that's not completely true. Longest, what kind of longest length? So another counter example, let's say that I have this network. The longest length chain is 1, 2, 3, 4, 5, 6.

But, yet, you guys just told me that, in one shot, you get the answer. So you have to clarify what you meant a little bit. You're almost right.

[INTERPOSING VOICES]

AUDIENCE: In this cast, it might actually be 6, right? Because you want to find that path of the top one and the bottom one?

HARI BALAKRISHNAN: I said find a path or find a route, not the best route.

AUDIENCE: [INAUDIBLE]

HARI BALAKRISHNAN: How long does it take to find a route in a network? You said this almost. The longest path-- but it's not quite the longest path. It's the longest something path. Yes?

AUDIENCE: [INAUDIBLE]

HARI BALAKRISHNAN: The longest shortest path, that is the the longest path when you compute the longest over all paths with a minimum number of hops between one place to another. That's also called the diameter of the network. OK? Yes? Yes? OK.

All right, now, that's the time it takes a multiplied by t. And I might be off by 1. You know, it's that minus 1. Actually, it's not. It is the number of hops along the longest shortest path.

Now, how long does it take to find the minimum cost path to some destination D at all of the nodes?

AUDIENCE: [INAUDIBLE]

HARI BALAKRISHNAN: What?

AUDIENCE: [INAUDIBLE]

HARI BALAKRISHNAN: t times-- no, that's true that you can find it within that. That's too long. So I'm going to come back to this question. It's probably answered in chapter 18, I think.

But we've come back to that next time. It'll become a little bit clearer. But you should think about it. There's a nice, succinct answer to this question.

And generally speaking, in every quiz, there's some variant of this question. So you know, it's not explicit, but there's some story that requires-- yes, you have an answer?

AUDIENCE: No, I have a question.

HARI BALAKRISHNAN: Oh, you do. OK. I'll come back to that question next time. But yes?

AUDIENCE: Can you go over again what you mean by the longest shortest path?

HARI BALAKRISHNAN: Yeah, I can go over that. So let's say you have D here. You look at this destination D. And you look at, from every node, what's the path with the smallest number of hops-- number of hops, not cost-- to get to that destination, right?

For this guy, it's 1, 2, 3. For this guy, it's 1, 2, 3, et cetera. Whatever that biggest number is multiplied by t is the answer. That's how long it takes before every node hears some path.

But it's not quite the right answer for the best path. Because as you saw in this example here, it took us one step before n2 got a route to the destination. But it took us two time steps before it got its best route to the destination.

And the reason was that it has something to do with the length of the minimum cost path, right? Because in this case, the length of the minimum cost path is 2 plus 6, 8. It took 2 hops, which is different from the length of some shortest hop path, which was one hop.

So if I look at this picture, n2 hears about some route to the destination in the first advertisement. But then to find its best route to the destination requires us to actually wait around until we find this 2 plus 6 path, which took 2 hops. So it's a little longer, but it's not enormously long. It's just a little bit longer.

And the answer, of course, depends. In the worst case, it could be quite long. But it depends on the number of hops along the minimum cost path. And if you minimize that quantity, you'll find the answer to the question, how long does it take before every node finds the minimum cost path to the destination. OK, is that clear?

Any questions about distance vector? Crystal clear? OK. We'll see when the lab comes around. It is crystal. Everybody really, really does well in these labs. And it's a lot of fun hacking this stuff up.

So you'll implement both protocols. And you'll actually look at all sorts of failures. And it'll just be sometimes miraculous that it actually works even when you didn't consider some failure cases.

OK. Now, I'm going to talk about link-state routing. And this is the routing protocol that you guys were sort of attempting to implement, attempting to come up with. This is a radically different approach from the vector protocols.

In a vector protocol, everybody advertises, for each destination, a vector of tuples where it's the destination and the cost. In a link-state protocol, we don't do that. The link-state protocol does not compute in a distributed way.

In a link-state protocol, every node just says, I am node 17. And I'm connected to 16, 45, and 44. And the cost of my link to 16 is 7. The cost of my link to 45 is something. And my cost of my link to this other neighbor that I have is something else.

So every node advertises what I've shown up on this slide. Every node advertises a neighbor, its immediate neighbor, and the link cost to that neighbor. OK. In addition, in each of these link-state advertisements, there is a sequence number.

The sequence number starts at some initial value, like, say, zero. And every time one of these link-state advertisements is sent-- and that's done periodically as well, every t seconds. Every time you send a link-state advertisement, you increment the sequence number by one.

Now, here's the key step. The key step here is that, if I receive a link-state advertisement from you, I just send that to my neighbors. And then my neighbors will send it to their neighbors and so on. So it's a very this nice flooding protocol.

Every node sends out its link-state advertisement. And every neighbor that receives it processes that and then turns around and ships it to their neighbors. And they do the same thing to their neighbors and so forth.

Now, when this flooding process completes and every node is originating its own link-state advertisement-- so you're telling them your neighbors who you're connected to. She's telling her neighbors who she's connected to. And we all do that.

And we're all doing this in parallel. It's all happening at the same time. And all our neighbors are rebroadcasting this.

And eventually, every node is going to get one or more copies of every link-state advertisement, which means that every node can now construct an entire map of the network. Every node can construct this entire graph. And once they construct that graph, every node can implement some shortest path routing protocol to compute the paths over that graph.

This is very different from the previous protocol. In vector routing protocols, the nodes actually have no idea what the topology of the network is. All they know is they trust what the neighbors tell them. Here, under the basic model, every node has complete knowledge of the overall network topology.

So let me show this by example, and it will become completely clear. So let's imagine that this is what the network looks like. I'm going to show you a picture of node F. Node F originates its initial link-state advertisement. And then every advertisement, it increments the sequence number by 1.

And it says, I'm connected to node G with a cost of 8 and to node C with a cost of 2. And it spits it out to its neighbors. Each of those neighbors turns around and does the same thing.

You rebroadcast a link-state advertisement along the links that you are connected to. And they rebroadcast it and so forth. And eventually, B gets it. And he broadcasts it, too, though it was completely useless to do so.

Well, not completely. If packets are lost, it's pretty useful. Anyway, when this flooding completes, which takes some number of steps, every node now has at least one-- if no packets are lost, every node has a bunch of copies of this link-state advertisement.

If packets can get lost, as long as the loss rates are not enormous, every node might have one copy of a link-state advertisement. And, now, every node originates its own link-state advertisement. And, therefore, they end up with a map of the network. By the way, why do we have the sequence number in the link-state advertisement? Yes?

AUDIENCE: I don't know, if F broadcasts its advertisement and something changes and it broadcasts a new advertisement, by the time the two advertisements get to B, you don't know which one gets there first. So you want to [INAUDIBLE].

HARI BALAKRISHNAN: That is one of the two reasons why you have it. That's actually the second reason. And that's a valid reason. But the main reason you have it is that, if C gets a link-state advertisement originating from F with sequence number 17, and then eventually C is also going to get D rebroadcasting that link-state advertisement. Because F sends it this way, but F also sends it this way.

And that goes up here, and that comes down here. And then D rebroadcasts it. C needs a way of telling whether this link-state advertisement is new or old, right?

And the way it tells if it's new or old is it considers a link-state advertisement to be new if it's bigger than the last sequence number it received from that origin. So, now, every node has a map of the network. And, now, we run this integration step where we actually take this map of the network, and we find shortest paths to the network.

I need a show of hands. How many people know how Dijkstra's algorithm works from class? How many people don't know how it works?

All right, what I'm going to do is it's described very well in the notes. We'll talk about in recitation tomorrow, but I'm going to show it by example. And then we'll come back to this again on Monday.

But I'm going to tell you this now because you kind of need it for the lab. And it is described very well in the readings. And we'll do it in recitation as well.

So here's how it works. Let's imagine we want to find paths from A to all of the other nodes in the network. Now, initially, A doesn't know paths to anyone except for itself. But what A knows is this map of the network.

And what A is trying to do is to find routes to all the other destinations. The way it does that is it keeps building up in non-decreasing order of the cost of the minimum cost back to the destination. It starts building up information about the routes to the different destinations.

So initially, it looks at this table. And it says, it's connected to C. And it's to get to B with a cost of 6. So what it does is it says, all right, among all of the people out here, I'm going to pull in the person with the minimum cost path. And in this case, it might just pick the node C. Because between C and B, it doesn't matter which one it picks.

Now, the cost to all of the other guys is considered to be infinity. So it has costs of 6 and 6, so it pulls in one of them without loss of generality. It just picks one of them. And, now, it has costs to both of them. And it says that the route from A itself, at A, the route to C is this link.

What it then does is it goes and looks at all of the neighbors connected to A and C. And in fact, it only has to look at the new node it pulled in. And it has to adjust the cost of the minimum cost path to the destination that's connected to that.

In this case, it adjusts from infinity. It brings down the cost to D to 13. Because it knows that it can get to C in 6. 6 plus 7 is 13. Similarly, it does that for E at 10. And then it does that to F at 8.

So, now, it has costs of 6, 6-- that 6 is already in-- 6, 13, 10, and 8, and infinity. So, now, it has to decide what node to pull in next. And it pulls in the node with the minimum cost among the costs that you have so far. And that's this node over here.

So it pulls that in if my wireless works. There we go. And then once it pulls that in, it adjusts the route to that to be that green link. And then it goes ahead and looks at the neighbors of B. And it adjusts the shortest path cost.

In this case, that 13 now becomes 11 because 6 plus 5 going through B is shorter than 6 plus 7 going through C. And, now, it repeats. It pulls down the minimum. The minimum, in this case, is 8. It pulls in F, makes that be the route to F.

Now, the route to F is not that link. The route to F is, in fact, this link at the routing table. But it knows that because it knows that F is connected to C. And, therefore, the route to F is equal to the route to the parent, which is C. Therefore, in its routing table entry, it makes the route to F be that link, which is exactly the link to C.

So that's the subtlety you have to keep in mind when you implement this stuff in the lab. It goes ahead and adjusts that to 16. It now pulls the minimum, which will be 10, and then adjusts the cost of the guys connected to it.

So D changes to 10. And then it now goes ahead and pulls the minimum in. In this case, it's D with the cost of 10.

That's the link to use. And that link, therefore, the route to D is the same as the route to E. The route to E is the same as the route to C, which is that link. And, now, you finally conclude the algorithm by getting that last node in.

So I'm going to stop here. That was Dijkstra's algorithm and the two routing protocols. We'll pick it up in recitation tomorrow.