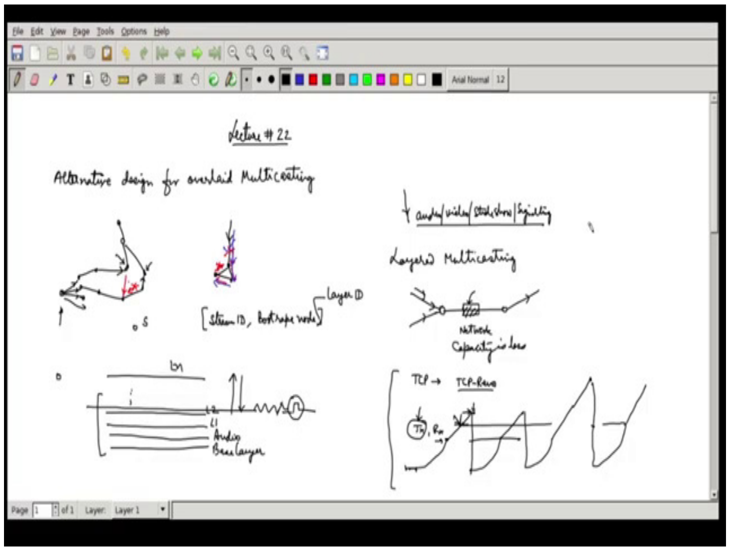
Peer To Peer Networks
Professor Y. N. Singh
Department of Electrical Engineering
Indian Institute of Technology, Kanpur
Lecture 22
P2P Overlaid Multicast: Alternate Design

(Refer Slide Time: 00:19)



Lecture # 22
Alternative design for overlaid Multicasting
[Stream ID, bootstrap node]
Layer ID
audio/video/Stills/Tone/Signalling
Layered Multicasting
Network Capacity is low
TCP → TCP-Reno
Audio Base Layer

This is lecture number 22, the Peer to Peer MOOC series. In this lecture, we will be looking at another alternative design for overlaid multicasting. In the earlier video, I had discussed a design where we were using neighbour tables. So, note if you want to get a field, he was having some neighbour tables, so these are the neighbours who are listed in neighbour table, he was sending a broadcast to these and these broadcasts will then keep on moving around and somewhere when the source is already broadcasting.

This will go to their neighbours and so on, so there may be multiple places the request may reach. For example, this which is here where the feed is already available, and another one signal reaches here, they both will start giving the feed. Then we had designed that we had decided that when the field is actually being received for example, say from two places from this as well as from this, only one of them will be accepted, other one will be rejected in a way that distribution feed actually reaches to the seeker, the guy who wanted to receive the feed.

But there was one problem which was also evident at one of the place in the lecture that if for example, a node is only allowed to have two neighbour tables, so these guys now having each other. So, before even this guy joins in and this guy is getting a feed from somewhere, and this feed is being for example, now being given to being received and given to this person, when this guy comes in these three guys will update their neighbour tables.

And when they will actually this guy will try to find out, he will find this is a more lucrative option to get the feed, so it will say okay give me the feed. So, feed is now supposed to come this way, but he will also relinquish this master node and ultimately all three will stop getting the feed.

There was this particular possibility which existed and in that case, when the moment is figured out, that feed is not coming now, when I left it. It means there is a cyclic thing. I should actually not subscribe to the new subscription, it will really relinquish this one and actually join back this particular feed and this feed will now again will start appearing.

This will become correct and this will again start appearing and now the feed will go in this fashion that was the way this problem can be solved. But now, today let us look at an ultimate design; we have this problem it will not happen all together. Now, this broadcast the join which is being sent as in the broadcast packet to everybody from a person who wants to join because there can be multiple guys who were trying to join simultaneously and these packets will actually be forwarded, they will not be discarded.

While even joining with only one person everybody who could have actually become may start receiving the feed, because it's joined broadcast will be actually received through everybody who is not even getting the feed now. So, the alternative way is that that every node, which if he joins will find a bootstrap node. So, ultimately when the source is starting all alone, he is the only guy in the bootstrap mode there can be multiple of them.

These, bootstrap node, will mostly be listed in a stream advertisement. So which will contain stream ID, so now you would identify this one, you remember this particular information is important to join. So, there is a bootstrap nodes maybe one or two. So in worst case, it will be the source itself, the source it has a provision for 4-5 kinds for further distribution can actually put a list and of course, this basically the thing will be there and of course in which layer this actually follows. So which DHT layer, because that is also important, you have to join that particular DHT layer.

Initially you will not be, you will be only in base layer or maybe some other layers depending on what services you are actually using to peer to peer system. So there will also be layer ID which will be defined for this. So stream ID and layer ID will be having one to one unique mapping. This solves, this guy first of all gets a bootstrap node, we have to essentially put this thing in its own routing table and say this guy this S is in the routing table, routing table action will happen. The new neighbourhood of S will essentially known to him, he will now start optimizing its routing table, S (Source) may not be optimal there may be other guys who are closer and who are maybe in a different logarithmic partitions, we are still doing logarithmic partitioning here so, ultimately it will build up its own routing table.

It will also build up the neighbour table which will be based on the proximity, the nearest proximity guys, and every time a routing table update actually has more than one option we will be choosing somebody which is closer to us, it will be either through RTT test router time delay measurement, if the guys were options are not in my neighbour table, if they are I know already, which is going to the closest one I will pick up that.

And of course now this way the routing table and neighbour tables will get updated. Now, one more thing which we can actually now removing, earlier there was only one media stream which was coming. It was audio, video, can be a slideshow or whiteboard, maybe some signalling, but all this is coming as one single layer.

So either you get everything or you do not get anything. We can now innovate on this, this is actually based on layered multicasting, and these are basically based on layer multicasting. Now, what happens is in the network congestion can happen, so you are actually passing a traffic through this node and this goes to another one, this is going to the router and there is a congestion in the network so network capacity has been reduced so there is no network capacity is less.

In that case what should be done? Only way you can actually handle it is by reducing the traffic which is passing through this, everybody whose traffic is passing through this has to reduce the amount of traffic and match whatever is the demand which you are putting on this particular bottleneck link to the capacity of this bottleneck link.

You cannot actually exceed that. You have to drop the things, now this actually is feasible in TCP, TCP Transport Control Protocol uses an algorithm for TCP Reno. It is basically a flow control mechanism but it also does congestion control. Here, there is a transmit window and there is a receive window. Transmit window is what is being adjusted periodically, you will start with a small transmission window and you find a round trip time in the acknowledgement, by the time you get the acknowledgement that time is within limits, and then you actually increase the transmit window, you double it actually. So, this keeps on growing exponentially first. At some point of time, it will then become linear after that and if you would actually start getting losses, that time you will then further immediately reduce back to the original minimum value again.

And this value will be the half of this value, whatever is this values this half I will decide. Again I will grow exponentially till this and then there is a linear thing. And if my receive window is actually much larger then only this oscillations will start happening. Okay, that is what happens in TCP. So, this time probably it has gone here so it's half will be somewhere here, so it goes back and then transmit window grows and then after this it will increase linearly.

This periodically happens that is what the how the transmit window is controlled. But then my receive window is below this point, it is somewhere here so then transmit time my actually the sending window which I am transmitting without waiting for acknowledgement will get stuck to this because there are so many minimum of these two and I will keep on operating here, this congestion control will only start get triggered when my receive window is much larger, I am not limited by the receiver.

So average value will be now matching towards whatever is the network capacity. So this will be done by everybody in TCP. But incidentally for all our media transmission for audio video packets because I do not want reliability if a packet is loss it is lost because even if you retransmit it, it is of no consequence by the time the time has elapsed, this utility.

For UDP, we cannot use this algorithm TCP algorithm. In UDP, what we have to do is I can actually now say okay, there is a base layer which is signalling, the signalling and some basic information maybe after that I can send an audio, so audio communication is important so audio goes in another layer.

So when we use IP multicast, we actually give separate multicast addresses to these and depending on if you are not getting all the things very nicely and there is a lot of delays, you are getting gaps you start you actually leave the certain upper layer multicast groups and ultimately limit yourself to something which is going to give you the best performance at your playback.

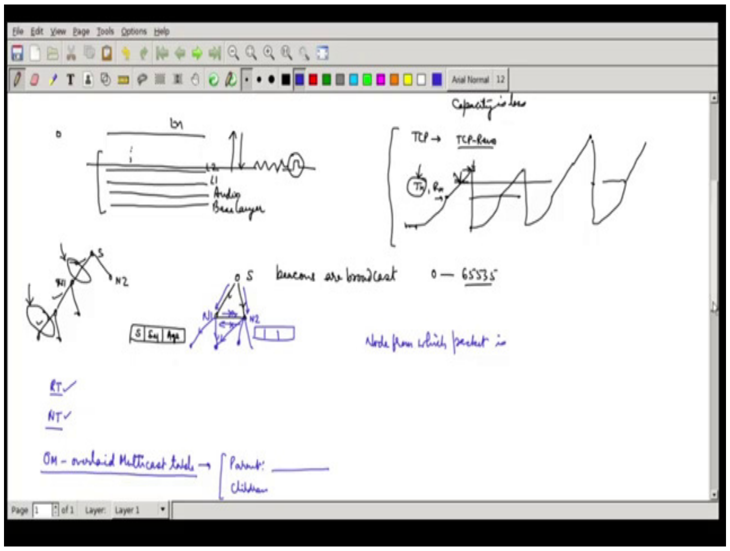
So you may be happy only with the audio coming in, you do not want the video to come in, you were dropping. Then videos based layer, this will not give a good quality video, but which is fine, you are happy with it. Maybe if you have still bandwidth, you can subscribe to one layer to support video quality will improve and you can keep on doing it. And there maybe nth layer which is the last one, because when you subscribe to all these layers, your audio is good and you get a high class high definition video. In fact, ultra high definition you can get.

Depending on what the network condition, you will start actually either joining or leading, that is what is going to happen. Normally if you are happy it means your things are satisfied till this layer and if you are happy for some time, then you make an attempt for joining the higher layer. And if you remain happy, you just again further make an attempt for the next layer, otherwise you will fall back.

Normally you start oscillating between layers because you become unhappy the moment you join, so you leave, remain for some time again make an attempt, again you keep on doing this actually that is the only way you can figure out what is the exact, how many years you should subscribe so, this is an experimental thing.

Another alternative way is, somehow you make an estimate of the bandwidth and you know how much is required and based on that join only that that much. And when your estimate says bandwidth has increased to an extent that next layer can be done then only make this attempt. The chances that you will fall will become less in that case, so both strategies can be used.

(Refer Slide Time: 11:43)



But this is an IP multicast. Now, even when this overhead multicasting we do actually have we are sending these packets which are like UDP and we will be using layers. Now when I am using layers it is important that in the broadcast even source is doing so all layers should be transmitted, this guy should be able to receive all layers. This is some node1, there may be another node2 is there so this also should receive.

So, as you go further down and down in the tree, the depth of the tree then the number of layers, the capability here should be maximum, here can reduce, here it can further reduce, it should never happen this guy has the full capability of transmitting all the layers and this guy can only transmit the base layer. Now this does not make sense, a lot of people will suffer.

Now this we can handle by using again a reputation management system which we have talked about in the previous video. Now, how the tree will get created? That is one of the questions. In the earlier video there abroad joined actually started from here, receiver the source was never bothered about it. But we can make a change here which we are planning to do now in our Brihaspati-4 design, the source actually sends beacons.

So beacon is a special packet, beacons are broadcasted periodically. So advantage is that your broadcast routing table will not be that complicated, it is only one source which has to be taken care of, not multiple nodes which are trying to join. Beacons are broadcast periodically and every node will have for example, these beacons have been received by these two nodes. These will have a broadcast routing table, which will say that it has come from source, this was a sequence number and this is edge, so that is what the beacon will contain.

So every time sequence number has to be incremented, it will never be decreased. So in worst case, this will go from 0 to 65535 assuming that I am using 16 bit for this count or sequence number and then it will go back to 0, that is why age is there. So, if an entry is not being updated for a certain time, this age basically expires I will force this entry.

Even with the zero sequence something will come. I will then accept it and move forward. So, now the packet will be coming here, packet reaches here, if there is for example, a connection from between these two, as a consequence one packet will come here, they will make the entry then they can actually make the broadcast in this direction, this guy will simply discard, this will be also discarded because the entry already exists in a table here as well as here.

So these are two guys, these can actually further do the broadcast and so on. As a consequence, you will get a tree getting created from source but only thing I mentioned there is a mesh. In fact, technically we do actually have a mesh because of the routing table being managed, but that mesh is not on the basis of what is the capacity of a link, capacity of a communication link between two nodes, whether it can transmit only or not. It is not on that basis, it is based on the node IDs basically if I look at the routing table. And at thus some neighbour table can also be there so proximity can also participate, but this cannot guarantee this kind of thing, so I have to build something extra.

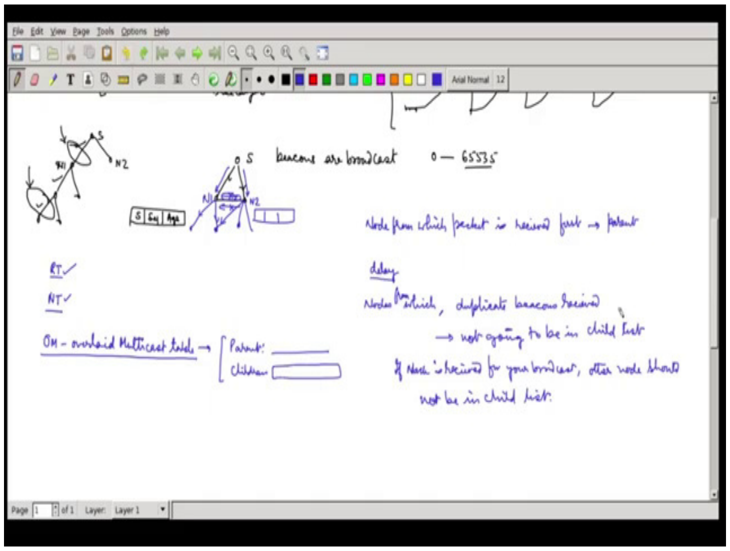
We create a third table we call it overlay multicast table. Now, this actually makes protocol slightly complicated, but this is going to work as per the current understanding and so, we call it a OM table, so this needs to be maintained and this will be in addition to RT, NT and OM.

This basically will now keep the record of who is my parent, so the node ID and basically the endpoint address for that and who are my children. Some of your neighbours may not be there in any one of them. It is possible for example, as is will be in the parent of N2 but N1 will not be in the child even if it is connected. Similarly, N2will not be in the child entry of the N1. So there may be some other nodes which may be here so they may be nodes like this. So this guy will be only the children of one of these two nodes, not both of them.

Some of them have to be discarded. In fact there is a protocol which we have to when this broadcast is done. So, we have to from whichever route I get my first packet in the beginning for example, see if this table is not being constructed, and this table is empty. Source will now send the beacon and because our routing table we are sure that every node is participating in the network, they are not disconnected.

So and of course we can also allow all the other nodes in NT2 also be used for doing this broadcast of beacon. So whenever a packet is going to be received from a source from a previous node, so that node ID, node from which packet is received. Once I am able to create this one table then of course my broadcast of the media stream is far easier I will always receive the things from parent and then I will pass it on to all the children one copy each that is it.

(Refer Slide Time: 17:24)



beacons are broadcast 0 — 65535

OM - overlaid Multicast table → [Parent: ______ Children: ☐

Node from which packet is received first → parent

delay

Nodes from which, duplicate beacons received

→ not going to be in child list

If Node is received for your broadcast, other node should not be in child list.

And as children are going to get they will manage my reputation and I will always try to find out from my parents who are their parents and we will try to always keep on trying to attach one of them to get a higher reputation actually. Because if I am actually serving more number of children and I am serving more number of layers we have now made the change that you serve more layers you can have a reputation, if you are serving less layers you will not.

Get higher reputation get less of So we have changed that formula slightly in this case. So nodes from which let us come back to the protocol here, the nodes from which the packet is received first, so as can come actually from multiple times this guy has got from here as well as from here.

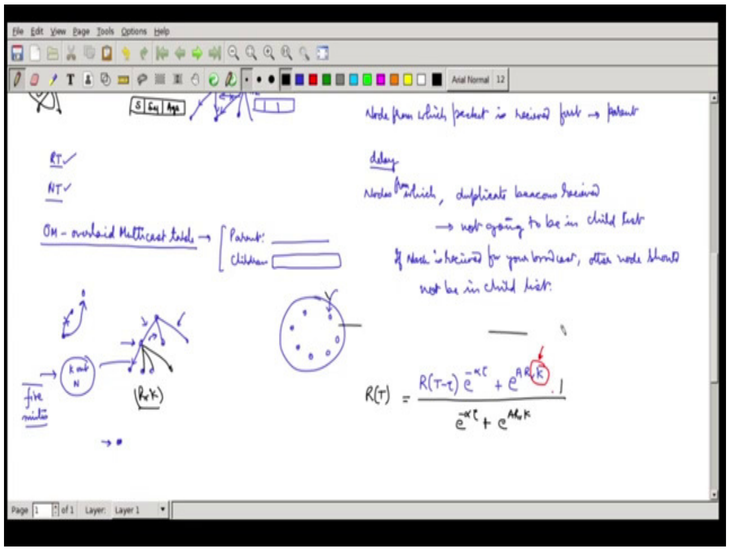
So whatever is the first that will become the parent, then it will actually it is better to do some delay here, so then there will be a delay which will be inserted intentionally. By the time you may get actually broadcast from other nodes. So nodes which are, so initially you will keep everybody in the children but you have still not broadcasted anything, but these children think, we have to keep on updating.

Nodes from which duplicate beacons received, I am not going to be in child list and of course, the moment you get a duplicate you for example, get a duplicate from here, you have already sent your packet so you got a duplicate so, you have to discard N2. And if you have not sent this one this particular packet because there was a more delay here, less delay here so this packet was not sent, only you could receive this one.

You need to send a NACK so that N2 will also not keep you in the children list. And if neck is also received for your broadcast, then also you will not be. You send something other nodes should not be in the child actually in those cases. In fact, these both sides the packet will never be happening because there is a different delay which are randomly put.

So it gets a broadcast, it has now delayed. This delay expires first to extend the broadcast this figures out its duplicate discards it and send a neck. So when the delay will be over, delay whatever you have inserted, you will not be broadcasting here, you will be broadcasting to other nodes. And both guys will only remain in the children. That is it. In this way, now you will actually form a tree. So, this tree is what is going to be used for now sending the broadcast of the media.

(Refer Slide Time: 20:53)



Node from which packet is received first → parent

delay

Nodes from which, duplicate beacons received

→ not going to be in child list

If Nack is received for your broadcast, other node should not be in child list.

OM - overlaid Multicast table → [Parent: ____ Children ____]

$$R(T) = \frac{R(T-\tau)\, e^{-\alpha\tau} + e^{AR_r \boxed{K}} \cdot 1}{e^{-\alpha\tau} + e^{AR_r K}}$$

The media will always be coming from will be received from your parent and you will be giving it to your children to nobody else. In case your parent stops giving, you have to essentially find out which is the second choice of the parent if it is feasible, you can actually keep parent 1 and parent 2, parent 2 is the one from where you got the duplicate. Second duplicate, first duplicate actually beacon and which was removed.

So you can actually keep even parent 2 also that is alternately possible, or after some time whenever you will get a beacon from some other that will be the first one and that node will become your parent, and he will start giving you the field actually. There will be some disruption which can happen, but with two parents being maintained their backup parents, you should be able to handle this.

And of course, we also have a mechanism whereby you know the use of grandparent, you always ask your parent who is your grandparent, this information should be there. If you stop getting it, you just try to join a grandparent and get the field; most likely this guy has failed, okay.

Actually, he is not giving it to you, but gives it to somebody else, there will be a full separate procedure by which he is pushing you out and trying forcing you to get connected to somebody else. Now, because we are using layers, I need to change my formula of giving the reputation. Ideally, what will be happening from source the guys who can actually get all the streams have higher bandwidth with the source? They will receive so they will actually earn higher reputation because they can provide a higher capacity.

The guy on the bottom side should be with lower reputation, because they are either not feeding others, or they are incapable of feeding others or they are getting very few layers both. So my reputation has to be now based on layer. So if somehow it is getting a feed from here, and he is getting some layers, say he is getting K layers out of N, K out of N layers he is getting.

So you need to send a message to the reputation. Again, there will be a DHT separate DHT network where the reputation of everybody will be managed. And again, rule is that for every node, there will be somebody responsible, we will just compute the hash of the node ID and then find out here and then report the message, do them send the messaging. So again, every five minutes, you can actually if you are getting the feed continuously, every 5 minutes, you can send a bit. We have kept it 5 minutes, but this can be any duration depending on the way you design is your choice.

But normally, it is this guy, which will figure out whether what all updates have got in last five minutes, if you have already sent an update, it will anyway not be accepted. It does not matter on you; it matters on the reputation DHT servers. One of the reputation server gets the message, what it is going to do is it has to compute what was the reputation earlier so maybe last reputation was built four times earlier.

And now suddenly there is a reputation has to be estimated now. I will find out what the decay in the reputation so alpha has to be suitably chosen plus the new guys who are actually now giving me the thing. So I will now put A, now I am actually doing something very interesting here. You can put a thought, because this was a very important thing, A is a constant again it has to be manipulated properly figured out.

And then what we do is we add the reputation of the node, which is sending me and how many layers, so more layers he is doing it, this is actually reputation gets amplified. So I get amplification benefit in that case. If he is getting less layers or he is having less reputation, so these both actually technically gets multiplied.

So it is better it is always I think advantageous for a node to give the feed to the guys who are actually is able to receive more number of layers as well they have higher reputation. If you have multiple options available to you, you will tend to choose somebody who has a higher reputation as far as who can get all the layers.

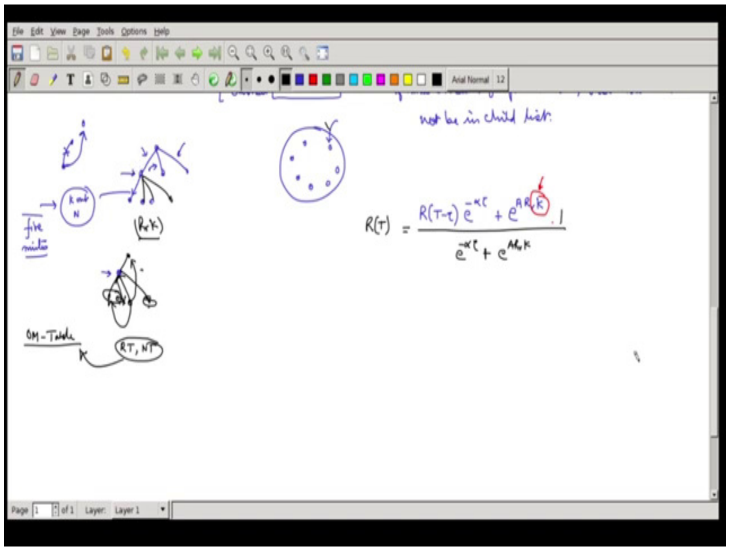
So as a result, the guys who are having weaker bandwidth and who are actually having lower repetition, they are not cooperative to the network, they will tend to go towards the bottom, they will tend to become leaf node in that case. So, this is the only new chain which has come in which was not there in my earlier thing. What is the chain? Again I will multiplied by 1 and then do the averaging. It is going to be $e^{-\alpha}$ are basically weights e^{ARk}.

This gets higher and higher weight, if you get more and more feed. So this is how the formula gets changed and now there is a new reputation value of the node which is actually feeding. So these are basically the receivers which are trying to send out, if some receiver is not sending correct updates, a feeder can actually kick him out anyway he is not gaining anything out of him and this poor chap has to find out somebody has to latch on.

So he may ultimately go to the bottom of the tree and may get very few layers, not a high quality stream. For him the only best strategy to be cooperative and to send the things honestly, updates honestly so that the other guy does not kick out. For the guy who is giving the feed for him, it is always best to find out people who are having higher reputation and give their feed to him to the maximum or the guy who can consume, he will actually try to optimize AR into k.

So if you get many options, if you can give feed to many people you will compute AR into k for each one of them, whichever has got hierarchy he will essentially always tend to give the feed to that person? So not only is your reputation but the link chemistry between you and that node also does matter in this case, this was not there in the earlier system which I discussed, this ensures that my topology keeps on moving.

(Refer Slide Time: 27:00)



$$R(T) = \frac{R(T-\tau)\,e^{-\alpha\tau} + e^{A R_r \boxed{K}} \cdot 1}{e^{-\alpha\tau} + e^{A R_r K}}$$

Now, interestingly what we do is currently OM is not being used; OM is being just created, but will now change dynamically. So we have yet one more change is that we are going to create another part of the algorithm whereby that every node can ask for basically trying to join another node that should be feasible, so you can always ask your parent about his parents, so you should know your grandparent actually.

So this has to be part of the protocol, if somebody is not doing you will anyway will figure out some other better guy will join him, so you will give him bad reputation. So it is better to always cooperate and give it. So, whenever a node is going to be added to you, for if somebody sends a request you will also always send that information about who is your grandparent, this information is sent so he actually knows who the grandparent is and he can keep on attempting periodically to him.

If he feels that this guy will actually prefer him actually compared to somebody else. This way this can upgrade by actually gaining reputation actually, if you have higher reputation then you can join here. And then for example this guy figures out wants to kick this guy out, he will tell you who are the siblings, he will tell I have all these further nodes which are there. List of these will also be provided to him and he will be kicked out. So this guy can now join somebody here, here or here. It again, essentially you are pushing this guy towards the bottom of the tree. And by giving the parent you are giving you an opportunity to go up actually.

So normally this will be done because if you do not do it, this guy still can get the beacons from the entries which are there in the routing table and neighbour table because this must be there in those entries. Beacon may be coming from there. If it gets a beacon from somebody else who is better than this, he will try to attempt in join there actually and will leave him, so it is better to give the parent actually still you will at least get the feed.

Even if this does not tell about the parent, this information can be retrieved statistically from the neighbour tables or routing table based beacon broadcast, because that anyway will happening. And additionally when the beacon is once these tables are getting this old table ultimately will become different actually over time, so this may contain entries which are not there in it RT or NT.

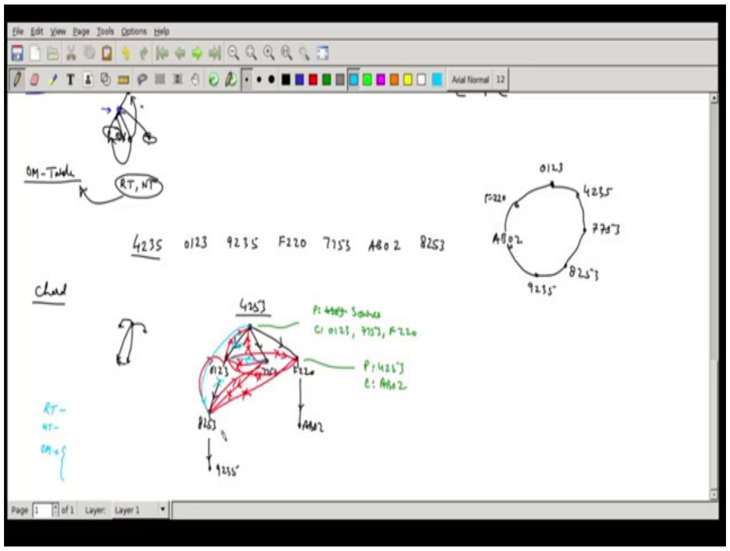
There will be some entries which are not present here but they are present here. That will happen over time. And the important thing is that when you are taking out somebody you just give the list of all siblings to him so that he can try joining to them but he is basically he will get downgraded adversely.

So, you will always trying to make an attempt to join somebody higher and you are trying to essentially whenever requests are coming to you, you will try to allow only those guys who have reputation into k basically, but k will come through measurement that how much less he can receive from you, so higher the AR into k the better it is for you actually.

So this is roughly what is algorithm which can be used and typically normally you cannot have large number of children, the more number you have better reputation you will get but after some time, you actually normally do not get any advantage. Sometimes you would like to kick out the existing children, accept somebody else's children because he will give you a better advantage actually. This can happen.

This actually will trigger re-optimization of the tree continuously and this will optimize in such a fashion that what we desire that links towards the source will have higher capacity and they can carry more number of layers and they will all the most the source will tend to have higher reputation into k actually, AR into k, that is what will be higher for them not AR this time, which was true in the earlier case.

(Refer Slide Time: 30:51)



Let us do a small short example where I will show that how the movement can happen. I am not taking large number of nodes, I am considering a Chord topology here, so let the nodes be 4235 a 16 bit number, 0123, then I have 9235, then I have F220, then 7753, AB02, I have just chosen something randomly and I have tried telling how this actually works, let the source be actually now 4235.

So if this is the source and I am actually assuming this is a chord, but this is going to be possible with everything with chord of course, I am also maintaining less number of finger table entries. The node will only be maintaining a successor, a predecessor and a node midway across so whichever is the root for this. So this this node essentially will be pointed to, these three will be kept everywhere. So other finger table entries are not there that is what I am assuming.

A node actually can do that if he does not have a higher memory space, need not maintain. For your predecessor, successor you can always be rooted but you may require more number of hops. Let us see what happens, so 4235 you can actually arrange them in a circle so it becomes easier for you. It is going to be 0123, who is the next guy after this is going to be 4235 and after 4235 you will get 7753, and after 7753 you will get 8253, and after 8 you will get 9235 and from here you will get AB02, and then F220, this is how they will get organized in their Chord and of course you can count your seven nodes yes, these are seven nodes.

So 4253 in the beginning because there is no OM table, you have only a routing table. I am also assuming there is no neighbour table also here. So, this is not a pastry, this is Chord.

Chord does not worry about the neighbour they are not required. This is only based on the numerical progress, essentially which matters here. So 4253 will now give a feed, these are the nodes participating in the DHT layer for the overlaid multicasting, 4253 will give a feed to, we will find out who are the neighbours there will be three neighbours actually for him.

So, one will be 0123 obviously, other one will be 7753 you can see and then diagonally opposite 8 plus 4 is 12 which is B, which means F220 will be the next entry. So, the beacon will be sent from here, so 123 so how the network connection essentially based on that, so the beacon will come here, here and here, they will be sending it to others. They will maintain a routing table, so duplicates are going to be there and they will be removed. So 0123 will be now sending this information beacons to home.

0123 will send it to 4235 back discarded, it will be sending F220 discarded, it has already got the beacon, the Mac will be sent so these two entries will not be there in the children list. And then of course third entry will be 0 plus 8 is going to be 8 so 8253. Yes, this will be the entry which will be there next one. I have made some mistake in my nodes but anyway, I am going to correct it here.

So 0 plus 8 because it is a 16 digit so, I am going 2^m- 1 is actually 8 in this case, okay? So 2 raise power 4 is 16 correspondingly A253 will be here, so this will be receiving the beacon perfect and then let us say all three has been exhausted now go to 7753. 7753 we will have 4235 to give a feedback will be discarded actually 7753. 8253 it will be doing this way, of course, discarded already beacon has been received from this route and then 8 and 7 are 15 so F753 so it comes to 0123 this will be coming here, it will be discarded.

Then F220 will, F220 shows that it has to be AB02 which is no more. This will be receiving the beacon, so red ones are the one which are discarded actually. F220 then goes to 0123 which is here; this will be discarded because it has already got the beacon, so this is discarded now.

F plus 8220 so 8253 so next, this will again be discarded. All three has been exhausted, only black one is the three through which is the beacons will be received rest everything has been discarded. Now come to 8253, 8253 goes to 7753 first, which is going to be discarded obviously, and 9235 does not exist, so we will have 9235, and 8 plus 8 is 16. F253, it comes to 0123 actually, so this will be also discarded, this already receiving the feed this basically tree getting created based on routing table beacons which have been passed some of them will be discarded.

Now come to AB02 so AB02 will have, in fact we have already got all 7 now. Whatever, nodes you pick up. I have already created all 7 nodes in the network. Now, whatever link you pick up now, they are all going to be discarded, this is where the beacons will not travel. But this will be the final situation which you will have and each one of them will now maintain an entry. For example, know that it itself is the source, these guy know that the parent entry will be itself 4253 and children entry will be 0123, it will maintain 7753 and F220.

Depending on the reputation it will be happening F220. F220 will have now parent and 4253, and children entry is AB02. In fact here you do not need to keep it; you say you are the source itself. Similarly, AB02 will have only a parent entry, this will have only parent entry, this will have one children entry, and this will have one children entry, only parent entry here. So, this is how it will be.

Now in this case, suppose somebody becomes a free rider, what is going to happen? 0123 becomes a free rider it does not get any feed, as a consequence the reputation will go down. Okay, his reputation goes down and meanwhile 8253 knows about 4253 or somebody else who is a parent of 0123.

When he joined actually that time it figured out, in between 0123 created a problem becomes a free rider does not want to give the outgoing link on the bandwidth, this does not have much money for example. In that case, 8253 will not connect to 4253 to make a connection, 4253 will figure out that 8253 has got a reputation because it was giving feed to 9235, but 0123 is not giving any feed so its reputation is going down slowly. It will start giving; it will throw this guy out, give him what are the sibling addresses, 7753 and 8220.

So 0123 is out so in that case, a new link will get created now, as a new link will be this entry will change and now this is the connection. This poor chap now has to figure out somebody else sibling appear so it will connect him. It says I am anyway not gaining any reputation because I am not feeding somebody is asking. I will very happily oblige and will make a connection in this fashion. This guy will be out, this connection will be out and you get a new topology now coming in, so 0123 comes here and this game moves upgrade. This is how basically the continuous movement of nodes across the topology will happen and one table slowly will become different than your RT and NT. NT is neighbour table with a routing table and there is now overlaid multicast table.

This will start evolving based on this exchange controls. And ultimately topology will move in such a way that it will stabilize when the higher reputation into the number of layers whichever of those nodes will move upward and those layers which have to this value is less value will tend to move downward.

So, there is a dynamic reconfiguration will keep on happening so this is another way we can actually build overlaid multicast, it is slightly different design. In this case, of course, it does not matter on a neighbour table because of routing table the connectedness is guaranteed and over which we are sending beacons and after that there is a dynamic optimization which happens.

So, this is a kind of a pretty much a stable thing, you will never get routing loops in this case at least. So, in the next video, we will look into how the male peer male transfer actually can be implemented. So that also is a part of the design for which we have been doing. Initial code is there but we have to essentially the volunteers had to work on it and further make it. I think a better quality code before it can be released. Thank you.