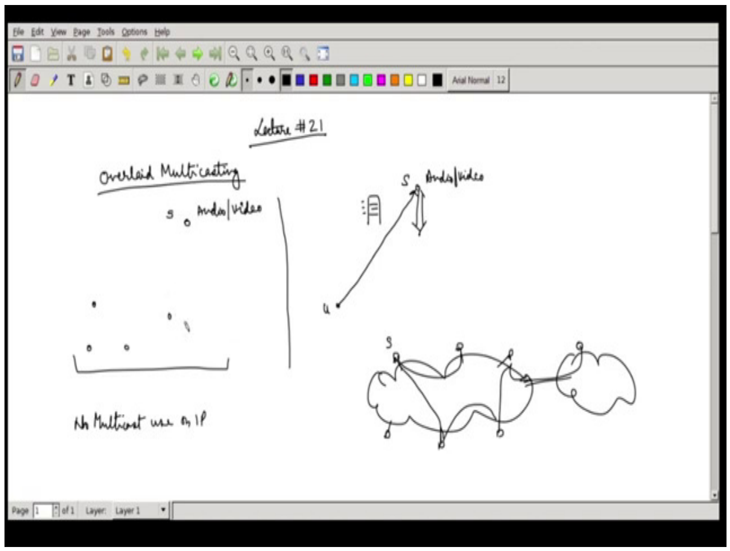
Peer to Peer Networks

Professor Y.N. Singh
Department of Electrical Engineering
Indian Institute of Technology, Kanpur

Lecture-21

P2P Overlaid Multicast: Basic Design

(Refer Slide Time: 0:13)



So welcome to the lecture number 21 in peer-to-peer networks course. In this lecture, we will be looking at the issues related to overlaid multicasting. In overlaid multicasting, the idea is that somebody is trying to send the audio or video stream or for that matter any real time thing, but there is only one source and there are many people who would like to access this. So, there will be many people who would like to access this particular session.

So, one of the simplest example because of which we have been doing it is to for transmission of live lectures. So, that was the idea in Brihaspati 4. And the basic mechanism which we have thought of is that instructor will actually give audio and video and this audio video and of course, screen capture will be available to everybody and that people will be now accessing through this mechanism which we are discussing. And whenever somebody is would like to ask a question or something, they should be able to raise their hand.

So, that will be done through a signaling mechanism. So, they have to send a signal which can go to the source and source will queue it up and then structure appropriately can look at the signals and then actually give a confirmation. So, this confirmation can be sent back directly, but this will be because they both will know their endpoint addresses.

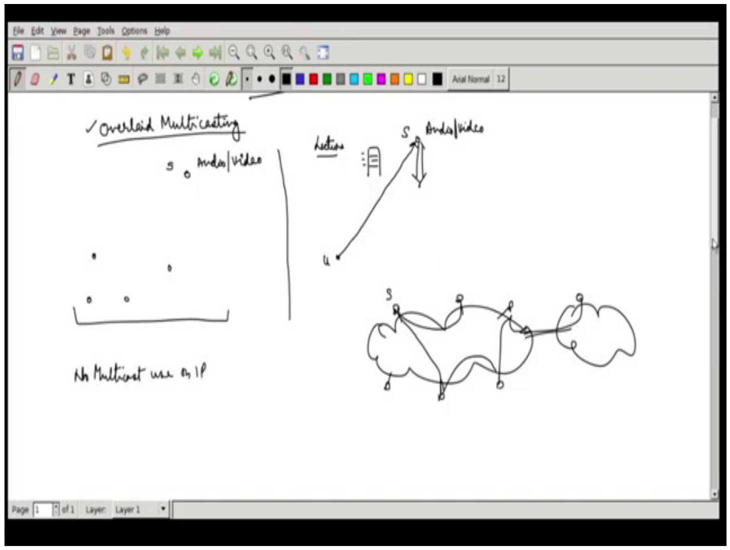
So, mostly the forward signaling from user to this will be coming through DHT route, but the reverse source can always tell the confirmation to the user that I am now confirming and then you are allowed to interact. And then the audio video of the student will be essentially streamed directly over a point to point link either using HTTP tunneling or to NAT or through a proxy. So, it is a direct connection and S (Source) will source will now actually mix all this audio video coming from him and his own interaction these both interactions will be now available to everybody as a stream.

So, once it is over, he will be disconnected and then source can take up more questions from the queue. So, every any one of instructor can actually anyone of the student can interact with the stream, that was the idea behind our design. But in general this can be used even for IPTV kind of implementation if they wish. So, but the question is now we are not using in this case the multicast which is being provided by IP, there is no multicast use on IP layer.

So, what we are doing is there is already a network and users are connected to this network the, there is an IP network. Then add application level I am actually going to create a broadcast tree. So, essentially source will be here, so it will communicate to this guy, may communicate to this, this can then communicate to this person and so on. This guy may then actually be able to talk to another network on somebody else. So, users are actually forming an overlaid network in this case.

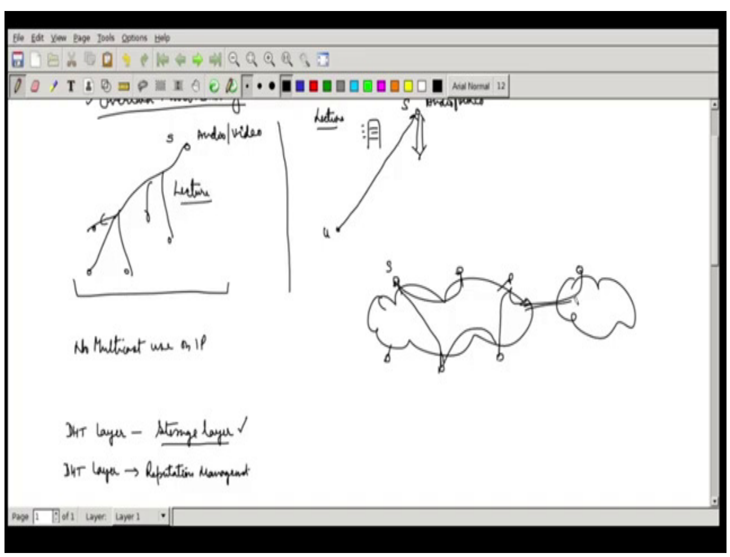
So, how this will be done in a peer to peer system that is the idea? So that is what I am actually now talking about in this lecture. So, we can actually take care of lectures or you can actually have conferences, all this actually can be done in this fashion.

(Refer Slide Time: 4:05)



So, this is a typical lecture scenario. Similarly, we can think of a conferencing scenario also. But normally, when in conferences, when large numbers of people are there will be one person who will be chairing it. So he will be like the instructor and everybody can raise their hand and then everybody can speak. So but the under conversations are going to be or everybody's views will be available to everybody else and that is how they can actually come to a consensus decision. Or they may not come but still they become aware of each other. That is what the purpose of any conferences is actually.

(Refer Slide Time: 4:38)



So, how this will be done? So 1 important thing is that we have talked about DHT layers earlier, if you remember. We have talked about a storage layer at that time. So storage layer was anybody who is providing in storage service to others, that will become participate in this of this storage in DHT layer. So, rest everybody can actually become a leaf node. But when you are leaf node, there is no use because you cannot actually use any storage service because you do not gain in the reputation.

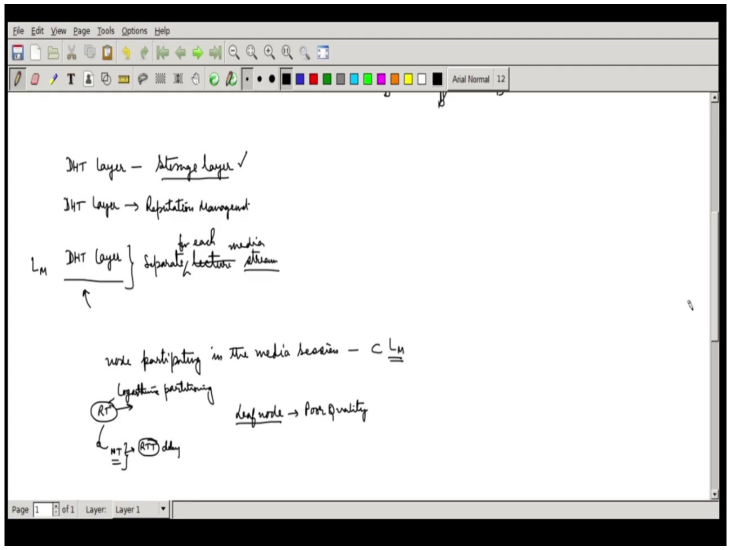
If you remember, that is what I had discussed in 1 of the earlier lectures. So, they need to be a part of the storage, they need to contribute something, the same user there can be multiple devices contributing and henceforth he will actually get a bigger share of storage from others. So normally, whatever you contribute, you get roughly almost similar thing from the other side. If you contribute less, you get less and this is basically we use the mechanism of reputation mechanism, reputation management system.

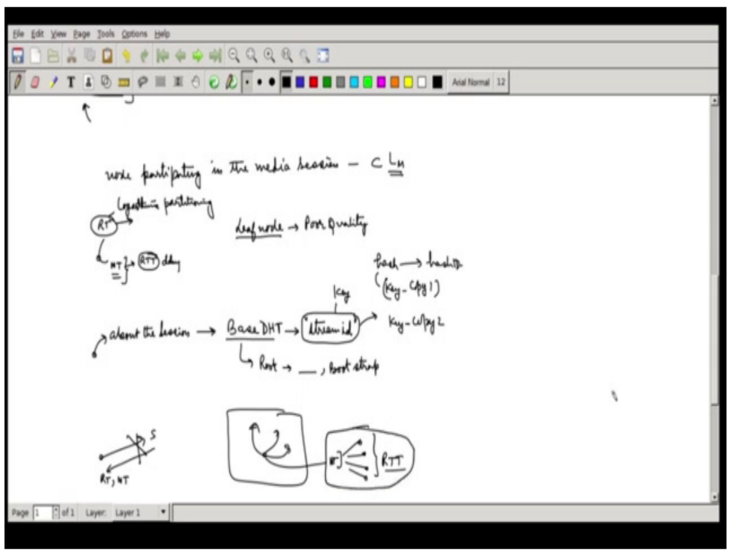
On another DHT layer, which was a reputation management layer, which is different, this is not much load, this is basically only the some people there is always for every node, there is going to be somebody who will be acting as a reputation manager. And everybody is using the service will keep on sending the information about the service being provided to the reputation manager of the server. And he will keep on updating the reputation of people.

So, when somebody is coming in with a request to view, that time you check the reputation and based on that, you will provide him service or you will deny it. So that is what the mechanism is. We will also be using a similar mechanism here, but it will be in a different context. And we can actually prove that strategically, there will be only one strategy, which allows the people to cooperate in case of overlaid multi casting system. So, now we are going to for every media stream, so every lecture, so there is 1 lecture I am talking about and these guys are going to listen to this.

So, there you are receiving, so I call this a lecture. So, this every lecture will be a one layer, that is very important, there will not be one single common layer for all the lectures, but for every lecture, I can actually prepare a layer.

(Refer Slide Time: 6:59)





So in case of your DHT routing, there will be a separate layer for each lecture. Or each live program, you can call it or each conference or each TV channel. So, separate for each lecture. And I should not use lecture now, I can call it a stream. But since I was doing it, as an extension of an LMS, because we did Brihaspati 3 LMS and Brihaspati 4 came, because there are some technical problems in that, in the sense, it was not massively scalable, we required a lot of servers and that is where I decided that we have to go to peer to peer we because that auto scales.

So, more than number of users has the more the system will become better and I do not need to invest in servers and everything. And that is more environmental friendly, because I consume less energy, less hardware, air conditioning required at the data centers. So, we had discussed all this earlier. So first fundamental thing, each stream, media stream I should call it now, that is the word which I will be using, media stream will have a separate DHT layer. For every node that is participating want to receive this media stream has to be part of this layer.

So, I can call it a layer for the media streaming, where every node participating or in the media session, so it can be a passive thing you are simply listening or once in a while you are interacting, has to be a member of this DHT layer. So, he has to be a member of that, so that what it means every node which is participating will have one routing table and will also have one neighbor table.

And the routing table will contain only the members which are, only the nodes which are the member of this layer, neighbor table will also contain the nodes which are the member of this layer. But this will be basically based on who has got the best performance round trip delay time. Round trip time we call it RTT delay, so has the least delay, that set will be maintained here. And this will be based on located next partition so that the network will never get partitioned.

There will be some node; it might be power participating the layer because anybody who is in the layer, so their node IDs will be present in this case. Anybody who is going to be leaf node, he is going to probably receive this stream but he will be at the lowest end. But he will be actually having a poor quality that is a problem. And there will be also he will be suffering from the delay. If there is a TV broadcast is done of a cricket match.

So, by the time the wicket is taken, the person gets out, by the time the last guy knows actually there will be a delay, because he will be essentially in this broadcast he will be at the lowest node actually, it is the lowest level, we call it the leaf node actually. Because these guys cannot garner the reputation, their reputation will always remain zero in the system. We will actually, when I talk about reputation then this will become obvious. So now, every node will now look at the RTT that we keep on the way it was done with earlier DHT layer same way the routing table will be managed.

Now, with this how the actually stream will be received. So most likely cases that whenever you are joined, you will join and you will come back to the bootstrap node first. So anybody can be bootstrap node. So, you want to join this session you will first of all find about the session actually. So, you will find about the session, so this information can be kept in base

DHT layer because this is not a permanent kind of key value pair, because this should expire after some time, then maybe the guy who is actually there is going to be a stream ID.

So, you will be getting through advertisement or through a mail or some other source or some of your friend might be telling it. So, you will now have a stream ID, the stream ID can be simply a channel name. So, you can say star TV and it can be the stream ID. And then this is stream ID you will pick up and from there you will find out, you will do and you will compute the hash value of this.

So again, this will be, hash of this will not be copied this is actually key. As we had done earlier, we will be having key and then copy 1, copy 2, kind of thing combinations which will be there. So that even if the node dies off, these entries can be maintained. Then of course these are for shorter time, but is still the same thing will be done. You will take a stream ID copy 1 and compute the hash of this and this will give you a hash ID.

Using the hash ID, you can search in the base DHT now, and go to the reach to the root node. The root node is supposed to be actually storing the stream ID and then corresponding what is going to be the bootstrap node through which you can connect. The bootstrap not only will be the first node, which will be in the DHT layer of the media stream. So, most likely it will be the source node always and maybe a few other nodes which can actually can come in.

So, media stream can give, it can always request few more notes to go and then do it. If media stream is there it is fine. So once that is there, so media's ID, node ID will be available here. In that case, the current node will be first of all talking to the source and getting its routing table. So, it will be also now making a neighbor table entry. And then further because whatever entries which are here based on that, it will start keep on optimizing and after some time it will stabilize and it may not be connected to root source node at that point of time.

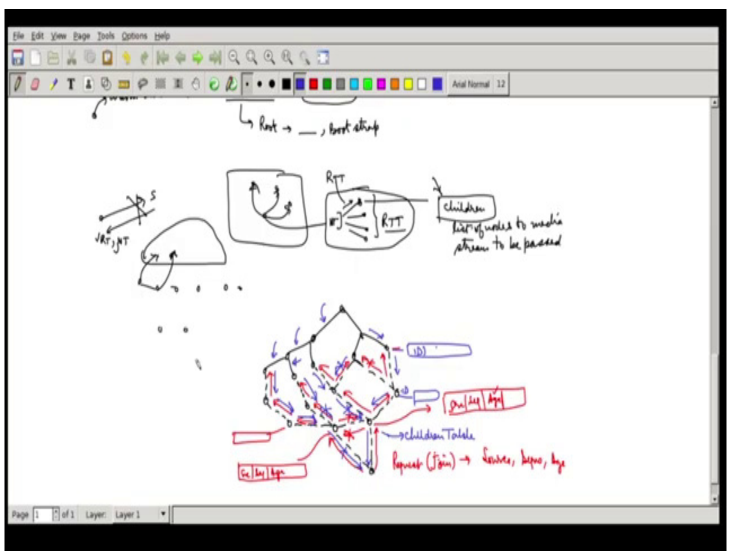
So, it may be connected to different nodes in different partitions, and may be connected to a neighbor table, which is going to be the closest nodes based on RTT. So, this part will ensure the connectivity this part ensures the proximity. And that is what we are going to use now for creating our stream.

Now, remember, all these nodes are already far actually interested in getting the media stream, that is why they are part of this DHT layer. So ideally when you will be joining so anybody you will connect with always have a stream, so whether it is your routing table, whether it is in your this thing. So, when you do keep on you actually optimize, you can find out which is the closest node and you can always connect to this. So currently I am talking about the reputation or any that thing, I am assuming that everybody is cooperating, so somebody asked the media stream node will always willingly, is willing to give the stream to him.

So, there is no problem in regard to that. Normally I would like to receive this team but would not like to give it because I have to pay for uplink and downlink bandwidth both. So normally that is what is going to happen, I would like to only receive, I do not want to give it and remember there is no payment mechanisms which have been talked about so far. The financial things can be thought of in the system. We have been already working on that issue in B4 design.

But currently we have essentially, if you are forwarding our stream to somebody else, you are actually paying in that sense because you are not helping the network to pay for it. So, you should get the stream because of that, so people have to be cooperative. I am assuming currently that they are cooperative, now how the stream will be received.

(Refer Slide Time: 15:11)



Worst-case scenario that there is a set of nodes they all join together at the same time, they all join together, but in that case, what is going to happen is, that they will be always first of all connected to some people who are already receiving the stream. So, they will connect to bootstrap and they will start to do it, then even this guy will also join, at some point of time these may get connected actually. But in the beginning in real life, when in real life, in actual implementation, you will never find that none of that is you cannot, there will not be a situation where none of these guys who are there in the routing table or neighbor table are receiving a stream, but will never ever be happening.

So, the idea is very simple with whichever, home you have the RTT which is least, you just send a request to him. Even you also maintain a children table. And then he will essentially start feeding you, you will say this is the media stream I am getting and these are the guys to whom I have to send. So, there is a list of nodes to whom the media stream has to be given, these are basically packets. So, you will start getting the feed. So, this is pretty straightforward thing.

So, the moment you connect to the source, source may not give it. So, you may ask, but he says I am loaded, so he may refuse. So, you can keep on asking from all of other neighbor table or routing table entries and at least one of them will start giving it. So, once you start getting the feed then at least you are listening, but in the meanwhile, you can keep on updating your RT and neighbor table. And then what you do is you find out if there is a better candidate from the current person from whom you are receiving party.

If the RTT from the current person is very high, you will tend to choose a person with whom the RTT is least actually, because that is more efficient for you. So, the idea is that the total number of average length of the links which are used in this broadcast, this sum total has to be minimized, that will lead to the most efficient use of resources. In a hypothetical case, I can take a scenario like this when we have a distribution happening. Normally this will be a worst case scenario this will never ever occur most likely.

So, then there are nodes which are connected but they are not receiving. So, these dashed lines are the ones, so these are trees, so these guys which are actually already connected with dark line, they are already receiving a stream. So, these guys are not. But they are all supposed to, they will all be trying. So ultimately, they will. So, even if the last guy I am here and I have only these two as my in my neighbor table or routing table, these are only entries which I have. So, I can send the join request.

Now, I am giving a hypothetical case this actually will ensure that, yes things happen. So, if it happens here, it will happen in the real life. Real life is far simpler; normally any 1 of your neighbors will always be receiving it before. If suddenly large number of people will tend to join, then of course, it will take some time before the network gets built up, but everybody will keep on getting successful, will start getting the feed, it may take some time before it happens.

So, in a neighbor table, I will just send a request, join request to both of them, but I will whenever the request will come I will tend to pick up from somebody who is going to have a better RTT. So that I will choose I will actually put fewer resources from the network to use for this streaming system. So when this guy will receive, it will say I have received from here, now in this case this source will send a request packet, did the join request this will always contain the source where you have it, there is going to be a sequence number and there is going to be an age.

When this will be received this will actually I am just doing a broadcast outing, remember. Here I will make an entry that this guy has sent the packet is the last sequence number which I received and this is the age, so whenever the age will expire I will purge this entry actually. And if I get a fresh source in a new actually packet which is have better sequence number I will replenish this age actually in that case. But, any packet which is going to have sequence number lower or equal will be simply discarded. So this ensures that duplicates are discarded.

Now, this guy will now send this packet to this side as well as to this side as well as to this side, because it does not have a feed, if it would have had the feed it would have sent the feedback directly to this guy by making an entry in case of a children table which would have been there. Then multicast requires multicast routing information base but I am calling it in simple language a children table.

But since it does not have, it cannot use this because not receiving the stream, it will simply broadcast. Now this not receives this same request second time. So this also maintains a similarly broadcast table which will say there it is I have already received a packet from this, with this sequence number this is the age. This entry would have been there when this packet comes; this packet will be simply discarded because you wanted to receive this same thing.

And this might also have sent it in the same fashion and this would also have been discarded here. But then there is a broadcast which is happening, the duplicates will be discarded and this packet, this request packet will keep on moving and every time this broadcast table entries will be updated at every node. So, this is what is going to happen, ultimately it will reach here. And here this might get discarded actually.

So, each one of these guys who are receiving feed gets it and they suddenly they figure out that yes, somebody is asking for a feed. So what you will do is you will create a, you build up a children table here and it will make this person's entry. So this will also start automatically getting subscriptions when this guy tries it out. There are too many nodes, even 1 guy doing it will also work but most likely they would have already initiated in what the feed back at the time.

When this starts, somebody will be there he will get figured out that we will get the feed and from there he will get the feed for itself. So it will make a child entry that this is a stream ID and this is the children. So this guy is whatever his ID that ID will be listed in the children table and whatever feed is coming that will also be broadcasted to him. From this side, feed is not going to come because this has been discarded actually being a duplicate request.

Because it came from the same source, if this guy would have made a request, this would not have been duplicate but this has come from here. So I, the basic idea is this need to get it to this particular guy. But incidentally in this case everybody will get the feed, whoever has asked for it. But of course, this link would not have been there this guy would not have got the feed you might have to start a separate search for it.

So because this guy would have been not requested, so he would not have been given the feed the feed would have to be taken by this guy from here actually. But Incidentally, I have taken this link. It is okay. It is the request will go here and feed will come from this side. So this guy will also give a feed to him, this guy will also give a feedback making the entries in their children tables. Now, here are actually there are two feeds which are arriving.

In this case, this node has to figure out which guy actually has, remember these nodes are in their neighbor tables actually. You send it to your neighbor tables; they will then further forward it to whichever nodes which are there in their neighbor tables and routing tables. It will now look at the RTT of these nodes. So if the RTT for this is less it will simply disregard this one and this one will be taken up.

Idea is very simple because a join request was sent. It created an entry in the children table. And we can what we have done is periodically you have to keep on sending the join requests if you want to receive the feed. In the beginning you will do a broadcast and later on you can only send it to the guys from whom you are getting the feed actually. This guy will not stop giving the join request here but we will keep on giving it to this guy.

Every few minutes or every few after an hour kind of thing, this will get replenished to entry will refresh in the children table and it will keep on getting the feed. Then it will be coming here, so feed will this guy will receive the feed only one feed so it will now. As already request was coming so it will now create an entry in the children table, it will send the feed here. Now there are two feeds, this guy will find out which is m lower RTT, so it has to pick up that person and the remaining will be discarded.

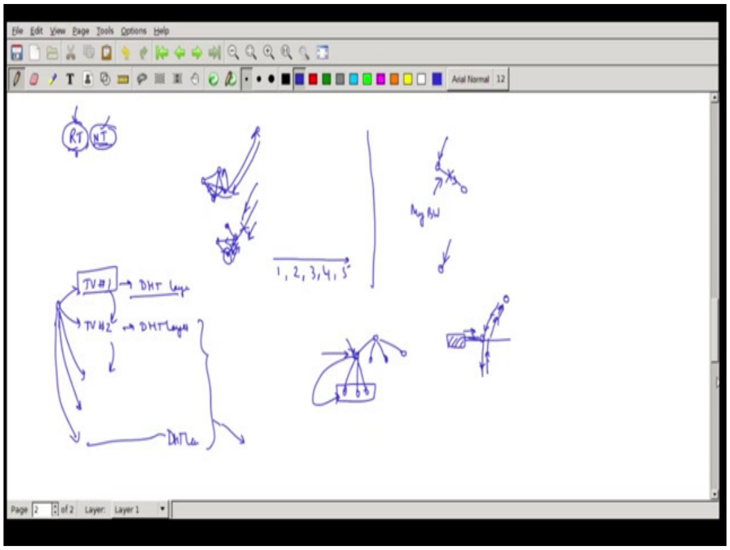
This feed will come from this side; it will stop sending the joint here actually. Once the joint is not sending and this guy does not want the feed which is not the case in our scenario because everybody need to receive the feed because they are part of the same DHT layer (())(24:54) actually. So this guy needs keys to keep on consuming. There is no need to simply disregard the whole thing. In that case, of course, our strategy should have been very different, I would have used a different protocol, which would have been something like a protocol independent multicast in that case.

But this is also fine, I do not have to maintain a routing table here, that is a beautiful thing. Through a broadcast I can actually create stop and so now the feed will be start coming on this direction, and then the feed will come in this way to everybody. So this guy now has to make a choice between whichever has got the lowest RTT, so this might discard this and this packet will come and this guy now has to make a choice of this will disregard this and this will come.

So you will start, you have now started getting the feed through this particular route, and then of course, as time goes by, you are only going to send joins in this direction. So everybody will only see the joins in those directions only from here you are getting the feed. Next time you find out somebody is having a better RTT you can send the join by identifying yourself as a source toward this direction. This guy has got a better RTT. Then the join and let us see what happens because he might also be receiving the feed after some stabilization.

So you keep on changing dynamically and ultimately you will have 1 single tree getting created. And if you look at the RTT on each of the links, sum total of RTT is for all active links will be minimum in this scenario; this will automatically will get optimized.

(Refer Slide Time: 26:34)



Now, it is possible that your neighbor table, if I am not using routing table, so in each node actually has a routing table as well as a neighbor table. So, maybe you can ask a question why not, why to use only RT, why not to use or I will not use RT, only use NT neighbor tables because they are actually proximity-based thing. So, I will actually always build up a best broadcast three, multicast tree so is but there is a problem here is there actually say 3 node, 4 nodes which are very close to each other. And my neighbor table has only three possible entries, three maximum entries.

There will be only maintaining entries here. And actually somebody who is a feeder is sitting here. I will keep on sending only if I do, if I only use NT, I will keep on sending only requests here and I will never get the feed. I should also send a request to RTs also all entries which are there in routing table. So that way you will always be ensuring that you are getting connected to the source because everybody is using both RT and NT. I will also be sending a request here.

If I do not get any response from here, I will get a response from here and I get the feed stuff and because other guys are actually talking to me I may start getting the feed to them, and once the feed is available, perfect. So, now people will try to optimize based on RTT, so but I will never be choosing because I am getting the feed from here and I am giving it to others, because nobody has responded to me earlier. Of course, I will find out that I am giving it to these three guys, I cannot get the feed.

There may be a situation that you are getting a feed from here, you give the feed to these two guys and this guy gives a feed to him and I find this guy is having a better RTT and also getting the feed. In that case I can send a join, and then the moment I will actually send the join so that I can get the feet. And you may even come from; you may start sending the feed back to me. But then there is a loop cyclic loop here, these needs to be broken in such scenarios, because, the feed can only come through this route.

So, one of the ways is when actually you are giving a feed, you should actually tell that you are, when you are feeding to others, and you are trying to join to somebody else it has to be disjoint path always. So of course, in this scenario, what will happen is when I join it here and I leave with this one, suddenly everybody's feed will get stopped. And in that case are no other options because this is not coming from anywhere else. I will again start doing are RTT thing and I will again connect to it.

And that is when I should figure out that yes, it is not happening. But if I actually keep it alive and I keep on getting food I may get the feed twice. So, whenever I am actually switching over and by switching off the earlier feed I get this, I am not getting the feed that time I should figure out this is the invalid thing, I should not use it I will only use this 1 and this will be simply disregarded even if its RTT is smaller. I have to only do with the other pools actually.

That way actually you can actually break the loops and there are various other methods also by which this can be done. So, this is one of the methods by which this is possible. Now another problem which will happen in this scenario is that you are listening to 1 TV channel and now you want to suddenly want to go to another TV channel. Now that is a complicated scenario. Because you have to now go through a different, there is a different DHT layer and there is going to be another DHT layer.

So, 1 of the ideas generally will be something says 1 to 2 to 3 to 4 and so on, because there is going to be overhead, time overhead after which you will be able to get the feed. When you are doing this, it actually means you should actually connect to multiple DHT layer and channel surfing is happening. So, while you are at TV1, I should actually have joined, the stream end should have been getting a feed for some of other DHT layer also. Of course, you are getting the feeds for all these layers, but you are only using this one.

And when you are getting feeds for these, you might end up in also giving it to others while you are not consuming it. So, these can be the scenarios which are feasible. That is the only problem which happens in this case. But this also can be taken care of to a certain extent as the people will not mind, anybody who is doing very fast surfing, for them there is going to be too much overhead, they have to run too much DHT layer also a lot of computing overhead and bandwidth will be consumed.

Which also happens in even in case of IPTV because you have to subscribe to multiple IP multicast layers. And you consume tremendous bandwidth when you are doing channel surfing. So normally, whenever you switch to another channel, you do not release it immediately; you actually engage it for some time. And when you are searching that time, you try to also subscribe to other one.

You have to look at the behavior and then based on that at the receiver side, you have to figure out a way so the user do not feel this channel surfing delay which is going to happen in the system. This is I think another complication which is there. We have looked at the loops, we have looked at this, now I come to an interesting problem, we call it free riding.

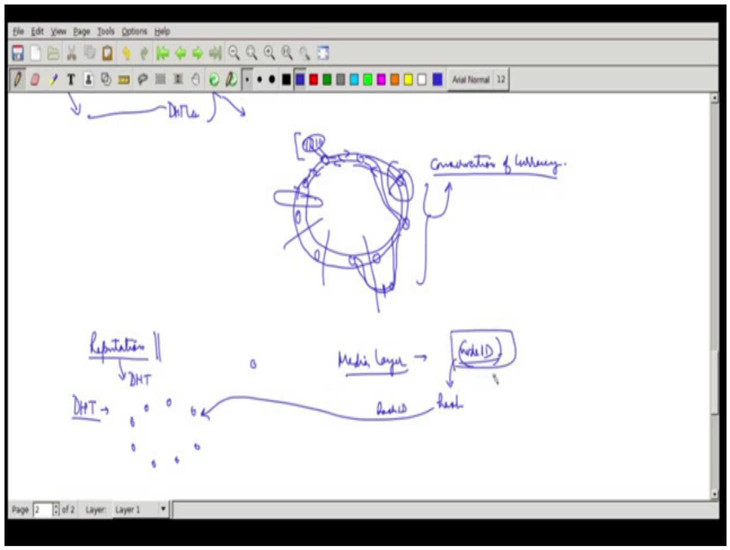
Normally if I am getting a feed I am happy because I am using it, now I should give the feed to somebody else this is consuming my bandwidth. And of course I am paying for it, I would not like so normally people will have a tendency, the most optimal strategy for them is only receive and see it, do not give it to anybody else. But that we our network will never take off because source only can give it to the limited people.

Now question is why this guy should actually broadcast it to somebody else. If I can make some mechanism by which there is an incentive to be here, because if he is not going to be here, he will end up in here and his quality of reception will be bad. He saves on the bandwidth, but his quality of reception will be bad because there will be more packet losses, there will be more paths. So let us look at this how this can be done.

One possible way which we thought of in the beginning is that we can make a closed loop economic kind of stuff. When you are getting a feed you are paying something, you are giving a feed to somebody else you also getting paid something. So whatever is the value which is deriving by viewing this video that is the only value which you are paying, now this value that add into this so you are paying a higher to the source.

That actually means there is a currency transfer who is required and digital currency transfer requires something like a blockchain, if you have to do it reliably. Or you require some central banking where whenever the transactions happen, both the entities will send the request and the bank will keep track of who actually holds what amount of currency that what point of time. So that could be the only way.

(Refer Slide Time: 33:42)



But the only problem which we see in any economic system, normally there will be a closed loop economy which exists always. So, for example, there are many entities, so you are transferring some value to him. So, you will transfer the currency to him. In that is you are getting some value from somebody else you are transferring the value, so the currency circulates actually, currency circulates in the economy and the value circulates in the difference, so everybody contributes.

Somebody who does not contribute gets bypassed, value goes directly and currency also goes directly, this requires means poor now. It is possible there are some nodes. Some of the currency, some part of the currency or value passes through this way. But there is a very small amount, but the sum total of this has to be equal to sum total of this. So, in fact, when you want to reach actually route maximum amount of this flow, it will only flow through you then you will become rich guy actually.

And of course, if you storing, start storing the resources then there is an issue because they have not been, the help taken out of the flow. So, economy actually contracts in this situation. So, this is also not encouraged. Now, the only problem here is that there has to be conservation of currencies, the total amount of currency has to remain same. Currency cannot be destroyed cannot be produced. And so implementing blockchain will be much more cumbersome. So, we have thought about it and ultimately we came up with an idea that we can use reputation in this case.

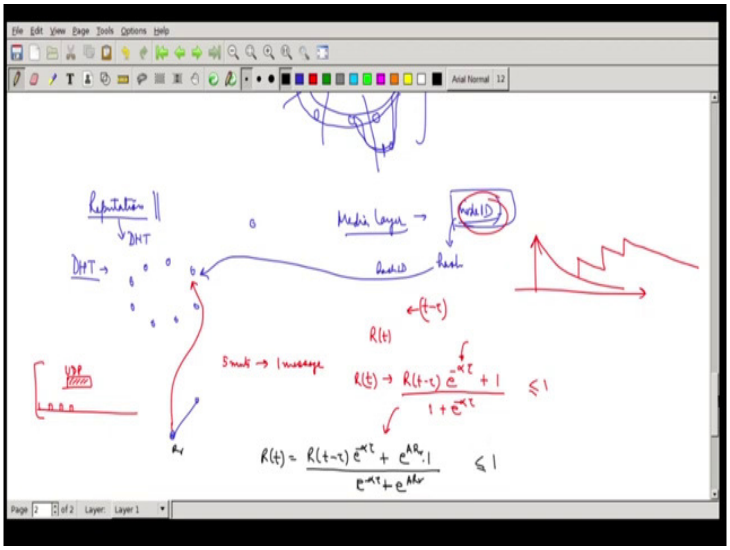
So, depending on a certain desired behavior, we can actually modify the reputation of the person. So, you have a tendency to gain a reputation because when you have reputation, you have an advantage when you are interacting with others, you always act as if so that you gain higher and higher reputation. And that way we can actually give an incentive and people will be willing to know, not only receive the feed, but will also give it to others.

So, we had thought of it, we are also actually doing it, though we are actually still investigating this how this can be done. But I find this is a far superior method, it is actually simpler to implement, and I do not have to maintain this conservation of currency principle. And then of course, there has to be equitable distribution. So, everybody has to whatever he is spending that much he need to earn in the scenario.

If that does not happen, somebody will go out of the network. Somebody will start dominating, that is not desirable for our system, if you want to build up a peer to peer network. The way it is going to happen is that if a node we will actually maintain another DHT network. Remember in storage DHT, when we had talked about storage layer, we have talked about storage DHT also and then we have talked about the reputation DHT at that point of time.

Reputation for that we will have a separate DHT, the same actually reputation DHT can be used here also but there is a reputation regarding the overlaid multicasting. And now we, so this the reputation DHT will have nodes, some nodes will be there. Every node who is there, who is actually participating in my media DHT layer, every node will have a node ID. So, I can actually take this node ID and then I can compute the hash of this and this hash of this node ID even without that this can be done but I would like to always do hashing because that gives a randomization. And then find out a root node for that, for the hash ID so you get the hash ID here. So, this guy is the person who will be holding the reputation of this person, this node.

(Refer Slide Time: 37:31)



So, the rule is very simple that whenever a node is supplying, so some node is receiving. So, whenever this node receives, it will now send a message to this responsible node in the DHT layer. Every five minutes it can send only one message. So, we have a restriction of that time, every five minutes one message, and this message will come here.

So, it will now update the reputation of this person. So, reputation is going to be something which is going to change with time. So, reputation at time t is this. So, I want to find out, let this current time be t, reputation at earlier time so for example, $(t-\tau)$, at that time the reputation was updated for this node, for this particular node I am talking about. So, there is going to be $R(t-\tau)$, but this reputation should decay.

So, we are always taking the concept, if you have a reputation this, if you are not gaining in the reputation, reputation will decay with time. If some positive message comes, the reputation updates, if more positive comes reputation updates, otherwise it keeps on decaying. If you are not active, you are not participating, you are dead, you have to start from bottom always and you have to gain the reputation by becoming more and more cooperative.

So, this reputation will decay. So, I will define a parameter alpha. So, what this parameter alpha should be we have to design appropriately. I will only give what happens if the alpha has small value, what happens when alpha is big. So, what will be a tendency of more depending on this alpha value? So this has to be network constant. So, this will be decayed reputation which will come. Now, this node has actually given an update to you. I will say this reputation will always be one because of this thing.

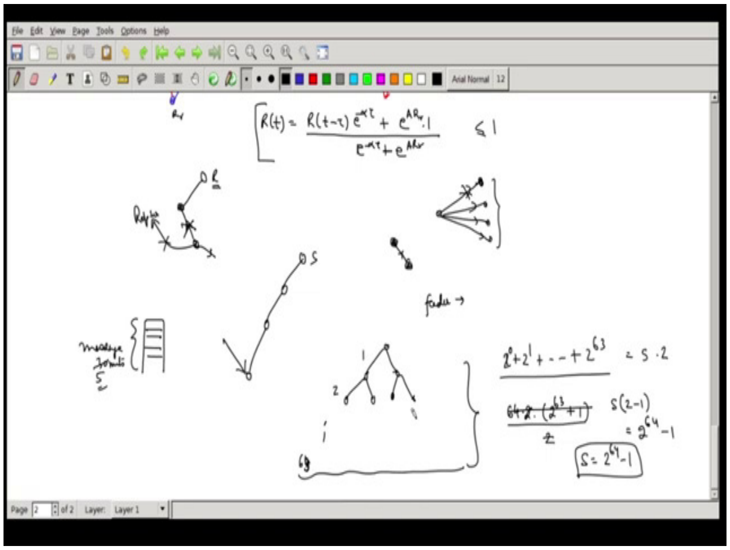
And I can now do weighing actually. So I can have 1+ e $^{-\alpha t}$, this will give me a new reputation value R(t) for this particular node, for this node. Now, this value will always be less than one. But interestingly, the problem is that this guy's reputation is small or high, based on that my reputation, if somebody has an having a reputation and he is getting a feed from me, I should actually have a higher increment. So, I will have a tendency to give my feed to a person with higher reputation.

So, that is the actually advantage you have, if you are going to have a reputation, you have a chance of getting a feed from somebody else who is having higher reputation than you. So you actually can be closer to the source, you will get the things in time, you will get more number of packets, there will be less amount of loss. Because more number of hops is there more the loss can be in there in the packets. Remember in the media we do not actually use TCP, we use packets.

So, it is like the UDP things which are UDP packets which are transmitted, if they are lost, they are lost. So you may be getting a lot of media, so if packets are lost, you will get that kind of a jerks in your audio and video media which is being played back. So, it is always desirable to be towards closer to the source, which is going to have highest reputation actually. Because you are going to get it and your reputation is higher, that is why he gains higher mark.

Let me change this formula further. If we change this formula to that R(t)will be equal to R(t-τ) $e^{-\sigma\tau}$ +1, I will not put up an exponential formula and I will put A and this will be the reputation of the receiver, the guy who is sending me the update. So this is reputation is now AR. So when it goes, so I will use this reputation, I will multiply it by 1 and then I will do R(t-τ) $e^{-\sigma\tau}$ +1 e^{-AR}. This value will always be less than or equal to 1 and it will always be greater than 0 obviously.

(Refer Slide Time: 41:50)



So, it actually means, if this guy has a higher reputation, I will again, I will have a higher gain, if this guy has a lower reputation; I will get a lower gain. Also, as a consequence, now what is it going to be the best strategy for a node. Node always wants to get a feed from a high reputation, there is very important, he can only get it if he actually has a high reputation, because when somebody is asking me, I am going to choose somebody with a high reputation only so that I will have higher gain in my reputation, because of this mechanism.

For you, for anybody to gain a higher reputation means he has to forward, otherwise he cannot gain high reputation. People have are the leaf node, they will have the least amount they will have zero reputation because they are not forward. Given an opportunity, they will start forwarding, as soon as possible somebody asked because they gain reputation. Then now within whatever options when they have in their neighbor table. If they find out not that in our situation, you cannot connect to anybody; you have to only connect to somebody who is in your neighbor table.

I am now also trying to optimize on the round trip time. I have a limit on how many can be there. I will pick up somebody else who is got a reputation; I will try to connect to him. If he accepts, it is fine, I get connected to him and leave this guy, this guy's reputation actually, and then overtime goes down because he is not giving the feed to me. And I am not sending update about him. Now, there can be situation when some node says, I am actually getting the feed, why I need to send up data about him, so it becomes a rogue node, he has a malicious.

So, this node is not sending to the reputation server, it does not send the message. So, we also do it because we have to do apply that 5 minute rule, so every node, every reputation server will maintain a queue of how many messages were received in the last say 30 minutes regarding the updates. So, anything which is coming from the same source is simply discarded. So, it has to be actually each five minutes. So, if this made not be 30 this can be five minutes only.

So, every 5 minutes if some somebody tries to send say 10 requests within 5 minute frame. All remaining 9 will be discarded, only the first one will be. I can always go and check periodically that whether the guy whom I am feeding is sending the updates or not to my reputation server. This can be checked by this guy. If he found he is not doing he can simply kick him out. That is it. So if you are not cooperative you are doing malicious things, you are out.

You can have a reputation as zero, you have to find out somebody else and you have to be honest, if you are not you cannot get the feed. So, this strategy is gained, theoretic technically what we have done we have actually proven that there is no better possible strategy for him to operate in the network except to be cooperative and forward whenever it is feasible. Otherwise you end up becoming a leaf node; you will go lower down in the topology.

The guy closer to source will actually have higher reputation; the guy who is in the bottom will be having lowest reputation. So, if you are not cooperative, you will not gain reputation and hence forth you will end up in becoming going down to the bottom. Of course this we do not have a dynamic topology in the current scenario, we are actually using neighbor tables and we are trying to steal ideas to whatever is coming, we will be optimizing the round trip time also.

But yes, we can actually now add more flexibility further and we can say that if I get more information because every, guy they are here are also changing neighbor tables. When I get neighbor tables from my neighbors I can actually use that particular issue, information also to try and try to connect to them actually, so they become my neighbors. I can now make what we call another neighborhood, I can actually no make what we call a separate tree structure distribution, where I can actually say who is going to be my feeder.

So, feeder holds and then of course, whenever my neighbor's table is coming from others I can find out if somebody is a better feeder with high reputation available and he is willing to feed me I can just simply keep on changing my feeder. And this feeder also I can use to get what are his neighbors actually if I can, if he is willing to share, otherwise anyway, I will keep on doing it again and again and again.

So, but this is the way we can actually create overlaid multi casting system. So this is going to be pretty much massively scalable and you have to understand that this is very lucrative because even if I do only source only gives it to two guys, these two guys gives further to two guys and this is the first hop, this is the second hop, by the time I do 64 hops, this is 63 hops. So, I will be having a huge number of nodes which can get the feed.

So, this is actually an extremely large number because what I am doing is I am doing this $(2^0 + 2^1 + \ldots + 2^{63})$. This value will become actually 2 dot, so this is going to be right, this number comes to be 64 into this divided by 2. So, this is going to be a pretty huge number Actually. This is going to be, I made a mistake here this has to be S multiplied by (2-1) also this will become to this total number of nodes which will be there is $2^{64}-1$, which is extremely large number.

So almost everybody, every computer on this earth has been counted, this number is going to be still larger than that. So, with 64 hops, I will be able to actually stream it to everybody. So that is actually become I think a very nice system and can be massively scaled up without using any servers. But we need the higher high end servers to actually start feeding each other, feeding all the users, end users start feeding.

Only thing you have the way you have to do is you are forwarding it to others and by reputation mechanism you are trying to reduce your delay. Lower reputation guys will be at the bottom always. So, with that we close this particular lecture. In the next lecture we will look at another interesting topic. Thank you.