Peer To Peer Networks

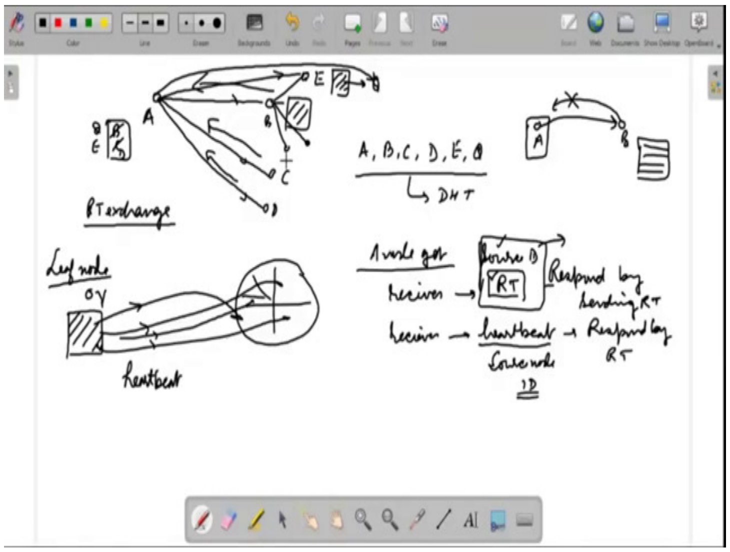
Professor Y. N. Singh
Department of Electrical Engineering
Indian Institute of Technology, Kanpur

Lecture 7

Static and Dynamic Partitioning of Node ID Space Fixed and Floating Partitioning

Welcome to the next lecture of the Peer to Peer MOOC. In my earlier lectures, I have discussed about how the routing table management actually will be done. Basically, all our routing tables will be updated. In the beginning, a node will normally connect to a bootstrap node and will get only the routing table or some neighbours' information from there and thereafter it has to build up the pointers to all other nodes in the whole network.

(Refer Slide Time 00:47)



In one of the lectures ,I have mentioned that if there are nodes, which are existing for example, A and B, so there will be, if A actually contains B, C or D as their neighbours, so A as a routing table only has these three entries. A will then essentially send its current routing table to all these people and because B, C, D will get the routing table, they have to respond by sending their own routing table back to them. B will do it, C will do it and as well as D will do it. So there is going to be a routing table (RT) action, RT action which will happen.

Another consequence, A will now come to know of, who all are the neighbours of this guy and if some other node is a better candidate to be put in the table. So it will then remove that entry and maybe replace that one entry by which is now come to its knowledge. Next time the routing table will be going here and this guy's routing table will now be coming back to him and then it may actually have information of say Q which is a better candidate so it may remove and put in the queue.

This is how it will keep on happening. Next time this Q will become the neighbour. So neighbourhood relationship will keep on evolving with time. And I mentioned that this will actually lead to an optimal routing table where everybody will have the correct routing table, but this actually does not happen. There is a reason for that we would like to explore that reason in today's lecture.

These two nodes A and B as of now or for that matter, all the nodes which are mentioned all A, B, C, D, E and Q which I have mentioned here, they are all participating in the DHT here distributed hash table. But there will be some nodes which may not be participating. For example, a node y which is not participating in DHT, but it is a leaf node. So know this can generate queries, and anybody can only contact only through its endpoint address, which is basically IP address and port number and transport, if that information is available. Through DHT nobody can come here but this can actually send the messages from DHT to other to other nodes. So that actually means it will also maintain a routing table.

This will be going in the whole node ID space which will get partition logarithmically like this. So this will be keeping pointers, various points, and the closest node in each partition, but how this will actually get created? This will also in the beginning will connect to the bootstrapping node and from that we will get a routing table, it will not send the routing tables to them, but it will send heartbeat. This heartbeat when it is being received by other nodes, they will be responding back by sending the routing table.

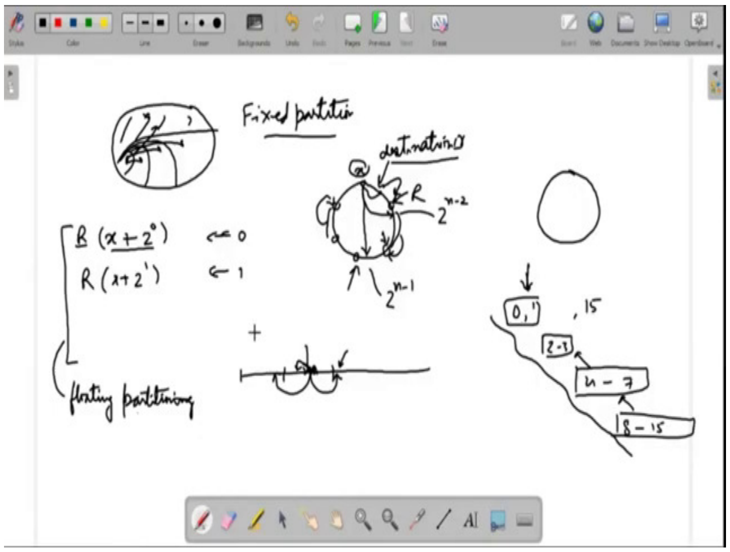
The rule is, if a node receives a routing table, it will respond by, the response will be, respond by sending routing table. And similarly, if it receives a heartbeat then also respond by sending a routing table. Now, only thing when you are receiving a routing table you are also getting to know about who has sent you the routing table. It is possible that A is containing B in the routing table but B is not pointing to A, so B does not contain A in routing table that kind of case can happen.

If B does not contain A in routing table when A will send. The information about A will be new information about B because B does not contain A. So it means the source ID here, will also will be used along with the routing table to update the local routing tables. Basically this new information will be used by the receiver to update the local routing table by better entries. Any entry which can become a better entry if it can replace something will be taken up from this, so this will include source as well as the routing table book.

While in case of heartbeat, the source node ID will not be used, source node ID will not be considered for updating the routing table. So that actually means the nodes the leaf nodes will never ever appear in anybody's routing table because nobody will be putting them in their routing table. And they never send the routing table so that is never getting merged, but only thing they will be responded by sending the local routing table.

This guy will also always have the updated routing table at any point of time. Now, let us come back to the situation which I mentioned earlier that even with just routing table exchange, there is no guarantee that you will get a stable routing table. This problem typically happens.

(Refer Slide Time 06:04)



In fact, I had mentioned earlier about logarithmic partitioning where I said there is going to be a node ID space. You are somewhere so we will find out the smallest partition then the bigger one, then the quarter and then half. So this is one partition, so you get one half, second half is further divided into two halves, so you are on one of them. You will be keeping one pointer here, one pointer here, one pointer here and one here.

But, when this node ID partitioning is such that it is independent of where you are placed. In that case we call it a fixed partition. Tapestry algorithm for example, uses fixed partition, Emilia uses a fixed partition, but Chord does not use. In cord what is way, the partitioning is done is, if your node ID is x. So normally I will be choosing some 2 raised to the power 0, I will find out the route node.

The route node in a cord is basically defined in this fashion; a Chord will consist of the node ID organized in a circular fashion. If this is your destination ID, so I can define that any node which is higher than this destination ID, but smallest in value in cyclic fashion so, this will be the route node for this thing. For this whole ring, this will be the route node, for this whole ring this will be the route node, so we can define route node in that fashion.

In Chord typically for a node x, I will define something called finger table. So zeroth entry, the first entry will be like this, the route node for x plus 2 raised to the power 0. Then the second entry which corresponds to one will be x plus 2 raised to the power 1, the route node of that. Technically what I am doing is I am now going diametrically opposite, this node will be the route node, so this value is going to be 2 raise power n minus 1, then this is the 2 raise power n minus, 2 whatever the route node that I will be picking up and then doing this.

Then I can do 1 by 8 so, so this is also a logarithmic partition, but the partition does not remain the same. I can actually do that, I can define fixed partition, I can say that 0 for example, in a 16 node case 0 to 15, so if you have node 1, you will be in partition 0, 1. There is another partition I will define 2, 3, then I find a bigger partition which is going to be 4 to 7, and then there is a bigger partition which will be consistent of 8 to 15.

Every time the partition size goes up and you are lying in this particular partition. This is a fixed logarithmic partition, but finger tables are not. In this floating partitioning like this we can call it a floating partitioning because depending on what is the value of x, the partition will change and the route node will change correspondingly, the entries will change.

In this case, for every node who is trying to look into 4 to 7 partition, the node closest either will be on the right side or will be on the left side if it is circularly organized thing, and that closest node has to be kept. This is very similar to which I told earlier that whenever this partitioning does happen, if you are here you will find out the closest node here and that enter you should maintain. And I also had to explained the reason for this, the closest node here and the closest node here and within its own partition all the nodes ID should be there with you, so that is a fixed partitioning.

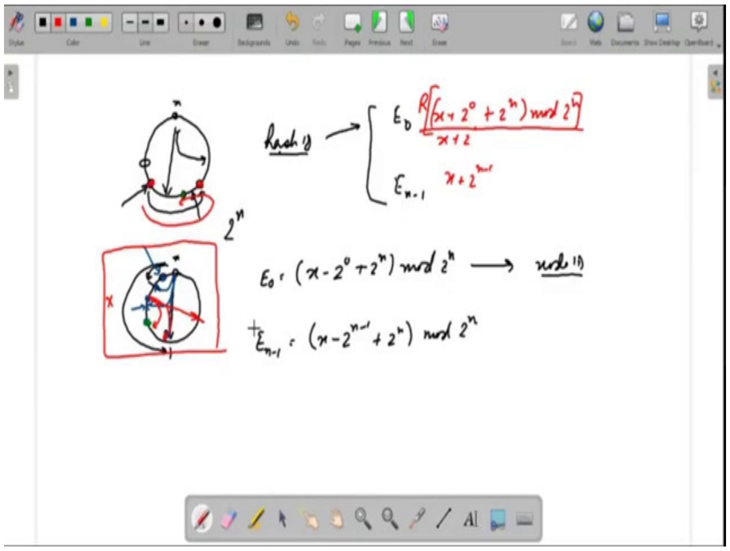
Then, the floating partitioning is logarithmic partitioning. This will actually change depending on the value of the floating partitioning case, depending on what is the value of x the partition will change. In such a scenario, the earlier mechanism of doing this neighbour table exchange will not lead to the stabilization of routing table; you may end up in incorrect routing tables.

Let me now talk about an example. Here I will show that it actually leads to incorrect routing table and how this can actually be corrected. This will also tell us a subtle concept of how we can actually use, in optimization strategy basically it is a gradient descent approach to actually correct the routing tables if they go wrong. In fact, even if you do not use routing tables, just use a component called leaf sets.

What is basically local neighbourhood information where everybody only just maintains that and exchanges that, it should be possible to always build up the correct information. You can do still the routing but the number of hops required will be very, very large. Routing tables logarithmic partitioning routing tables act like a shortcut. While in case of fixed routing tables, this local information has been guaranteed by way that you actually maintain the information at every node.

So within your partition, everybody will know, this is from your neighbourhood partition the closest node. So you always have everybody knows about their most local nodes, the locally closest nodes are available with everybody. And whenever you are going to do the exchange of this information, this will always lead to the correct structure. Now, this actually does not happen in the slotting partitioning method. So for Chord , this is actually what we call finger tables. So these finger tables will not give you the correct value. The correct stabilization will not happen. So let us take a case where this actually happens.

(Refer Slide Time 12:00)


hash id
E_0 $R[(x+2^0+2^n) \bmod 2^n]$
$x+2$
E_{n-1} $x+2^{n-1}$
2^n
x
$E_0 : (x-2^0+2^n) \bmod 2^n$ → node ID
$E_{n-1} : (x-2^{n-1}+2^n) \bmod 2^n$

So, if you are here, I will be now keeping a diametrically opposite number and whichever is the first node after that, that I will be keeping as the node which is going to be closest. Now the problem with this kind of thing is, next partition will be this, so I might be picking up this one, so there may be node sitting in here also. So now suppose if you have a hash ID, normally the routing rule is very simple. Routing rule is, I will now look into all finger table entries, I will now find out from hash ID the distance to these entries one, with the entry E1 to E_{02} to E_{n-1} entry, if their 2 raise power n is the node ID space. I will pick up each one of them.

If it turns out that I am closest to this particular entry, which is currently these are my entries which are going to maintain in the finger table, now this one is not in the finger table. So in this case, what will happen? If my hash ID happens to be somewhere here I will now search for this. I will find out this is the closest guy, but this guy is not route node, route node is this.

Normally, it is a good idea that you will always find out the predecessor. You find out whichever is the closest ID, reach there, and this guy now should hand over the packet back to either this guy, and this guy will do the next search. So at least this guy immediately next guy, this will be the first entry and you will be able to hand it over to him because this will be the closest from the hash ID, so that is the only way it can be done.

But there is an alternative way. You can actually choose routing table in this fashion that you can actually have your x here. You can actually now make the tables the first entry E_0 entry will be $(x-2^0+2^n)$, and of course you should add the modular things so that you always get a positive distance. Similarly, you can get now E_{n-1}. I will be now keeping, so these are basically the pointers where actually I will be going. x will be pointing to this if I am going to do $(x-2^{n-1}+2^n)$.

I am actually going halfway across here. Actually now going 1 if it is 2^0 the first case, so whichever is the extra node to this. So from this point E_0 whichever node is closest, from this point to this node the distance should be minimum . If there is a node sitting in here, this will be my route node for this, but whenever 2^0 is 1, so either it has to be a previous node immediately x-1 or it has to be x, there can be only two entries. If there is something else you do not have x minus 1 node, x itself will be the route node in that case.

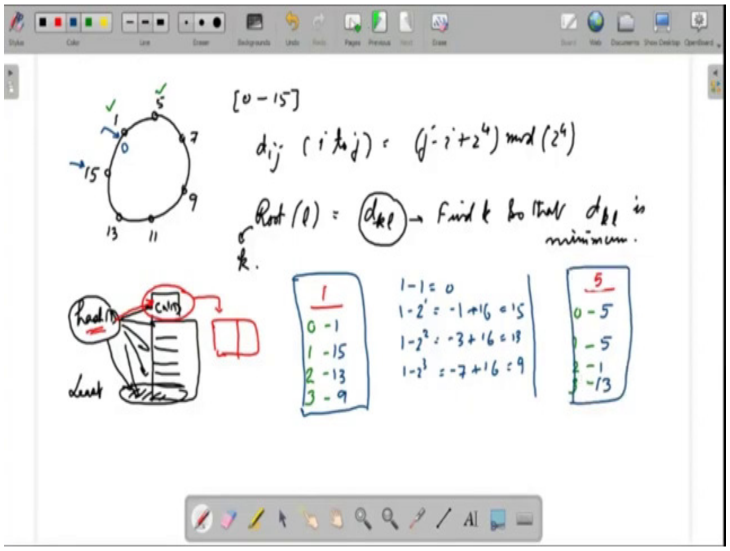
If I use this kind of pointers, so these pointers will be something like this. I will be pointing here, I will be pointing here. Now, when I compute hash ID to this distance. If I compute has ID to distance. If this is the guy is the minimum so I will end up in coming here, so I will end up in being here. And this is the next node which is present, so this is the route node. So my hash ID is present here and hash ID is closest to this node so I will hand it over the query to this guy. So this guy will further now search for the previous node, if the hash ID and in between there is no node, this itself will always reach to this person when I am looking for the routing table for this node, so this node will contain a routing table of this kind. So the route nodes like this.

I will be always; I do not have to change. I do not have to do this backtracking which I did here. I reach. I gave the query to the closest guy and then which gives it to the predecessor which then again do the search. So this was required when I was using x plus 2 raised to the power 0 x plus 2 raised to the power 1 and x plus 2 raised to the power n minus 1 and then computing within this the route node thing. So, when I was keeping the entries like this, then I was moving in the, I have to do this backtracking and then search again. In this case this will not be required so I am going to consider this particular scenario. This had already discussed in one of the earlier lectures.

Let us take an example and see what happens, whether I get the correct routing table or not when a new node enters in. When new node enters in because of that routing tables of whichever nodes will get affected has to be updated and we will see if I actually do this, this is not going to happen.

And as I mentioned earlier, this local information which I have, this local information is very important. So if you know everybody in the neighbour then it is always going to converge. So we will actually find out the method that how this local information can be incorporated. So I will actually use something called leaf set for doing this local information. So first of all, let us build up this example one by one.

(Refer Slide Time 18:21)



So I am taking a cord, so cord is going to contain node ID 1, node ID 5, node ID 7. This is an example, this can be done with any example and you will find that most of the time your routing table will not converge to the correct value action, if you are only using routing table action. And I will give you the reason why you actually end up in right routing table when you create something called leaf set.

A local neighbourhood information of every node is also being maintained. So in this case, this is a 16 node table, so my node ID goes range space is 0 to 15. So I need to build up a stable routing table. So as I told you, d_{ij} so for distance from i to j will be given as ($j - i + 2^4$) in this case, more 2 raise power 4 that is for our example.

And root node of hash ID l will be d of k l, this need to be minimized, find k so that d_{kl} is minimum. So that k will be the root node in that, so the root node of this will be k. So under this example, let us now build up the table. So we have to build up the routing table for everybody.

Now the routing rule which is going to be used is also very simple as I mentioned earlier, that you will now look into the finger, if you have a hash ID which has to be routed, you will look all the finger table entries, all node IDs which are mentioned there. You have your own current node ID, we will now find out the distance to current node ID and distance to every entry.

Now, whichever entry is going to give me minimum the least, we will just pass on this query to that guy. So, while this query is being received by this person, again it will do the same thing and at some point you will find out hash ID is closest to the current node ID. So, current node ID will be the route node. So now it has to look into its own indexing server and find out the key value peer corresponding to the key which is being searched, or whatever query is coming this current node has to handle that, it need not be always a key value peer search.

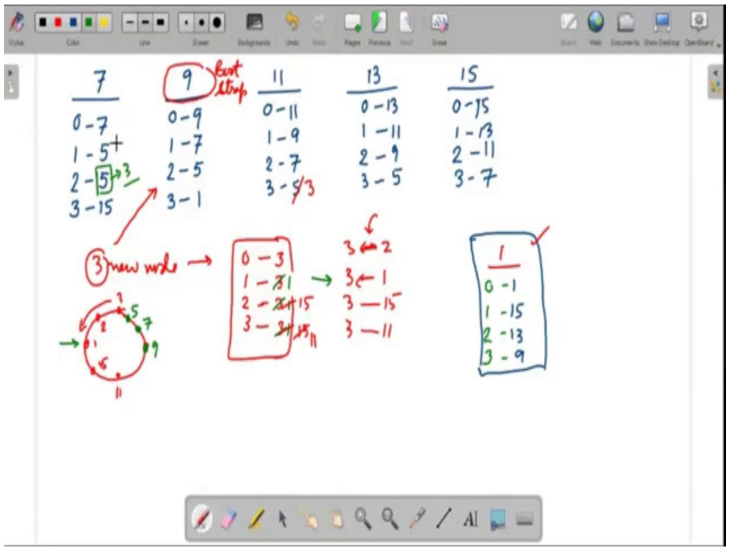
So, now let us look at how the routing table will be done. So, for node 1 for example, how the routing table will be done. So the entry will be, there will be 0 entry, 2 raise power 1 and 2 raise power, so there will be only 4 entries there. So for 0, so 1 minus 1 will give me 0, as 0 does not exist as actually you can see, so 0 will be somewhere here.

So the route node will be 1 as per the definition, so I will write 1 here. With the first entry it is going to be 1 minus 2 raise power 1 so it will be minus 1, so there is no minus 1 here so I have to add 16 which will be 15 so 15 does exist. So for there actually I will be directly putting 15 because that is the closest entry for 15, 15 I am looking for.

So, similarly 1 minus 2 square, which will be minus 3, so minus 3 plus 16 will give me 13 so I will have entry 13 here. And similarly for 3, 1 minus 2 cube which is minus 7 plus 16, it will come out to be 9. So this is what will be your entries for the node 1. In the same fashion, I can actually now find out the entries for node 2. Node 2 entries will be, so there is no node-2, actually we have to find out, we have done for 1, so we may have to do it for 5 so I should write here 5 actually. So for 5 when I am going to write the entries so, I am just going to after this, write down all the entries which are there.

So for 5, it will be 5 minus 1, which is 4, so 5 will be the entry which will exist here, 5 minus 2 is 3. So when I am going to insert a node 3 it should become 3, it will not actually become 3 to routing table action depending on which bootstrap node it chooses. So I will be choosing 9 here. So 5 minus 2 will become 3, so route node will be 5 here, 5 minus 4 is 1, so 1 it does exist, so 1 will be here, 5 minus 8 is minus 3, so minus 3 plus 16, it is going to be 13 actually, This will be the next entry. So one entry I have got this for node1, this will be for node 5.

(Refer Slide Time 23:56)



I can now similarly build entries for node 7, so I am just going to write it down. For 0s for 7th I will get 7, you can actually check. So you have to remember this figure, there is only 1, 5, 7, 9, 11, 13 and 15. So 7 minus 1 is 6. So, there is no with node, between 6 and 7 so 7 will come. 7 minus 2 is going to be 5 so 5 is going to be present. 7 minus 4 is going to be 3, 3 does not exist again 5 will repeat here, and for 3 I will be repeating 15. Similarly, now doing it for 9. So for 9 it goes to 9 because there is node here. For 1, it is 7 minus 9 minus 2 is 7, so 7 does exist. 9 minus 4 is 5, 5 does exist, and 9 minus 8 is 1.

In fact, you can see the pattern the way the node IDs have been chosen. So 0 will give you 11, 1 will give you 9, 2 will give you 7, 3 will give you 5. Node number 13, in the same fashion, 0 will give you 13, 1 will give you 13 minus 2 is 11 so 2, 13 minus 4 is 9, 13 minus 8 is 5, and there is the last node which we have, so 0 will go to 15 so these are finger tables which have been created, but finger tables have not good enough, so 1 to 13, 11, 1, 3, 2, 7 so my routing tables are complete.

Now, this is table routing table. So, if 11 and 13 will exchange the routing table for example, So,11 will come to know off 11, 9, 7, 5 is already knowing, it will come to know off 13 and 11, 9, 5 is already here so, 13 is only new information, and 13 is not a better option for any one of the entries so this table will remain as it is. Similarly, 13 will only come to know off, 11, 9, 5 is already there, 13 is there, it will only come to know of 9, 7, only 2 entries and only 7, and 7 is not a better option anywhere, so this will remain stable. Whatever exchanges you do currently everything will remain stable. So that is a steady state routing table. If there would have been an error it will keep on changing till it comes to a steady state.

You exchange with anybody the routing table, in fact, whatever entries you have you exchange with them, you will end up in getting the same entries ultimately, so local entries, existing entries are still the optimal entries, so that is a steady state scenario.

Now, let us take the case when node 3 is being inserted into the system. So, node 3 is going to be inserted. So this is a new node, but how it will actually find out routing tables? So normally in every DHT network there is going to be a bootstrapping node, an initial node from which it will get the information.

So in this case, I am actually assuming it 9 years of bootstrapping node. The bootstrap will happen from here. So when 3 will get created, 3 will have a routing table of 0. It will say this is the only node known to him, only node known to him, only node known to him. So, all entries will be 3, 3 because there is only one guy. So when it will now, it will be told by the authentication server that 9 is a bootstrap node so it will communicate to 9. So 9 will get a routing table which is going to be this.

9 will come to know that there is a node 3 which is available here. So, what it will do? So, whether can something be better than this? So only thing place 3 can get placed is between 5 because after 2 it is actually 7 so it has to be only somewhere here I need to check. So 9 minus 4 is 5 so it is 5, 3 cannot come here. So, 3 actually is not going to be incorporated, this table remains unchanged.

But what happens to this table? So 3 will come to know of that there is a 9 which is existing. I have to see this, 9 be placed anywhere. So, 3 minus 0 it will remain there, 3 minus 1 it is 2, it has to be something which is going to be more than 3 and less than whatever number you are getting. So 9 actually should come between those two rings, between 3 and whatever number you are getting. So your 3 and this number is 3 minus 1 which is 2, so 9 does not lie here. So 9 cannot be placed here.

This number is again 3 minus 1, so 3 or 1 again this does not lie so 9 cannot be placed here. So this is 3 minus 4 which is going to be minus 1, minus 1 plus 16 is 15. So 9 cannot be again placed here, because 9 does not exist in between. And this will become 3 minus 8, it will be minus 5 plus 16 this will be 11, so between 11 to 3 also, this does not lie.

This is actually I am going from this node to this, because I am going in a backward direction in the table, I am actually going in this direction when I am computing this minor section. So in this range you are not getting it 9 is always on this side. If I am looking at 3, so 9 is somewhere here, so initially this has to be 2, this has to be 1, this has to be 15 and there has to be 11. So 9 is always here somewhere. So it does not lie anywhere so none of these entries will get updated.

So 9 is not going to affect here. Now 7, 7 is somewhere here, so this is 9, then 7 is somewhere here. Again, it is not going to impact anywhere, 5 is here so 5 is also not going to have any impact. 1, yes 1 is going to have an impact because 1 lies somewhere here, so 1 lies here.

So, let us see where the 1 will actually get impact. So, 1 will impact actually at this place. So when this one 2 raise power 1 is going to be used. So 3 minus 2 is 1 so node 1 will get replaced here. So you will have a 2 here in this, in this case sorry it is 1, we will get a replacement of 1 in this case.

Now node 1 is known so now what will happen to this entry? This entry will also be 1 because 3 is not there, there is another mode which has got inside, so for 15 and 11 also this will be happening. 3 is still not aware that 15 and 11 are existing because these were not there in this routing table so I will get 1 at all the 3 places. So in the next step, I have 3, 1, 1, 1 so again next time if 3 exchange is the routing table with 9, no change is going to happen.

Now 3 has what all entries with itself. let us see. So now 3 is going to, we have only these entries available and nobody else is 3 so 3 will only send a request to 1, 1, 1 only. So 3 can only do the exchange with routing table 1, so routing table 1 is this. So I will use this 1, 15, 13, 9 on the next one.

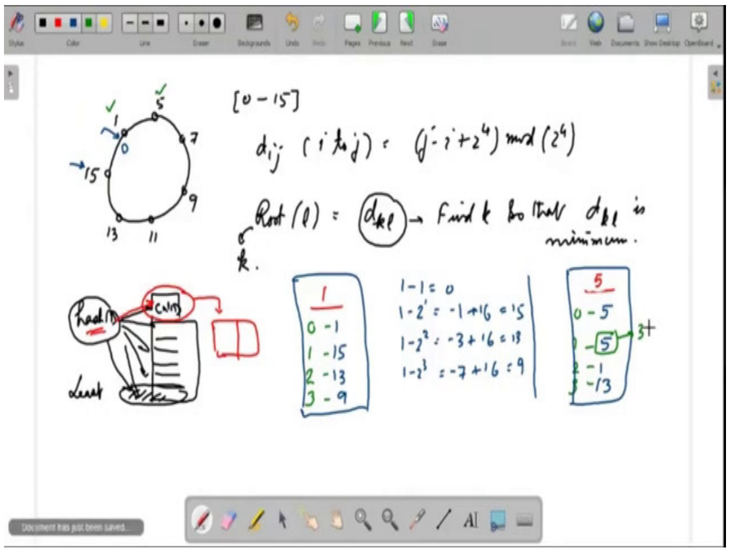
Let me just write down. So this is the node 1. So, now 3 will be now doing exchange with this so it will now get information of 15, 30 and 9. So 15, 13 and 9 so in this case there will be a change which will happen which will be in the routing table here. So, this will remain 1 because 1 is already known, knowledge about 15 will come so 15 will be coming here, and here it will be now coming as 13 in the next step.

And in the next step it will now be talking with 13, it has to talk with 15 so it will become aware of 11. So, ultimately it will also replace this entry, which is there sitting here also by 11. So, they will get 3, 1, 15, 11 which is this table entry for 3. But now if it actually keeps on exchanging information with 1, 15, 11 so 1, 15, 11 there is not going to be any change. If this will come to know of the entries from 3, 1, 15, 11 there is no change, this tables remain as it is. At 15, 13, 11, 7 no change is going to happen, 13 will get the information no change is going to happen here.

As a consequence now, you will reach to a stable state. So only place where the change will happen is when it is going to talk to 11, 11 will come to know of that there is a node 3 because node 3 is the source and 11 was the entry. So, it has communicated the table here. This 5 will become 3 that is the only change which is going to happen, and after that table will keep on just working as it is, but now there is a problem actually.

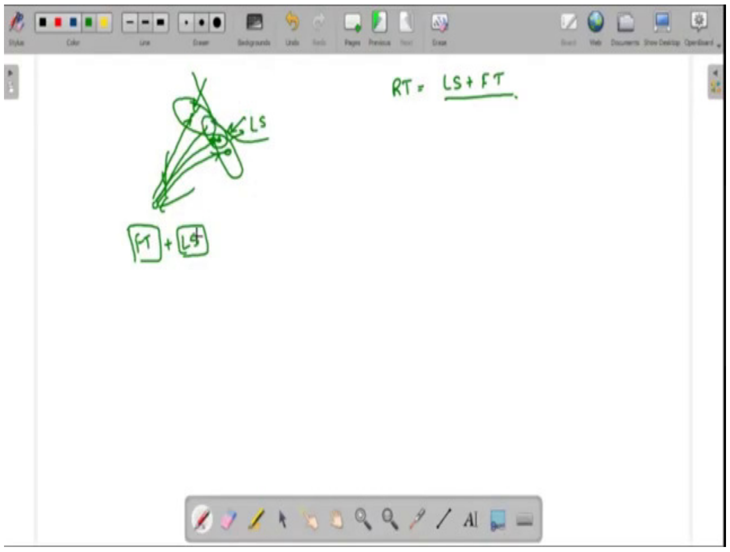
Problems lies here, the change which has not happened is, one is here, so here the entry should have been actually 3, but there is no way 3 entry could have been made here because 7 has not interacted anywhere. So even if now 3 keeps on doing routing table exchange, this will remain 5 this will not become 3, so this has to be 3. So 7 minus 4 has to be 3 so but it is 5 here, so this change will not happen.

(Refer Slide Time 35:13)



And another place is the routing table of 5. So 5 minus 2 have to be 3, but this remains 5, this also should have become 3. This 2 places it will remain wrong. How to sort out this particular issue?

(Refer Slide Time 35:33)

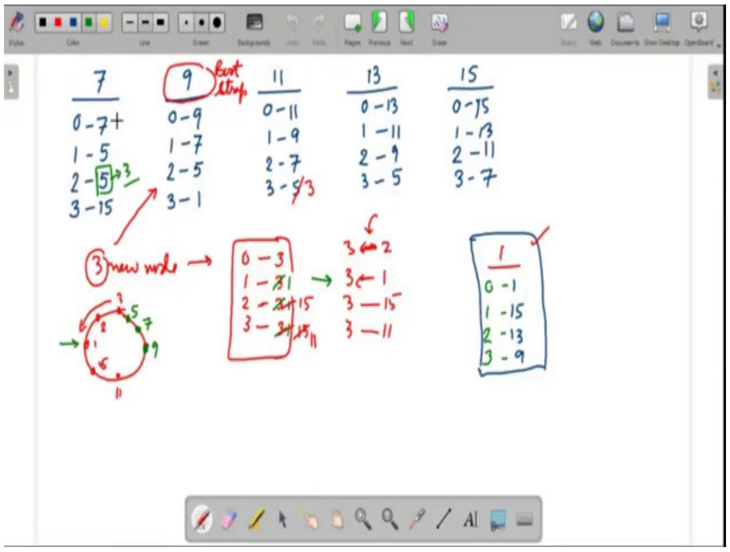


Now the reason for this, which I can explain is that every node is pointing to some other node, so there is a pointer that is what is happening. So when I am looking at a node 3, node 7 for example, it is pointing to 7 minus something to node route node. For whatever node 5 if there is neighbourhood information available, so what we will do is, each node will now also maintain neighbourhood information; we call it a leaf set. So complete routing table is nothing but leaf set plus finger table.

This routing table is what is going to be exchanged with everybody and everybody will now not only when they are going to merge from the receiving tables, When a node will receive a lot of these tables from others, it will not only actually take care of finger tables being updated, it will also now updated the leaf sets also correspondingly. The moment we do it, we should be able to correct this particular problem. Let us see what will happen if I am going to use the leaf sets.

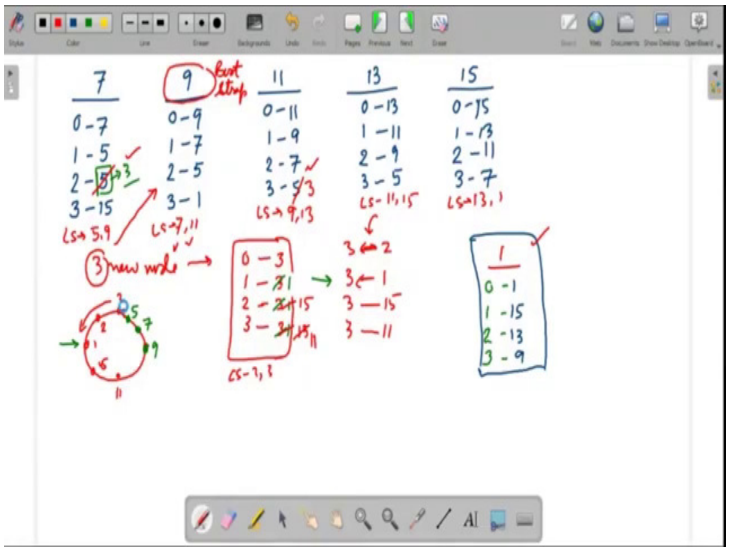
The reason for this is now if I have leaf sets. So if there is a better pointer which is there. I will now start pointing to this. Next time this guy will tell its own leaf set. I can now keep on moving and ultimately I will reach here, now more never gives me a better option so this is now the most optimal option. I am able to actually now slowly move just like going through a gradient till you find out an optimum. So this is like a gradient descent method, basically to find out the optimum entry in your routing table.

(Refer Slide Time 37:18)



So in the previous case when we have this routing table and I was not able to optimize 7 and 5, so rest everybody seemed to fine, but what 7 and 5 will do if I start maintaining the neighbourhood tables. So let us see what will happen with the neighbourhood table. So I am going to add something called neighbourhood tables in each one of those entries now.

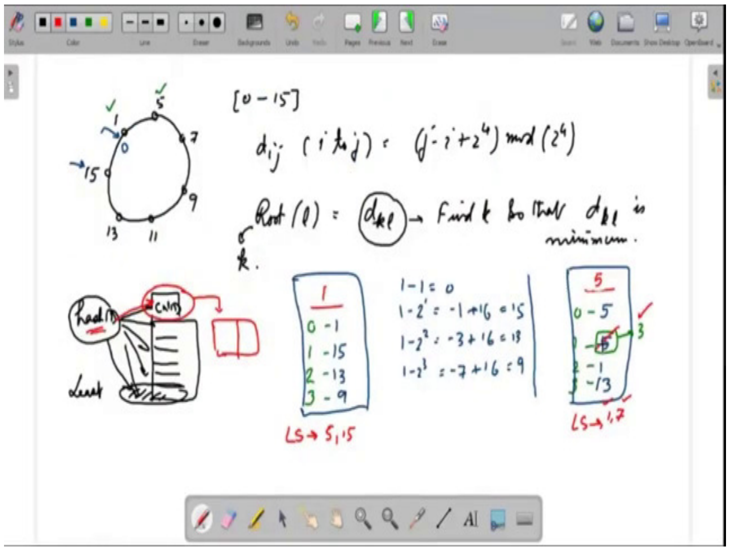
(Refer Slide Time 37:39)



So the neighbour table or leaf set here, so who are the neighbours of 1? I am just keeping 5 and 15. There are 2 more entries which I am going to make. This leaf set is going to have now, I am only keeping predecessor and successor, so 5 will be maintaining 1 and 7. Going to the next one, this leaf set will be maintaining looking at the diagram 5 and 9, 7 and 11, this leaf set will contain 9 and 13, this leaf set will contain 11 and 15, this leaf set will contain 13 and 1. So when 3 will come it does not have a leaf set so it only knows it is the only guy.

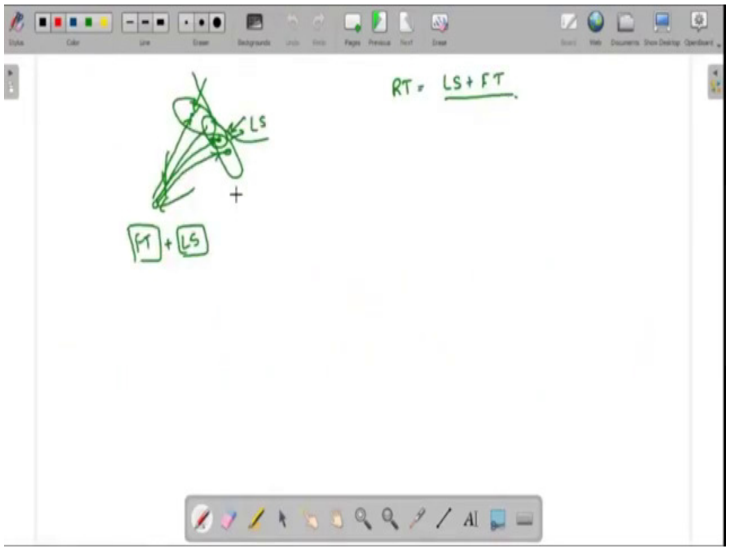
When to talk to 9, when we need to talk to 9, now it will get information of 7 and 11 also. So it will send its information to 7 and 11 also in this case. But they will not change because they have no better option when you send to. Except for 7 the information about 3 will come, but leaf set will not change but it will now change the entry here so this will get corrected, because 7 will be now available to him actually. So, 11 will be coming from him, 11 will anyway will come to know of now in 1 step itself about 3 so this will get changed in the first step not in second step.

(Refer Slide Time 39:41)


[0 – 15]
d_{ij} (i to j) = $(j - i + 2^4) \bmod (2^4)$
Root (l) = d_{kl} → Find k so that d_{kl} is minimum.
1-1 = 0
$1-2^1$ = -1+16 = 15
$1-2^2$ = -3+16 = 13
$1-2^3$ = -7+16 = 9
LS → 5, 15
LS → 1, 7
Leaf

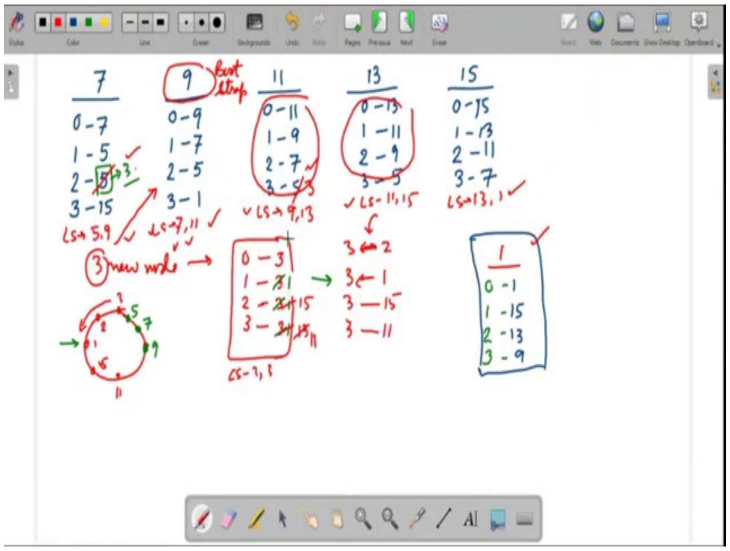
Now let us come to what happens to the 1, the 5 so 5 as 1 and 7 as the leaf set. So 5 will next time will talk to 7 because 7 are the neighbour is given here. So when it will talk to 7, it will get updated information about 3 and it will also update itself and this will get updated.

(Refer Slide Time 39:48)



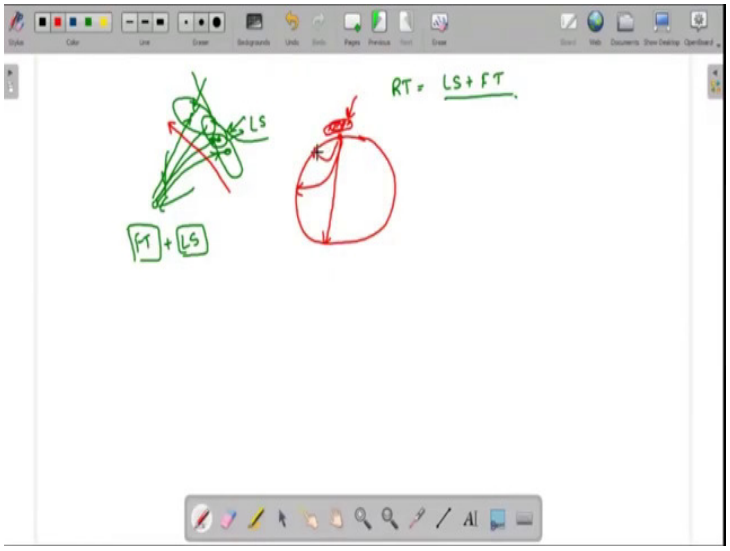
So using this particular principle of leaf set and following this local hood information and moving towards optimality will always lead to the optimal routing table. In the first step itself in fact, very few steps I will be able to converge much faster. So alternatively, even if you do not keep these leaf sets, these routing tables or finger tables, but you keep these leaf sets, you will still be able to operate the whole DHT, but it will take longer time to search.

(Refer Slide Time 40:46)



So most likely what will happen is people will be only maintaining their leaf sets by accurate talking to their neighbours. If I know my neighbour's neighbour, and if he is closer to me than my neighbour, I will just replace him and will then find out who is his neighbour actually. So this neighbourhood information essentially can be used. So unstructured of networks, technically use this but they replace their neighbourhood on different conditions, not on the node ID proximity. If in the unstructured network, they start doing it with node ID proximities then certainly yes, we will be able to do a better routing.

(Refer Slide Time 40:51)



So with that, let me close an important thing that how the routing tables is actually updated. The leafs sets do actually form an important thing. And if you are going to use anything which is organizing things in circular fashion, and they actually use something like this partitioning, it is better to use a leaf set. In fact, in general whatever routing table management to use, using a leaf set local information is a good idea. And this also become part of a routing table. So you also distribute this leaf set information also to all your neighbours in routing table as well as in leaf set. And whatever you are receiving based on that you update your leaf set as well as the routing table both.

I think this should be used everywhere because this guarantees the stability of the routing table, whatever you are making. But if you are using fixed logarithmic partitioning that also does guarantee the convergence of the routing table at every place, but in a dynamic partition, this actually is not feasible, you require the leaf sets.