**Peer to Peer Networks**

**Professor Y. N. Singh**

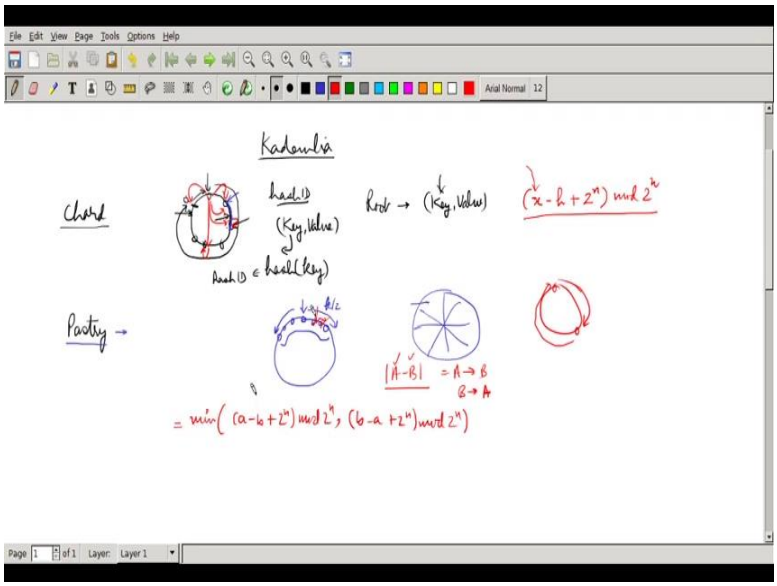
**Department of Electrical Engineering**

**Indian Institute of Technology, Kanpur**

**Lecture - 10**

**Kademlia: A DHT Routing Protocol**

(Refer Slide Time: 00:14)




In today's lecture, we will be talking about another DHT algorithm, which we call Kademlia. So, earlier in the lectures, we have seen a chord. So, in the chord, we were organizing all the nodes circularly, and it was such that whatever is a node ID, the node ID is increasing in this fashion. It is circular, so it goes back to the maximum value and then comes back to somewhere.

The maximum value will be here, so then there will be 0, a circular drawing. So, we arranged all the nodes circularly, and then we find out whatever hash ID for which I am searching the root node. So basically, we are trying to find out where the key-value pair is going to be stored.

So, we have to search it based on the key, so we compute the key's hash of value. So, this is what we call hash ID, and we find out if this hash ID is positioned here, in case of the chord because

this is an increasing node ID is going in the clockwise direction, so this node is going to be now the root node for this hash ID. And this node is the root node that is supposed to be responsible for storing this key-value pair.

So, searching for this key will compute the key's hash value and send a query to the root node. So, which, because of the chord routing, which we have discussed earlier, we ultimately will come and terminate here, and this will look into the table and search for key-value pair and return that value to the person who has sent the inquiry.

So, that was the chord, but in this case, there was a problem that these nodes can be physically is very, very separate, very far off on the internet. There was no way I could have optimized it. So, maybe this guy is sitting in India, the next guy is in the U.S., then again India, so you will be traversing to the internet too many times, so this will be very inefficient.

So, Pastry changed this particular thing. Chord was using two basic concepts; every node is always keeping information about who is my successor and who is my predecessor. Every node kept these two pieces of information. This is good enough for routing, but not this may take a longer time; this has a number of hops required to search a query will be almost n by 2 if n is the number of nodes in the network.

So, then we use the concept of finger tables. So, we are halfway across whoever is the root node; we are keeping that; one fourth across whoever the root node, we are keeping that; then one eight, and so on. So, when we want to push the query, we look at these entries and give them to the best possible option to whom it should be given.

So, if the query is lying in this range, in this case, it is lying here. So, in this case, the best is always to pass on the query to this. For any node, the node ID is highest but is smaller than the hash ID; the query is always forwarded to there.

And every node, first of all, will check whether the query lies between the successor or not and me. If it is between the successor and me, it will just hand over the query to the successor. And the successor will, every node will also check when the query is coming from my predecessor. If the query is being handed over to me by my predecessor, in that case, I should be the root node; that is the way it is going to figure out. And then, it will search into the database, and we will find out the key value. That is how the chord used to work.

The Pastry then invented went one step ahead. We are not going to use this kind of mechanism; we will use, in this ring, every node is going to maintain some k by 2 nodes that are immediately larger than the current node ID. Again, the organization is still circular, and it is also in a cyclic manner, the way it has happened in the chord and k by 2 on the lower side, which is smaller than the current node ID. This is what we call leaf set.

So, this predecessor and successor is a leaf set with the value 1. So here it was, k by 2 for a faster movement. Another thing which they did was they started keeping pointers based on how far you are actually from the node. So, we had discussed in the earlier case. The idea was if the node ID space is now partitioned based on that value of the node, so specific digits are based on that.

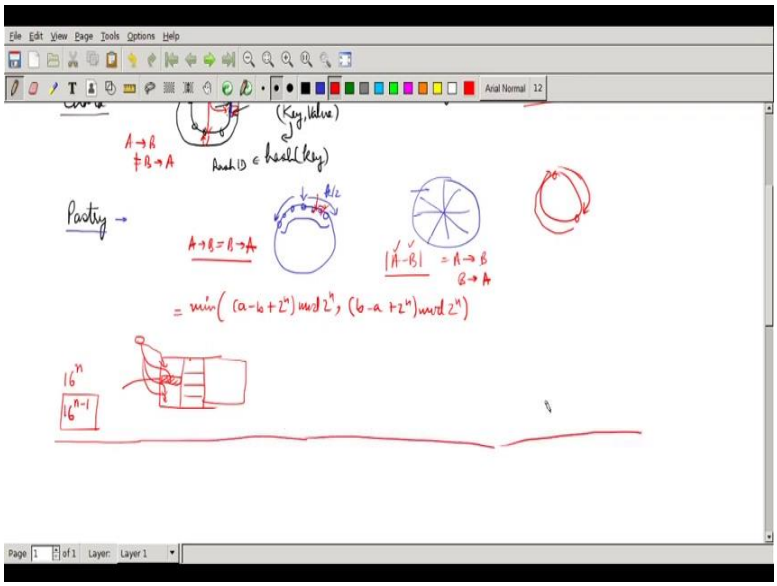
So in, based on whether your hash ID is matching with which digits you keep on moving it, at some point, you cannot find out any more matching or any closer and numerically closer entry with the same number of digits being matched. You start using your leaf set to find out where the node is lying. If the node lies somewhere here, it will then determine which one of the two nodes on both sides, lower and higher, is the closest.

Now an important thing is not using asymmetric distance. While in the chord, there is an asymmetric distance which you were using. Here is what it was always the node ID minus hash value plus whatever was $2^n$, which is the number of nodes there; n is a number of bits in the system, modulus of $2^n$. This was used for finding out. So x, whichever x gives me this value as the least value that will become the root node; that is what was happening in this mechanism.

In this case, this has been done. Whoever is the closest one, if they both are at the same distance, the higher one is always chosen. When, so it is the symmetric distance, so A to B and B to Y, this distance will be the same. So it is, basically is like A - B, so that is what is being considered where A is a larger value of B. So, there is a distance from A to B as well as from B to A. So that is a case which has been taken.

So, the way we write, we can write this is as a minimum of the two values. So $a – b + 2^n$ modulo $2^n$ this will be one distance. So, on a circular thing, if this is 'a', this is' b,' so this will be b to a. This is what the distance which we are coming with this is. Another distance is this, so a minimum of these two is being taken as the distance between two nodes. The second value, the minimum of the two, means it has to be b - a now. So, this is what is going to be the distance in Pastry.

(Refer Slide Time: 07:35)




This was a problem now. One has to be higher and lower, and we have to be always consistent across the whole network. Can we have something that will be symmetric again, but only one node will be the closest? So how can that be done?

In this case, there is, there can be a possibility of two nodes being the closest. There is only one node in an asymmetric node that will be closest, but it is asymmetric. So, A to B distance is not equal to B to A distance actually, so this is not true in this case. While here, A to B is equal to B to A. But in this case, there is more than one node that actually can come, so is there a possibility? And, of course, we also wanted to use what is being used in Pastry.

Basically, for every routing table entry, so for example, based on the first digit, 16 partitions were being created in the whole thing so that you will be in one partition. From each partition, you will maintain one entry and then whatever partition you are in that will be further based on

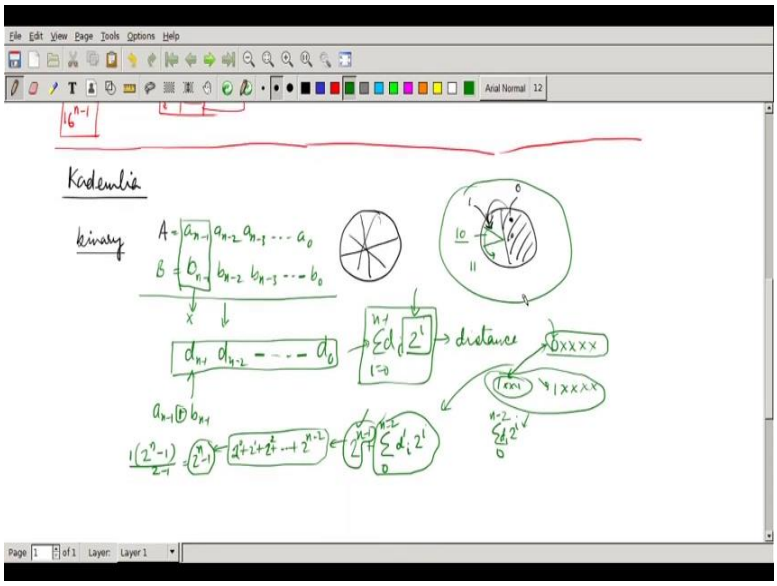
the next digit will be partitioned into further sub-partitions, 16 of them. And then you will be keeping a pointer to each one of those sub-partitions.

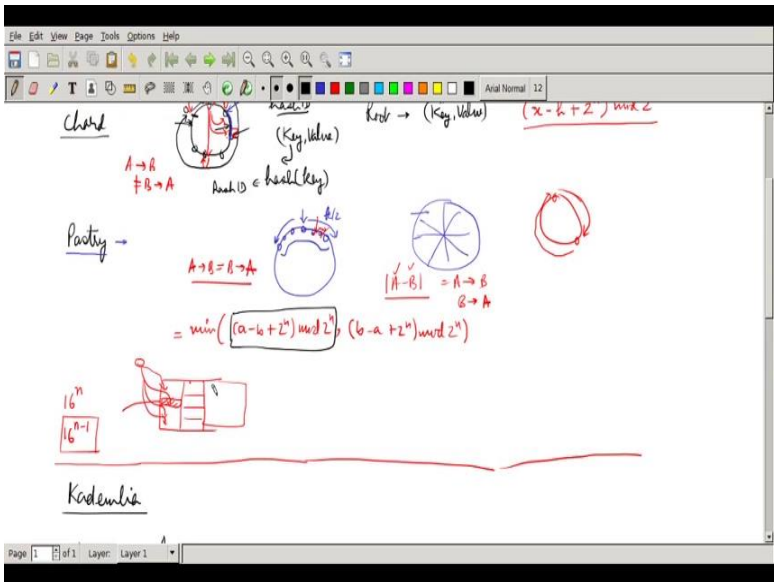
So, you are going to, technically, if you are using $2^n$, for example, it was hexadecimal, so there are n digits, so then $16^{n-1}$, these many possible values can come in here. So out of these, we were always choosing somebody who will have a lesser proximity metric, which will be closer to me based on round-trip time or the number of hops.

This is kind of an optimization that was possible in this case; this was because of the fixed partitioning. It was basically based on node ID; it was not based on the offset from the current node, which was happening in the chord. So, this is a logical evolution that has happened. So, this makes it efficient.

Now, the same thing can actually be used even here also. We still want to use the same thing, but we want to keep the asymmetric distance. This can happen in Kademlia, so we will let us talk about the Kademlia protocol, which is today's topic.

(Refer Slide Time: 09:43)




So, my apologies if in the, during the course I might have used incorrect spellings for this. So, this is the correct one, and this is what has to be used. So, in this case, the nodes are, node IDs are represented as binary numbers. So, I am going to only talk about Kademlia, which is using binary. And this automatically gets extended to Tapestry the moment I start using some other Emery numbers. Emery digits are going to be used for node ID. So currently, it is only with binary or bits.

So, if there is, we can take two nodes. First of all, I am defining how the distance is computed between two nodes, and we will come to a very similar formula as we have discussed here. This particular, something similar actually, we will find out even for this case, but it will be written differently this time because it is a very interesting organization. If the whole node ID space is partitioned, it is not in 16 partitions because I am looking at a binary system. We partition every time into two halves.

So, in the first half, what is going to happen is you are going to, all the nodes will be starting with a say, bit 1, all the nodes will start with bit 0. So, if you find out you are here, your hash ID is going to be starting with 1. None of the nodes here, none of the nodes in this partition, will be closer to your hash ID. Somebody here has to be closer, so you will simply hand over the request to him.

Now, this guy is going to do the partitioning further. So based on, now it is either 10 or 11. If your node ID is matching 10, so the closest node has to be here. Otherwise, the closest node has to be here. So you will hand over the query to him or further retain it within this domain and now search with the third digit. So every time, 50 percent possibilities you are removing. That is what we are doing.

First of all, let me tell what the distance is and then explain this thing in the form of how on by using a tree partitioning method so, if node A is going to be represented as with say, n bit so I can represent it as $a_{n-1}$, $a_{n-2}$, $a_{n-3}$, …., $a_0$. I can have node B, which can be represented as $b_{n-1}$, $b_{n-2}$, $b_{n-3}$, …., $b_0$. So how now I am going to compute the distance between them?

So, what I will do is I will take this bit, I will do the XOR between them, and I will get some dn minus 1. If they both are 00, I will get 0; if they are 11, I will get a 0. If only one of them is 1, the other one is 0, and I will get a 1. So, $d_{n-1}$ is nothing but $a_{n-1}$ XOR $b_{n-1}$, and that is what I am putting here.

The same will be done with the next digit; I get the $d_{n-2}$, and so on; I will get $d_0$. Whatever the numerical value of this, the numerical value of this will be the summation of $d_i\ 2^1$, where i goes from 0 to n - 1. So that will be the numerical value. So, that will be the distance.

Because this is what is happening, I can now represent this distance pretty well using a method. Because I am now representing this, so if the distance between that nodes because my hash ID is starting with 1, my current node is going to be starting with 0, whatever it matters. So my hash ID starts with 1, so we find some nodes, some nodes starting with 1.

So, when I compute the distance here, this will be something, so this will be 0 basically when the highest order is XOR. So, this will start from $2^i$, and this will go from 0 to n - 2 only. But when I am going to look for the distance to this because my hash ID is starting, so when I compare with this one, this should give me a value that will start with something n - 2.

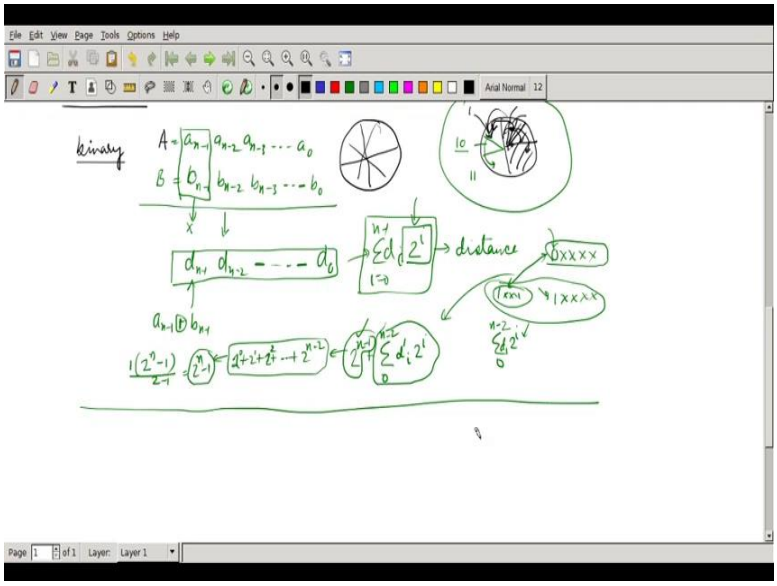
Some values will be there depending on the XOR, so this can be, I can call it d prime i $2^i$. Here I should similarly use di actually, the distance bit by bit for with this one. But there will also be going to be $2^{n-1}$. Now this number, if you actually add 1 + 2 + 3 +……, + $2^{n-2}$, if you add this, this is what you, what we are going to get is as this is an arithmetic progression.

Sorry, it is not arithmetic progression; this is going to be 2 raised to the power, the maximum value which is feasible for this particular thing. So, where all di primes are 1, then what is going to happen? So, $2^0$ will give me 1, so I should call it $2^0$ … $2^{n-2}$, total n terms.

So, the first term, which is going to be 1 into whatever is the ratio. The total number of terms is n – 1/2 - 1, $2^{n-1}$. So, this value is always going to be lower than this, which implies that none of these nodes which are starting with the highest order bit 0 will be closer to the hash ID which is

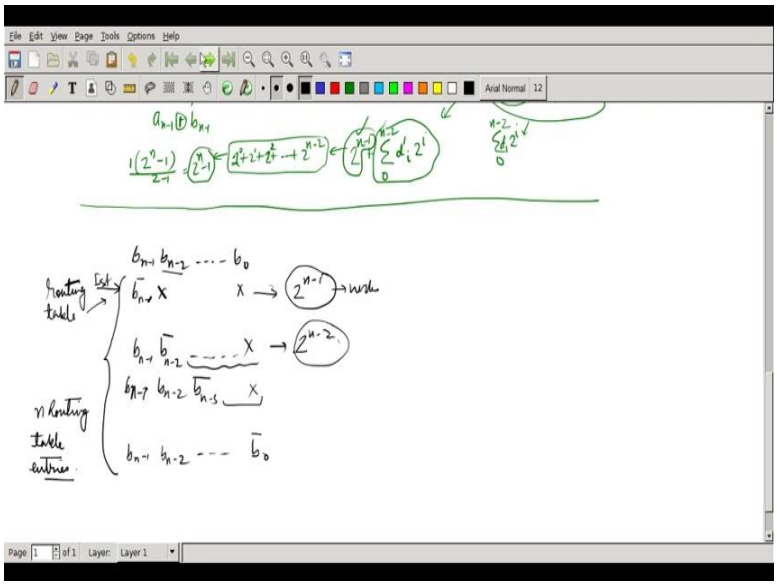
starting with 1, and that was the logic which I was trying to explain in this particular figure. Now, once we understand this basic premise, let us see what will happen and how the routing table will now be created.

(Refer Slide Time: 16:15)




So, routing table typically where it will be? This guy is now going to maintain a routing table to this guy, the other partition, when within this half, one node from here; then within this half, one node here; within this half, one node here; and ultimately, within its partition, there will be only 15 other entries. It can; it will, there will only one entry which can be either deviate at the lowest bit a0, b0. If that exists, it will be maintaining that one entry. So that is what will be the routing table. So, let us now try to make a routing table, how it will look like.

(Refer Slide Time: 16:49)



So, for node $b_n - 1$, $b_n - 2$, b0. So the first routing table entry, which will be there, here $b_{n-1}$ will be inverted, and all other entries will be anything. So, this represents a large number of nodes, which will be $2^{n-1}$ possibilities. This node is, this bit is fixed so other is $2^{n-1}$. I will pick up any one of these nodes and then be replaced here as the first routing table entry. So, this is the first entry.

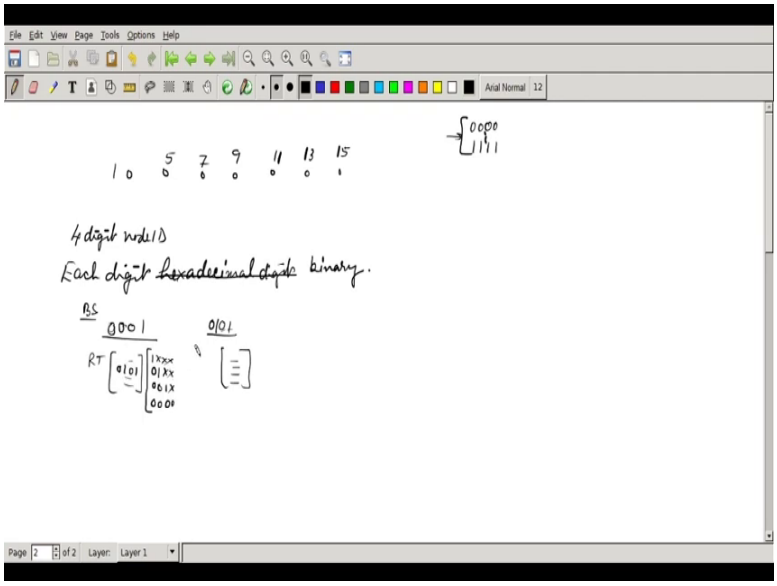
For the next entry, I will be taking $b_{n-1}$ as it is, but whatever the inversion of $b_{n-2}$ will be taken, I am putting bar. And then, of course, you are going to have any other pattern which can come here. So, we have now $2^{n-2}$ possible nodes, which can come in if they exist. If none of them is there, this will be blank.

If only one of them is there, that has to be there actually in that case; if there are two, so one of the two has to be there. And, of course, we can use the same principle that is used in Pastry. Whoever will have a lesser proximity metric, I can always put that for the better efficiency of the system.

The next step will be $b_{n-1}$ goes as it is, $b_{n-2}$ goes as it is, $b_{n-3}$ bar I will be choosing, and then all the digits can be anything, one entry from there I will keep on doing. The last entry will be now $b_{n-1}$, $b_{n-2}$, and so on, $b_0$ bar; if it exists, otherwise, it will be a null entry. So, I will be maintaining

the total n routing table entries. So, this is how the routing table entry will look like. So, to better understand it let us go for an example.

(Refer Slide Time: 19:00)




In the example, I am considering a case where the nodes pick up some nodes and put them. So, node 1, node 5, node 7, node 9, node 11, node 13, and node 15. Only these many nodes are there. So, in this case, I am also now assuming not a very long number of digits in the node IDs, I am looking only at the four digits in the node IDs, and each digit is a hexadecimal digit. So, these four-digit node IDs the four, all four has to be bits. They should not be a decimal. So each digital will be a bit. So, these are four digits, four bits technically. So, I am going to go from 0000 to 1111 and everything which goes between them.

So, all of these 16 and with the node IDs, I am considering for that case. We will be considering a similar thing but with hexadecimal digits in the Tapestry algorithm, not here. This, I think I should remove; four digits will be binary digits because we look at Kademlia. So, let us look at node number 1, how the routing table will be there.

So, maybe I can, in the beginning, I can start inserting the nodes one after another and then see how the routing table evolves. So, let there be only node 1, which will be there present in the beginning. Instead of finding out the complete table, so this will tell you how this happens.

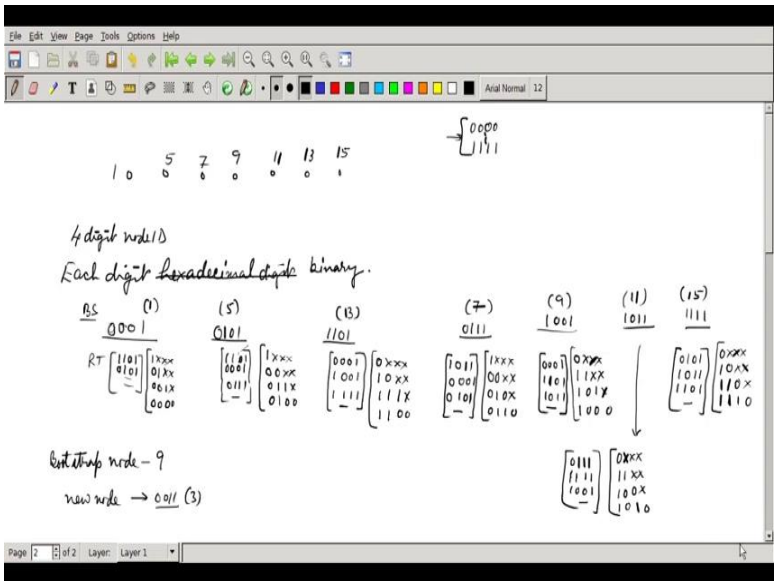
For node number 1, so the bits are going to be like this. This is the only node, so the routing table is all empty; there is nothing in there. There are four bits, so four entries will have to be maintained, so all four are empty in this case. We let node number 5 get added at the time. So, this will now be represented as 0101; this is 5. So, when this 5 will come, it has to attach to node 1, which will become a bootstrapping node.

It will simply then ask for; it will understand that 0001 exists and it will see if it can talk to them, it will essentially tell that I am so and so exist and it will also have an empty routing table. And this guy will now. Also, they have an empty routing table; they will exchange their routing tables. But they will also know who is the, who was the source. So, these are the bootstrap node initially, but later on, we will change the bootstrap node.

So, what will happen as a consequence? This guy will know that I have 0101, so it will start now putting up the entries. So do, does it have anybody which is starting with 1xxx? Nobody is there; then 0, the first bit will be taken for the second entry.

The second bit from the right side, or the second most significant bit I am talking about, is taken like this. Yes, this one matches, so this entry is now going to be replaced. We will have this entry as 0101. Of course, there are no more entries; there is no other node; it will not be matching. So, another possibility which has to come here is 001x and 0000. The four entries should correspond to these particular patterns. So these four, there will be only three entries now.

(Refer Slide Time: 22:50)




So, what will happen at this particular point? So let us see. So, in this case, the pattern has to be 1xxx, nobody is going to be there, so it is free. Then the next entry has to be 00xx. So, this guy matches 0001, so it actually can be put in. So we put that entry in second place. The first entry is full 0001, which will match with this, then we will have an entry which is 011x, there is nobody matching, and then the last one will be 0100, there is no matching for that. This is what is going to be a routing table after the first node has come in.

This is the pattern that has to be matched. The third node will be coming, say 7 or maybe, say 13 is coming. 13 will be 1101, this is 13. So 13 again, for example, is doing a bootstrapping with 0001, so it will talk to 0001 and then tell him it is my routing table is empty, but it will get the routing table from 0001.

It will come to know of the 0001 and 0101, both the nodes will be available. So, it will now look for what kind of entries it can create. It can create 0xxx; it can create 10xx, then it can create 111x and 1100. So, these are patterns that I have permitted for this guy. So, the first entry will be now, which will match if any one of the two can be there which can be matching, so it will keep any one of them so that it can keep 0001, other entries will not be there.

But now, this 0001 has learnt about 1101, so it does not match. The first entry is going to match with this; it will keep 1101. So, this entry gets updated. The next time it is going to the routing table, the exchange will happen, keep on happening with all the nodes, so 0001 and 0101 exchanged the routing table, so the knowledge of 1101 will come to 0101 also. So, it will also update its routing table. It will now change it to 1101. And this will keep on happening.

Ultimately, when all nodes come, let us see the final routing table. Let us try that. So, let us put up all the nodes; I have put already 1, 5, I have put 13, there will be 7, 9, 11, and, of course, you will have 15. So, four more nodes have to be done.

So, with the 7 now, I will just populate all the routing tables. Any other node that will match with 0011 so no other entry matches, so this will remain dash; no entry matches means this will remain as it is. What the fifth? It is going to match no other entry because no other entry is there with that particular thing. So, this one will be 0111, this is going to be 1001 and 1011, and this will be 15. This is what will be the bit patterns for these.

So, for 5 the first one entry will be 13 as it is, does not matter; the second one can only be 1, so there is another possibility, and the third possibility is there, the 7 actually can come here 0111 will be coming. We will just remove it, and then, and there is no other entry with 0100; there is no matching.

Now coming to 13, what is going to happen? Let us see. 13 will also populate itself. So, 13 will match with, 10 I have to figure out somebody, so 9 is there, and 11 is there, so anyone can be kept. So, let it keep 1001. Then 111x, there is only one entry which matches, and there is no entry with 1100, so this remains dash. So, in the same way, I can now design the pattern that has to be there. So, it will be 1xxx; it has to be 00xx, then 010x and 0110, which will be there.

So similarly, I can do it for now 1001, I can find out the pattern first. It will be 0xxx, which has to be kept, 11xx, which has to be kept, then 101x has to be, which has to be kept, then 1000,

which has to be there. Similarly, for 11, the pattern will be I can write it here. It will be 0xxx, 11xx, and 100x and 1010.

For 15, the pattern will be 0 all three x's, then 10xx, then 110x and 1110. So, let us now fill up what will be the STD state routing tables. So, for 7 now, I can fill up the entries correspondingly. Anybody with 1 will be good enough so that I can put. 9, 11, 13, 15; so let me put it 11, 1011.

Then 00, the entries which are going to match is only 1, which is 1010. The entry only, which will match, is 5, so 0101 and 0110 do not exist. For 9, similarly, let us put up the entries. It will be anything matching with 0, so it can be 001, or it can be 5, or it can be 7. Any one of the two can be kept but let me keep it 1.

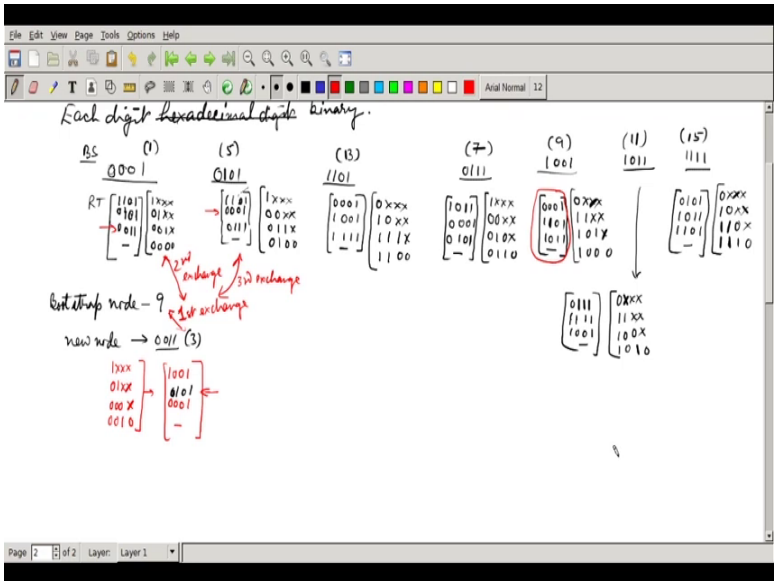
Then 11xx, so I can keep it 1101, 13. 101x, so 101, it is only 11, which is matching 1011. And 1000, there is no one entry for that. Similarly, for 11 now, the entry will start at 0, so I can keep it 7 also in this case. Then we will have 11, what will match is there only 13 and 15, I can keep 15 here. And then 100x, there is only 9, which is matching, and 1010 does not exist.

And similarly, for 15, there will be an entry. This, anything that matches with 0xxx, so this will be 0101, I can keep it actually, I am just choosing any one of the 3. 10xx is, can be 9 or 11, so I can keep 11. And then 110x, so this can be only one entry, which is 13, and there is no entry here which is matching with 1110, so this will be blank. So, this will be the routing table under STD state condition.

Now such a scenario, when a new node comes in, what will happen? Let us look at that. And this time, we are going to change our bootstrap node also. We will now keep the bootstrap node as, so the bootstrap node is now we are keeping as 9, and the new node which is entering is 3. So, we have to see whether all things will become correct if we just keep on exchanging the routing

tables with all the members in my routing table; everybody has to do that. This time, it will be 0011 or node number 3, which has arrived in the system.

(Refer Slide Time: 31:48)



So, with this, what is going to happen? So node 3 is going to talk to node 9. It will come to know of these particular nodes. It will come to know of these particular nodes, and it will come to know of 9 also. So, based on this, it will now fill up its routing table. So, the pattern it has to have is 1xxx which has to be there, 01xx it has to present, has to have 000x which has to be present, 0010 which has to be present. So, based on that, the routing table has to be created.

We create a routing table, and it will have an entry coming as anything that matches with 1 in these, 9 matches so that I can keep 9. 01, anything that matches with the 9 or something, there is no entry matches with 01. No entry matches with 000x; yes, there is one that matches 0001, which will come. 001, there is no entry matching with this; this will remain vacant.

So, this will be the routing table after the first exchange. So, the first exchange which has happened between them. It is the first exchange, and this has resulted in a routing table is created. So, 9 will also only come to know of 3, so it will keep any entry. So, it can, it already has an entry 0001, so it need not replace it.

Now 3 will talk to 1, and 3 will talk to 9 again. When 3 will talk to 9, which is the entry here, nothing will happen; when it will talk to 1, so it will tell 1 that there is a 3 which exists. So, it will now find out that if 3 can be put in. Remember, this third entry is 001x, it is matching with that, so there will be a replacement here that happened. A new entry will get created.

So this will be 0101, after that 0011, the third entry will get created; 000, nothing exists as usual because there is no new learning from here, only 3 has been learnt, and 9 has been learnt. So, 9 is already keeping an entry 13, so 9 cannot be used here.

If we come back, 3 we will learn about some new modes from 1 and based on that it will try to replace. So, in 1101, there is already have an entry 1001. 0101, can it go somewhere? Yes, it can go to second place. So, let us put it near second place. It will go to 01, 01 entry which will come. Now anything has 0010? 0010 does not exist; it will remain vacant. So, this was the second exchange, which happened between 9 and 3.

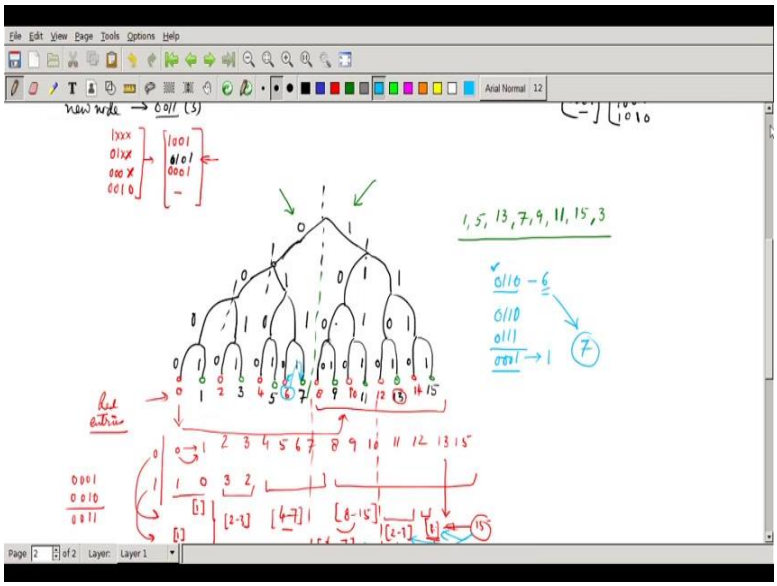
Now anything which, so it has lead to this change in 1 and this change in 3. So, the next time it is going to now make exchanges with each one of them. So, the new node which has come in is now 5, but this time the third exchange will happen with this; of course, it will keep on exchanging with 1 and 9, but that will not lead to any more change. So, there is a third new node that comes in.

It will come to know of new nodes. So, it will also tell 5 that these nodes exist. So, 5 is already aware of 9 and itself, 5 and 1. So this is already considered; in these, no routing table change can happen. Only because of 3, if some change has to happen, it will happen. Only 3 can come to the place where there is this at this place, there is already 1 which is existing, so there is no change in 5.

But let, if there is going to be any change in 3. Now, 3 will become aware of 1101 and 3 will also come of 0001 where somebody is already there and becomes aware of 7. 7 is not going to be used there because it needs 2, so 2 does not exist, so the 3 tables remain as it is; 5 table also remains. 5 also does not get 0100, so it cannot replace.

But now, where is going to be the change? So now, this 1 also has learnt this entry. So next time it is going to do the exchange of this table with everybody else. So, it will; it is going to tell about, tell 5 that which is the entry; it will tell 13 also about 3. So, this entry is now what is going to happen. Now, almost all entries are stabilized. After this, no more change is going to happen even if you keep on being exchange. But what exactly is happening in terms of partitions? We need to understand that. So, before I go to basically how the routing happens, let us look at through partitioning what is happening.

(Refer Slide Time: 37:07)




Let me do a partition, binary partition 0 and 1. So I do 0 on this side and 1 on this side. So this is the first bit. The next bit will be again from here it will be 0 and 1 and this will again, further

partition 0 and 1. This will also partition here, 0 and 1. Now, I am going to do further one more partition to get 4 bits.

So, what you can see is I have created 16 partitions. So, the first time I did a bifurcation based on 0 and 1, then each one of them is further being bifurcated, then those small partitions can, are being further bifurcated. So on every time, I am doing bifurcation. Now, each of these leaf nodes will be there in the tree is a node, a node ID, but we do not have nodes at all the places. So, the important thing is that we have the primary partition.

So, all nodes starting with the highest, where the most significant bit is 0, will be on this side, and where the most significant bit is 1 they will be on this side. This 0000, the node is going to be here if the node exists with 000. But we do not have a node with 00. So, let me see what is there. So, I can write them down what nodes exist 1, 5, 13, 7, 9, 11, 15, and node 3, which has joined later on.

So, for 1, it has to come to 00 something, so this is a node that exists. I am only just marking the nodes which exist; other nodes are not going to be there. So the next node is going to be 5. It is going to be 0101. So, 0101, so this has to be this one. 010101, I am just put marking leftover ones. So, we have 5 here. So next, we have 13. So, 13 is going to be 1101. So, it is 1101, so this is what, where the 13 is actually. This is 13; this is 5, this is 1.

So, I think, let me put it in the same colour. So, this will be 1, 5, 13, and then, of course, we have to take 7, so 7 will be 0111. So, 0 on this side and then all 1, so this will be the next entry. This will be 7, then, of course, 9. It is going to be this one, and it will be 9. Let us write it down; it is 7, it is 9, so next, let us put 11. So, 11 means 1011, so it is 1011, so this will be 11. And then we will have 15, which is all ones. Then we will have 3, which has been the new entry added, which is 0011. So, these will be the nodes.

So, let us now put the nodes here. So, at other places, nodes are not there; they are just free node IDs. So, these node IDs are not used; they are free. So, any hash ID belonging to these red nodes will essentially map to the closest entry. So, we can try seeing that I am going to put these missing entries now. So, the ones which are shown with the red are missing entries.

So, we will have hash IDs for this; they have to just go to somebody closer. So, for example, now we can look at who is going to be closer. So, the routing table is maintained, for example, for 0, so it is in the left partition. So, one entry will be there to any one of the nodes in this partition. In our case, if you look for this, so you have 1101, which is 13. So, this entry is going to be, is entered for 0.

So, if any hash ID which is belonging to this, we will just pass it to 13 and 13, well then, we will then figure out which another node it has to be given to either 9, 11 or 15 depending on the second bit, which is going to be compared. If the second bit is 1 from a second most significant bit, then it can be handed over to any one of these two entries: 15 or 12 or 11, then it has to be the closest guy. So, with 11, the closest will be 13, and so on. So, only one node will be closest as always. If the most significant bit is 0, it has to be coming on to the left side. Now second most significant bit will decide whether it has to go left or right, which is the way it is going to be happening.

Now, what is the distance? Now 0 is how much distance away from 1? So, if I compute for 0, so this is 0 distance. This one is at 1 distance away, this one is 2 distance away, this is 3, this is 4, 5, 6, 7, and then 8, 9, 10, 11, 12, 13, and 15. If you are talking about bit 1, how the distances will be there? So have a look at this. This will be 1 hop away 1 distance away. This is 0 distance away. This one is going to be; we can compute. So, this one will be 0001, and if I look at 2, it is 0010. So, this will be 3 distance; this will be 11. So, this is going to be 3 distance away.

This is 1, and this is 0 distance away; this will be 3 distance away. This one will be 2 distance away and then, of course, similarly we have to, will come for these and so, this guy will be actually for 0, this will be one hop away for 0, you can see. Or if I am looking at 1 this will be one hop away, and this particular range is always going to be 2 to 3 range away for both. So, this is for 0; this is for 1.

And this particular range will be 4 to 7 hop away. 4 to 7, it will be distance, and the last block will always be from 8 to 15 distance away. So, I will keep one member from here, one from here, one from here, and one whichever is a leftover. Even early, if you start picking up from some other node, if you pick up, say 12, or say 15, who will be there?

So, 1 hop away only will be the 12th guy, which will be should be 1 hop away. Then who will be 3 to 4 hops away? So, 11 and 12. So, it will come from here. This range, 10, 11, so I have made a mistake here, I need to correct it. This should have been 12, and this should have been 14, so let me rectify that situation. This should have been 12, and this should have been 14.

So, this one would have been 2 to 3 hops away compared to this guy 15. So, 12, 13 so I need to change. So, this is for 14. So, I am talking about node 15. This one is for node 15. And then we will have this whole range, the mid-range which is going to come here. These guys will be now from 4 to 7 distance away, and this whole block will be from 8 to 15 distance away.

This is how, essentially, you pick up any node, you can then define this logarithmically increasing or every time doubling block you can figure out. So, for 15, you are having 14 in this block, then 11 and 12 in this block, then from this is 14, this is going to be 12 and 13 will be in this, block 8 to 12, 11 will be in this block, and this will be for.

This will be 12 to 13 range will be in this, 14 will be in this, and then this one will range from 8 to 11. So, this is going to a 4 to 7 distance with respect to 15. So this is always for this. So, I can draw it in a different colour. So, this is the distance I am talking about. This is the range I am talking about. I should not make this square bracket, that is for distances. This 2, 13; this is 14 and this one, this last will be from 0 to 7. So this will be the last distance, which is from 15. So, from 15 actually, we will come to this particular range. So for every node, we can do this.

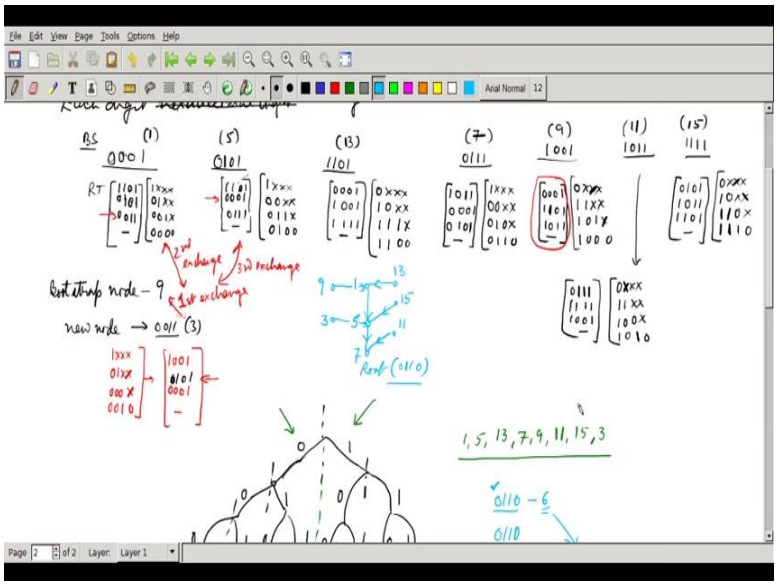
This binary bifurcation always ensures that every node is only a unique distance away from every other node. And there is only one node that is going to be closest. For 0, this one is closest; for 14, 15 are the closest, and if you look at 14, 15 is closest. If you look at 13, 12 is the closest. Now, when you want to search, for example, if I am going to search here, what will happen?

So, if you want to, for example, let us take a number and try to find out who will be the hash ID for that. So, hash ID, let me take 0110, which is 6; there is no node for 6, so where will it be going? So, it will be essential. If 0 is there, I will go on this side; the first digit matches. 1, I will come on this side. If I do not have anybody on this partition, I only have to move to the next one. If there is even one green node here, I have to just remain in that.

So, 01, I come here; there is another 1, there is a green node I come here. Then there is a 0. I will come here, but there is no node here in this case, so I need to go to the closest one, which is going to be this. So, 7 will be having the closest number. So, 0110 and 0111, so distance is going to be 1. So,7 will become the root node for the hash ID of 6.

And if you look at the routing table, we can prove to you; it can start from any routing table and based on the routing table, it will always be going to the correct node. So let us try it out.

(Refer Slide Time: 50:00)



So, I am doing it from 6. So, 6 is, I can give it to say 1, so where is it going to go? If you start with node 1, so node 1 will compare 6, 0110, it will find out the longest match it will be going with. 01xx, for that entry, it will be essentially going to 5. So, from 1 it will be going to 5. A node number 5 will now do the again matching 011 that will match with 7, so it will match 7 and 7 is the root node. So, ultimately it will reach with the longer, best match possible.

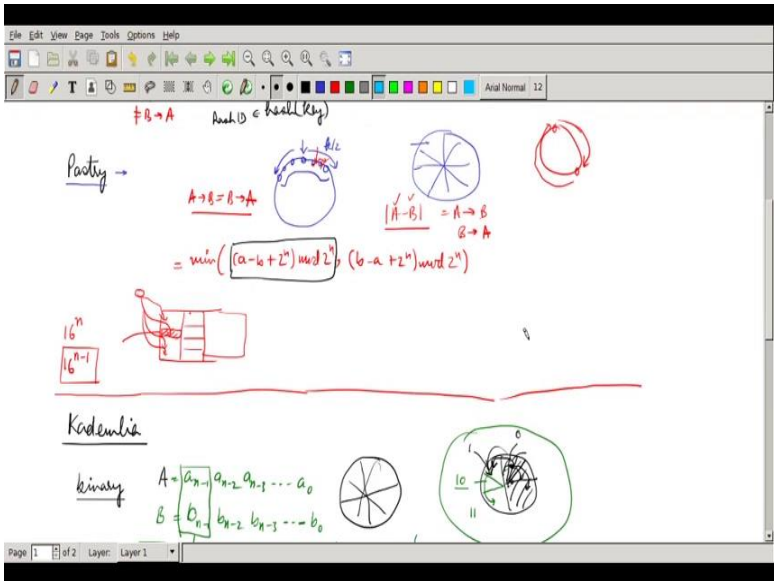
If, for example, now 6 starts with 13, where will it go? Let us look at 13. 13, 0110, so it will match the first entry; it will go to 1. So, from 13, it is going to go to 1. If it starts at 7, 7, the root node cannot give it to anybody; it will determine it is the root node. If it starts at 9, so where is it going to go? If 9, if you look into the routing table, the only entry is 1, so it will match with 1. So, 9 is going to hand it over to 1.

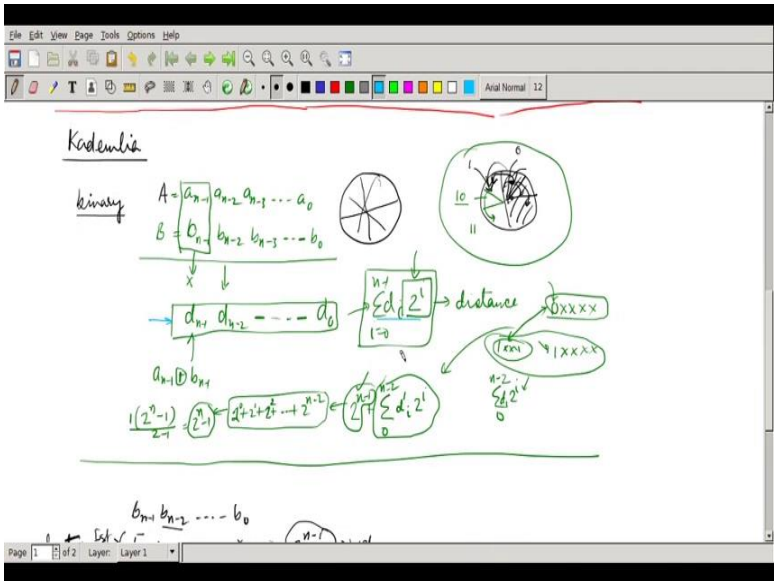
Then we have 11. 11, if you look at the routing table here, the entry is going to 0110. So, the first entry will match what will pass on to 7 directly, actually in this case. If we look at 13, so 13, it will match with 0101, basically 5. So, it is going to be from here; this is from 15.

So, I have got all 4 plus 3, 7 nodes. There is also 3, node 3, which is now available. If we look at this routing table, it will have the best possible match with the second entry 0101, 5. So, it will

also give this one, node number 3. This will be the tree with which now the routing of hash ID 6 will happen. It starts from any node it will always reach 7, which is the root node for 0110. So that is how the routing is going to happen.

(Refer Slide Time: 52:36)

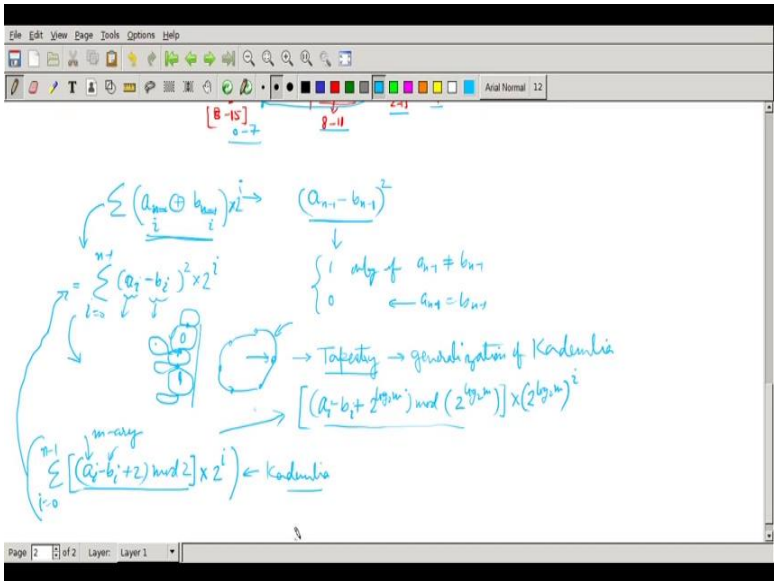



Now, let us come into the distance metric; we have done an expression initially, so we have to see that expression now. Come back. So, we have given this particular expression, and I said further, we are using XOR. This is the XOR which we are using, and then using this, we are

trying to figure out what will be distance. So, let me take this thing further, and what will be the appropriate distance metric we can use?

(Refer Slide Time: 52:02)



We were currently using XOR, but if I look at if a minus 1 XOR $b_n$ - 1, this can also be written as $(a_n$ -1 - $b_n$ – 1)$^2$. This also will give you 1 only if a minus 1 is not equal to $b_{n-1}$. So, one is 0; the other one is 1. If they both have 00, they both are equal. So, this value will become 0. So that is what also XOR was giving.

So, I can write down the same expression $(a_{n-1}$-$b_{n-1})^2$, and of course, this n now, it needs to be replaced by i because I have written a summation. So, this is what will be the physical distance. So, I will not call it minus 1; $a_i$ - $b_i$ whole square $2^i$, where i goes from 0 to n - 1. And this will be the same distance you will get from XORing and then finding out, basically doing summation and converting it to i, into $2^i$. So, this and this, both are the same.

I can further now try to generalize it. Currently, 0 and 1 are being arranged in a circle. So when, if you are matched to 0, you come to 1, if your hash ID is matching with 1 you come to 1 in the digit, and then this will have a circle. This will have a second digit circle.

So, this second digits will have, further there is going to be one circle here, this is for the third digit one circle will be here. So a similar thing is going to happen here, and one circle is going here. So, every time, there is going to be 1 for which is going to happen. So, we can now generalize. In this case, your ID is either matching with this or this; depending on that, you are either handing it over, or you are keeping it with yourself. Every time this is what is happening. And if there is even one single node in the other partition, you will get an entry, and you will pass it on.

Instead of this circle of only two bits, I can now make a larger one, something like this. So, I can now actually use here not one but, not a binary bits, but I can use Emery digit. In that case, there is a problem; I cannot do a square in this way. So, technically, I am now trying to find out whatever is my bit, who is the root node for that particular digit. And once I hand it over, then within that circle, I am searching every time.

So, this case can be generalized to Tapestry. Tapestry is a generalization of a Kademlia. Tapestry is now going to be a generalization of Kademlia in this case. So, instead of using a binary digit, I can use Emery digits. So, I can now put a - b, and if it is Emery digit, so there are m thing, $2^{\log_2}$; n will be there, I can now put this n modulus of $2^{\log_2 m}$. This will be the distance on that digit, and then I can multiply by whatever is a $2^{\log_2 m}$; raised power with digit ID. So I had to put $a_i$ and $b_i$.

So, this gives me the symmetric distance in a circular fashion. So if my hash ID for that digit contains a digit contained by the hash ID at the same level is in between this range. I will choose the node here, and further only with these nodes, I will be going to another ring and then finding out where the next digit of the hash ID will be latching onto.

We will be looking into the Tapestry in the next lecture. So Kademlia is a pretty efficient system, but we can now generalize the distance in Kademlia into this form, which will also be the

distance for a Tapestry. So now, if I do this, m become 2 binary system. Emery, I am talking about Emery, so it is a binary system.

So, m is 2. So, in that case, it will become $a_i$ - $b_i$ + 2 modulus of 2. Now, this incidentally gives nothing but a, the same thing. This $(a_{n-1} - b_{n-1})^2$, or this gives us XOR. So, both things are going to be the same as this. I can use this one and multiply it by $2^i$ summation i from 0 to n-1. So, this is the same distance as what we were computing here or here.

So, this is your distance, which is being used by Kademlia for finding out the root node. So, the hash ID, we will find, when we will find out this is a node, this is the hash ID bit, the whichever gives me this total expression gets minimized that particular node will become my root node.

The algorithm is simple; routing is simple in this case. There is also uniqueness in the root node, I do not have multiple options, and then I need not choose based on a certain property, which gives an advantage. So, in the next video, we will be looking at the Tapestry.