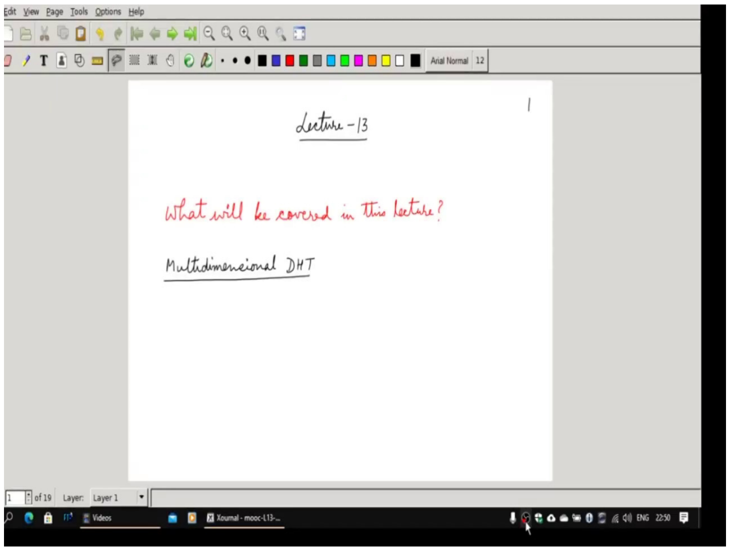
Peer to Peer Networks
Professor Y.N. Singh
Department of Electrical Engineering
Indian Institute of Technology, Kanpur
Lecture 13
Multi-dimensional Distributed Hash Table: Mapping of Peers into Multidimensional Space

(Refer Slide Time: 00:14)



Lecture - 13

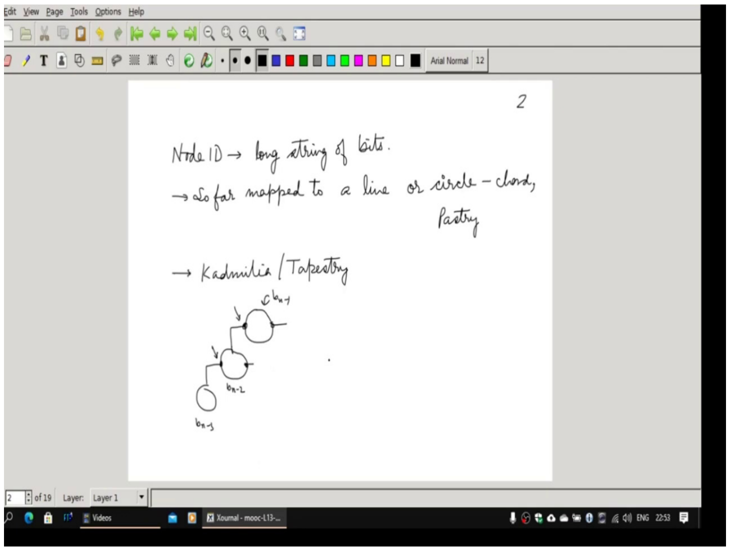
1

What will be covered in this lecture?

Multidimensional DHT

Welcome to the lecture 13, this is about Multi-dimensional Distributed Hash Tables, which we are going to talk about in this lecture.

(Refer Slide Time: 00:23)



2

Node ID → long string of bits.

→ So far mapped to a line or circle – chord, Pastry

→ Kadmilia / Tapestry

So, as we understand node IDs are a long string of bits which are used in any DHT system, so roughly it is going to be 256 bits or even higher. And in chord and pastry they are been mapped to a circle. And then on that circle depending on where the hash ID lies, we define a distance metric and then correspondingly who will be the root node for every hash ID.

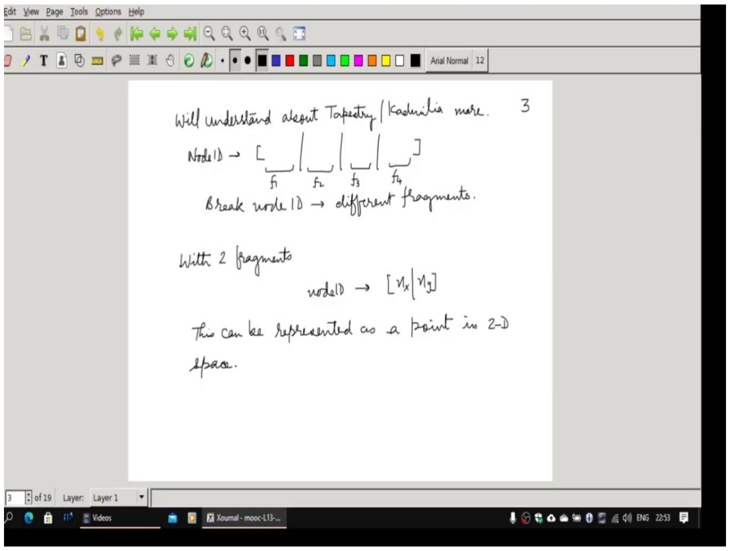
So in chord, for example, it will be a uni-directional moment which is permitted, so in the direction in which the node IDs are increasing in that whichever node is closest, the hash ID that will become the root node. In case of pastry on that circle whichever is closest in terms of numeric distance, basically difference between the two node ID and hash ID, that will become the root node.

Kademlia and Tapestry works in slightly different fashion, so they, for example in Kademlia we are actually take the first bit of the hash ID we compare with a node, so a node actually is going to, we are going to choose either 0 or 1. So, if you are at this particular, if you are at any node, you will find out if the first bit is matching with this, it is perfect, so you go on this side.

And then there is going to be another circle, where the current node which I am talking about is here, so current node was here, current node is here, if it matches it goes into another smaller circle and so on, otherwise it goes on to another side. So I am showing circle because the same thing gets extended to tapestry where instead of ((Refer Time :01:58) points, where we are just bifurcating, we start actually using the circular arrangement of the digit values which are not, for a bit these are value to do for a digital can be, it will be more.

For a tapestry, for example if the hexadecimal digit there will be 16 such entries, one will be occupied by the current node and there will be one nodes corresponding to each other entries if they exist. So it will be depending on which bit I am talking about. So here I will be, we will be handling b_{n-1}, here we will be handling b_{n-2}, here we will be handling b_{n-3}, and so on.

(Refer Slide Time: 02:37)



3

Will understand about Tapestry | Kadmilia more.

NodeID $\rightarrow [\underbrace{\quad}_{f_1} | \underbrace{\quad}_{f_2} | \underbrace{\quad}_{f_3} | \underbrace{\quad}_{f_4}]$

Break node ID $\rightarrow$ different fragments.

With 2 fragments

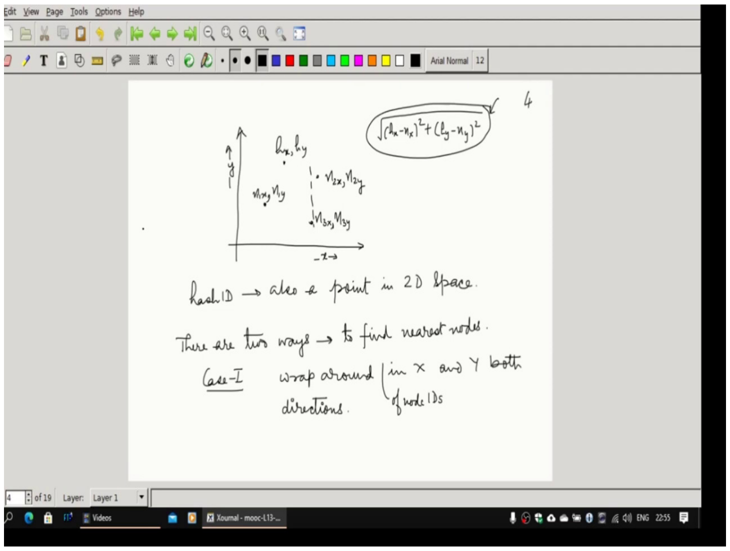
nodeID $\rightarrow [n_x | n_y]$

This can be represented as a point in 2-D space.

So, on a similar lines when we go through this multi-dimensional DHT we will probably appreciate better Tapestry and Kademila. So, node IDs are supposed to be fragmented or partitioned into multiple partitions. In Kademila they are being partitioned every bit is created as a separate partition.

Yet they can be f_1, f_2, f_3 and f_4 but for simplicity sake to understand everything let us take with only node IDs with 2 fragments. So, I call it n_x and n_y, now once you have n_x and n_y they are like x-axis and y-axis argument, so they basically now node ID can be represented as a point in 2D space.

(Refer Slide Time: 03:21)



hash ID → also a point in 2D Space.

There are two ways → to find nearest nodes.

Case-I wrap around in X and Y both directions of node IDs

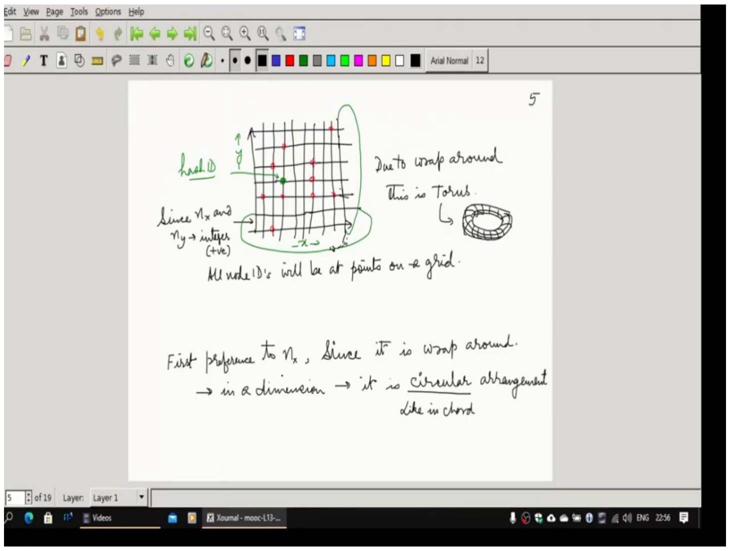
So, a figure will look something like this, we had x dimension is in this direction and y dimension is in this direction. So here hx and hy are the hash and these 3 other node IDs, so h_x will whichever node is going to be closer. Now closer essentially can be Euclidian distance, so where I can actually compute the distance as square root of $(h_x - n_x)^2 + (h_y - n_y)^2$.

So, I want to, I can minimize this distance and find out node such that this happens and that will become the root node. Second possibilities I can actually take precedence of one axis over other, so first of all, find out somebody who is going to be closest to h_x. For example, in this case if I am looking at in x direction so closest can be this, if I am not using a cyclic distance the way it happens in board.

If it is there is a fold-over of the dimension and if I am looking in only one direction, this guy prove closest because this is closer than this node. So, you will first of all hand over to this and this guy will now look in this dimension who is the closest will be the h_y. And now one more thing which we need to understand that these nodes, points are or the nodes are not place arbitrary they have to be on a mesh.

Because they do not only values of the axis which are allowed are integer values. We do not allow non-integer values to be there with are in the node IDs.

(Refer Slide Time: 04:55)



Since n_x and $n_y \rightarrow$ integer (+ve)

Due to wrap around This is Torus.

All node ID's will be at points on a grid.

First preference to n_x, Since it is wrap around.

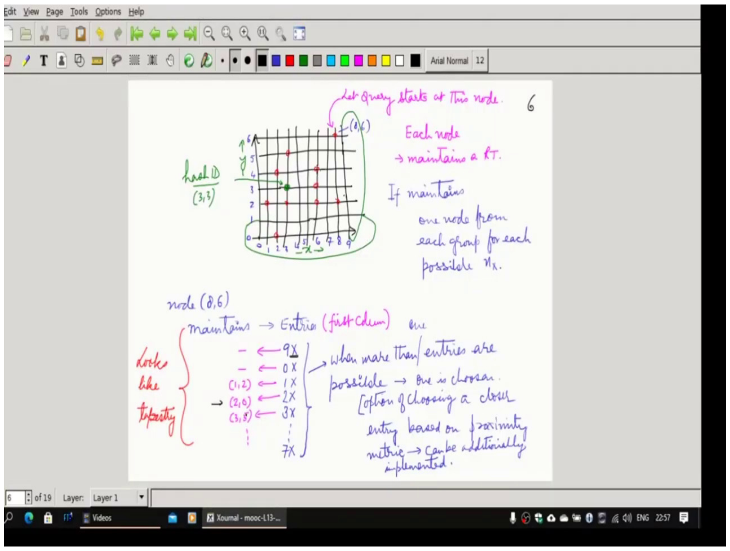
$\rightarrow$ in a dimension $\rightarrow$ it is circular arrangement

Like in chord

Now let us look at a case with the wrap around, here the wrap around actually has been done. And I have placed these red coloured dots are the actually nodes which are placed, there is hash ID also which I have shown. Now because of this wrap around this will look something like a Torus, this is a 2-dimensional Torus the way it has been flattened up and shown that way.

And if it is, not only nx, ny, nz is there it becomes a 3-dimensional torus, if you have n partitions, it will become n-dimensional torus. Now all again, all node IDs will be at the, will be the points on this grid. So, as I am saying that first instead of going the Euclidian distance, we can look for non-Euclidean stuff. So let the precedence be given to node x-axis and it is a wrap around so it will go only in 1-dimension in a circular argument like a Chord.

(Refer Slide Time: 05:47)

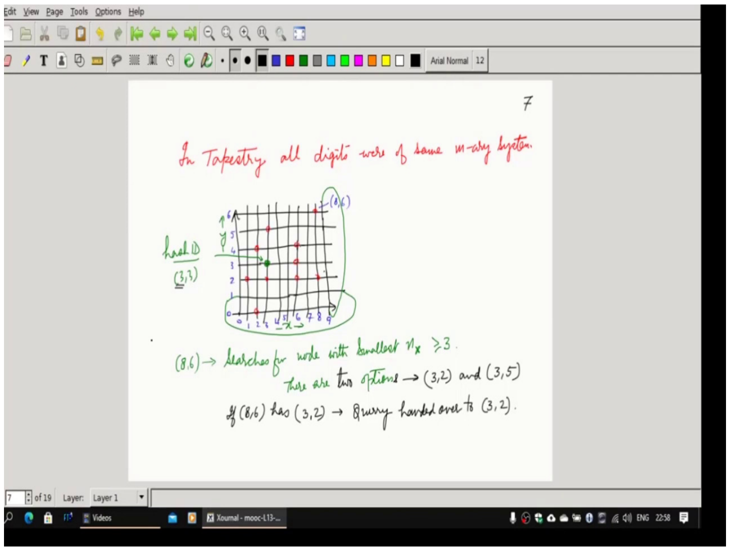

Let query starts at This node.
6
(8,6)
hash ID (3,3)
y
x
Each node → maintains a RT.
If maintains one node from each group for each possible n_x.
node (8,6) maintains → Entries (first column)
Looks like Tapestry
– ← 9X
– ← 0X
(1,2) ← 1X
→ (2,0) ← 2X
(3,5) ← 3X
7X
→ when more than one entries are possible → One is chosen.
[option of choosing a closer entry based on proximity metric → can be additionally implemented.

So, let us all find out who is going to be the hash ID for 3, 3. So we assume that the query is starting at 8, 6, so 8, 6 is supposed to maintain a routing table, so how it is going to maintain? So maybe first thing which you can do is that if we maintain, because n_x is 8 for this node, so with $n_x = 9$ any one of the node entry has to be present. So, I call it 9x, so x means it can be, there can be anything which can be put here, there can be anything which can be placed here.

And then, but there is no node in that column, so we actually going to leave it, 0 at similarly there is nothing 1, 2, 3 we will put the entries. Some places you can have two possibilities, for example this one, I could have put 2, 0 or I could have 2, 4 so 2, 0 has been put. In fact, there is a possibility here that we can use proximity metric which we have talked about in pastry routing.

So, basically who is closer to you in terms of either number of ((Refer time : 6:56) or in terms of round trip time delay. And then based on that any the one which is closer to you can be chosen and put. So that will optimize the performance of your DHT system.

(Refer Slide Time: 07:10)



7

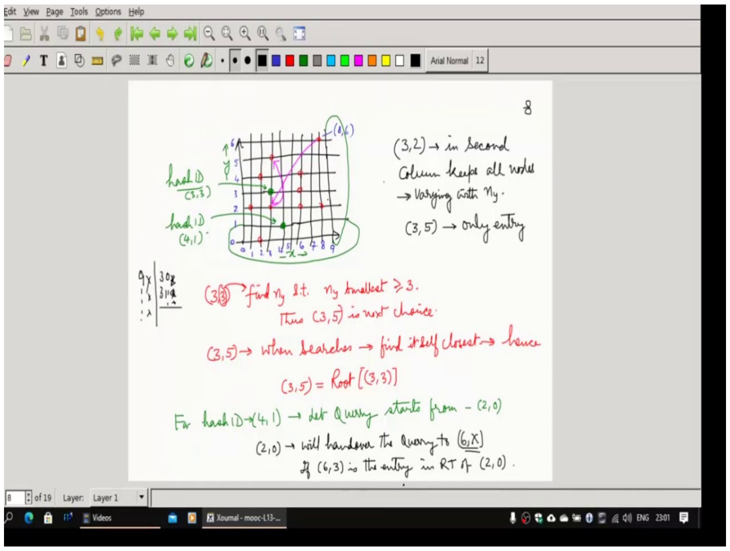
In Tapestry all digits were of same m-ary system.

(8,6) → Searches for node with smallest $n_x \geq 3$.
There are two options → (3,2) and (3,5)
If (8,6) has (3,2) → Query handed over to (3,2).

Now another thing which you should note here is that from x-axis I have only 0 to 9, while on y axis it is 0 to 6. While in Tapestry or digits were of the same memory system. So either using a hexadecimal or octal. But in this kind of an arbitrary partition I can use any modulo system for any digit. So, I am using modulo 10 and modulo 7 here. So, the way the rule is now from 8, 6 it will search for a node with the smallest nx such that it is greater than or equal to 3 because I am searching for 3.

And it has to be done in a cyclic fashion, the way it is done in the chord. So there are two options, 8, 6 could have chosen now or anything which is greater than 3 essentially we have to figure out either, now equal to condition is going to match so either 3, 2 or 3, 5. But both of them will not be there with 8, 6. So if 8, 6 has 3, 2 so query should be handed over to 3, 2 in that case.

(Refer Slide Time: 08:14)



8

(3,2) → in second column keeps all nodes → varying with n_y.

(3, 5) → only entry

(3③) find n_y s.t. n_y smallest ≥ 3.

Thus (3,5) is next choice.

(3,5) → when searches → find itself closest → hence

(3,5) = Root [(3,3)]

For hash ID → (4,1) → let Query starts from – (2,0)

(2,0) → will handover the Query to (6,X)

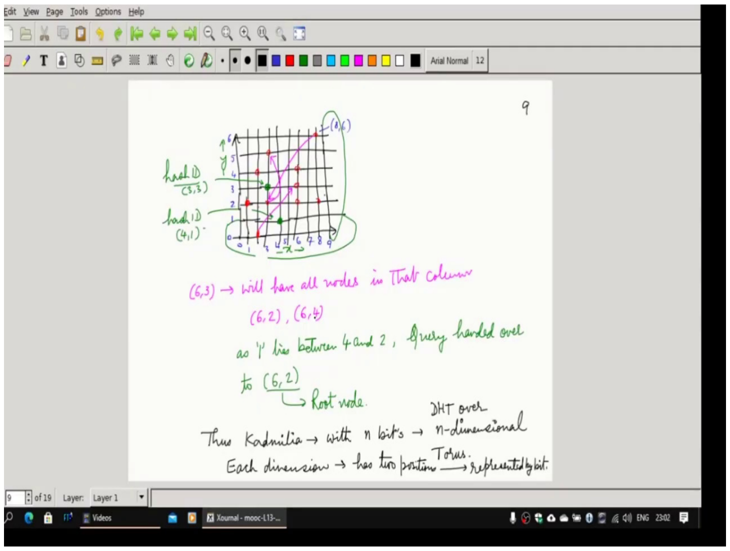
If (6,3) is the entry in RT of (2,0).

So 3, 2 is in second column keeps all the nodes varying with n_y. So, there will be two columns in the routing table, it is something like tapestry. The first column, I am actually first bits all possibilities are taken with x, second column we keep whatever was our value 3, 3. And then of course, we will try all 0, 1, 2, 3 and x in that fashion. So since there is no x there is no third dimension this x will not be present here. This will not be there, this will not be there you all know.

So whichever are present, otherwise there will be dash enter, no entry in the routing table. So 3, 2 in second column will keep 3, 5 as well as 3, 2. And then if you try to find out who is again, cyclic fashion which is the ny which is smallest but greater than equal to 3 which in turn out to be 3, 5. 3, 5 also can do the similar search and we find that it itself is the closest node, so therefore it will declare itself to the root node and search for the key-value pair which has come in a message.

So similarly, we can do the search for, hash ID of 4, 1. So hash ID of 4, 1 when you do the query and we assume let the query start from 2, 0. The query starts from 2, 0. So once the query starts from 2, 0 you just need to be handed over to 6 because that is the only thing which is available greater than 4, so it can be 6, 2; 6, 3; 6, 4. So, if 6, 3 is the entry in the routing table of 2, 0 it will be handed over to that.

(Refer Slide Time: 09:50)



(6,3) → will have all nodes in That column

(6,2), (6,4)

as '1' lies between 4 and 2, Query handed over to (6,2) → Root node.

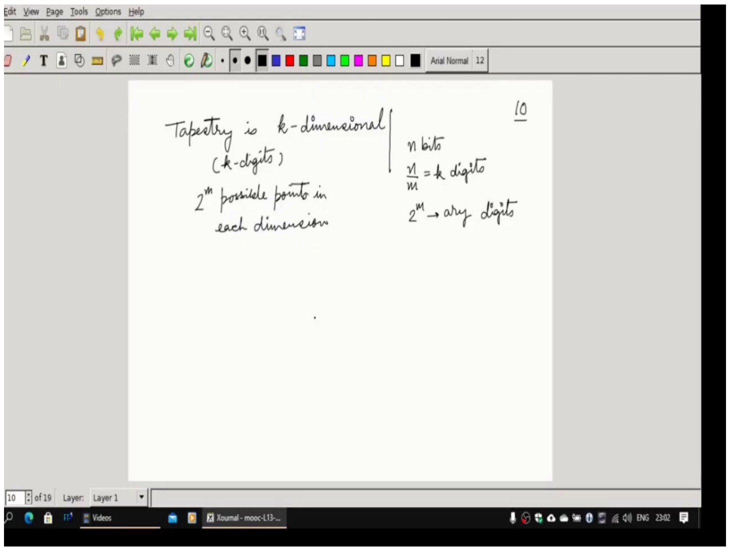
Thus Kadmilia → with n bit's → DHT over n-dimensional Torus.

Each dimension → has two position → represented by bit.

And 6, 3 will have all nodes in that column 6, 2 6, 4 in the second column actually of 6, 3 is a routing table. And as 1 lies between 4 and 2 query should be handed over to 6, 2. So 6, 2 will then search it will find that itself is a root node so the query will stop there and then query can be searched in the query database.

So this is how typically, this is very similar to what happens in Kadmilia. This happens in Tapestry; the same thing happens in Kadmilia ofcourse each dimension will have only two positions in that case. So, Kadmilia becomes an n-dimensional torus with n bit setting inside the node ID. And but each dimension has only two nodes in that case. That is a pretty this like it is not a ring it is creating by circles with only two points multiple of them and in n dimensions torus being formed out of it.

(Refer Slide Time: 10:46)


10

Tapestry is k-dimensional (k-digits)

2^m possible points in each dimension

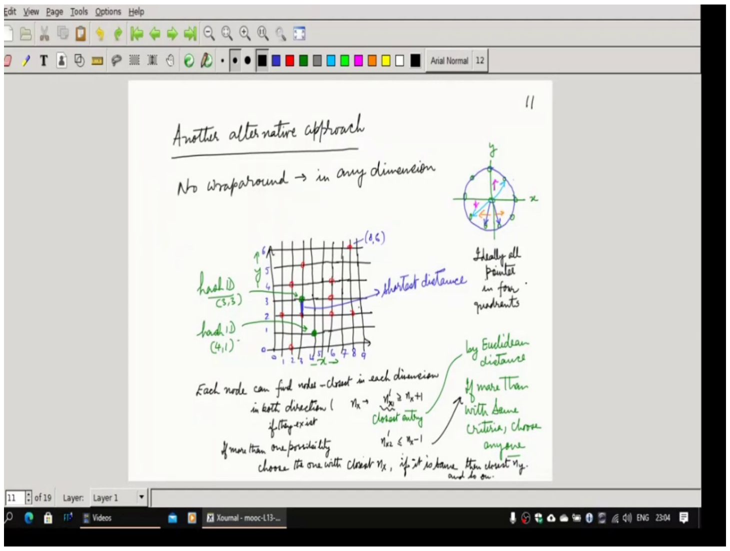
n bits

$\frac{n}{m} = k$ digits

$2^m \rightarrow$ ary digits

Tapestry turns out to be k-dimensional torus because they will be k digits, so there n bits, so n by m will become k digits. So 2 raise power m-ary digits will be used, so 2 raise power m possible points will be there in each dimension that case.

(Refer Slide Time: 11:01)



11

Another alternative approach

No wraparound → in any dimension

Ideally all pointer in four quadrants.

Each node can find nodes – closest in each dimension in both direction (if they exist

$n_x \rightarrow n'_{x1} \geq n_x + 1$ closest entry

$n'_{x2} \leq n_x - 1$

by Euclidean distance

If more than with same criteria, choose anyone

If more than one possibility choose the one with closest n_x, if it is same then closest n_y. and so on.

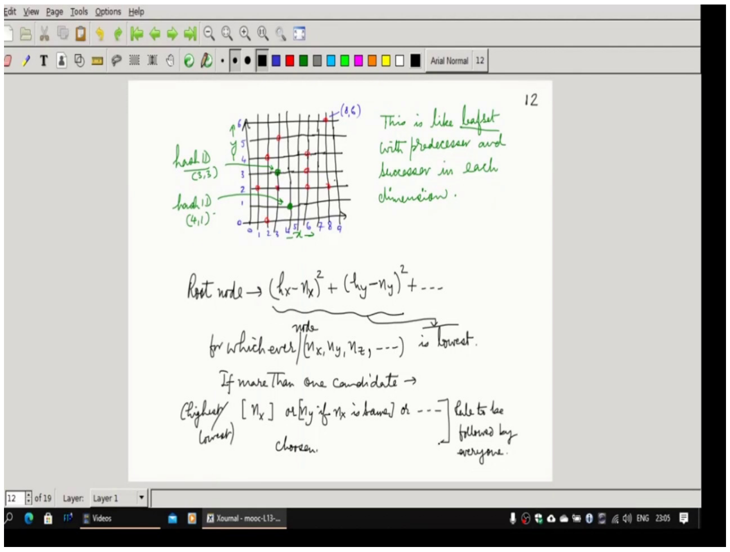
We can now look at an another alternative approach. I am not considering wrap around there, but it can be done. The algorithm is going to work without any problem. So, let us look at, what we do is each node is now supposed to maintain the closet node in each dimension. So, in x dimension it will now find out the closest node, which is going to be closest in Euclidean distance.

I will not be considering anything with the same x value, but it has to be greater than x or less than x, but closest Euclidian distance so everybody will do it and it has have done in all dimensions. So we will find out n_x which is going to be, n_x 1 for example for a n_x, if the n_x is the is the value of current node. So n_x prime, n_x plus 1 prime has to found such that it is greater than or equal to n_x plus 1.

And the overall entry is such that the Euclidean distance with that node is closet. Another more than one possibility choose the one with the closest value of n_x, if n_x is also same then choose the closest value of n_y and so on. So that precedence has to be also specified. The same thing actually here also been shown that all these nodes are at the same distance from the central node, so one up and one down I have to choose.

So no entry should repeat in the routing table, the left side and right side, so value x greater than the current value x less than value and then y and so this four can be the set and this is very similar to the way we use leaf sets in Pastry. But now or in chord we use predecessor and successor. So but they are only two here also we are using two but in each dimension there are two now. In chord there is only one dimension, so there were only two predecessor and successor here there are n dimension there will be two n nodes which will be there in this table.

(Refer Slide Time: 12:54)



This is like leafset with predecessor and successor in each dimension.

Root node $\rightarrow \underbrace{(h_x - n_x)^2 + (h_y - n_y)^2 + \ldots}$

for whichever node $(n_x, n_y, n_z, \ldots)$ is lowest.

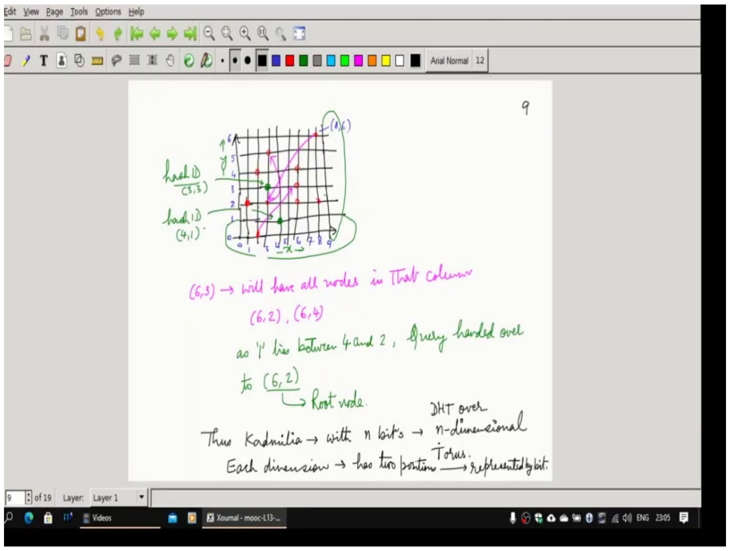
If more than one candidate $\rightarrow$

(highest/lowest) $[n_x]$ or $[n_y$ if n_x is same$]$ or $\ldots$ chosen. Rule to be followed by everyone.

So, this basically n-dimensional leaf set table and now who will be the root node let us find out. So root node will be always that the node with whom the Euclidian distance is ((Refer time)(13:07) So this, this value which has been shown square root of $(h_x - n_x)^2 + (h_y - n_y)^2$ and so on should be minimum. For a two-dimensional thing this, other terms will not be present.

We find a note such that this happens and that node will be the root node. If you have more than one candidate, then you will choose the higher of the values of n_x, if n_x values are same higher of the value of n_y and so on. So, this precedence rule also need to be specified and consistently followed or uniqueness of root node.

(Refer Slide Time: 13:42)



(6,3) → will have all nodes in That column

(6,2), (6,4)

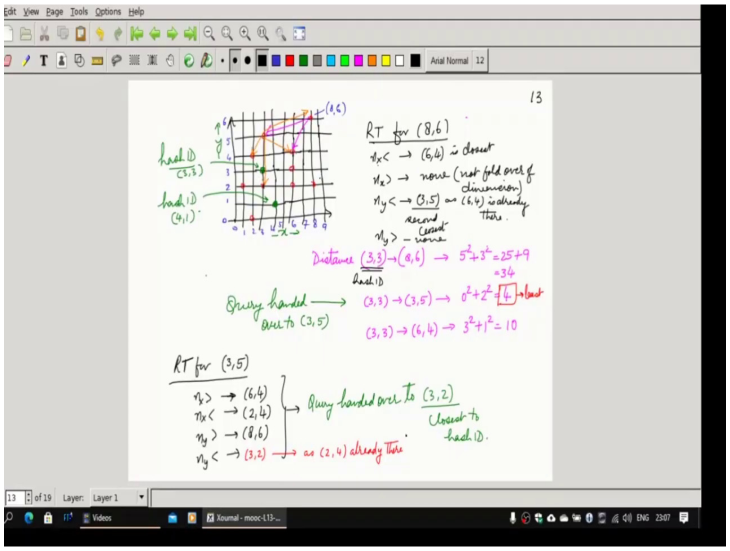
as 'y' lies between 4 and 2, Query handed over to (6,2) ↳ Root node.

Thus Kadmilia → with n bit's → n-dimensional DHT over Torus.

Each dimension → has two position → represented by bit.

So let us now take an example, the same example but now we have to find out hash of 3, 3 how this will be done in this case. Now this case is not same as like this where you are actually using first of all precedence of n_x was taken, find out the closest n_x and then that is freeze than. Find out the closest y, this is the root node was found here. Here it is based on Euclidean distance.

(Refer Slide Time: 14:13)

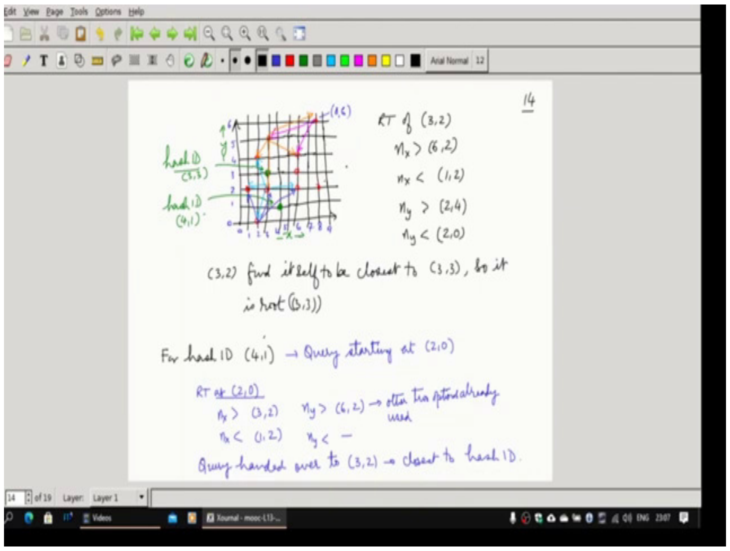


So suppose the same query a 3, 3 with 8, 6 what is going to happen. So for an entry which for which n_x is less than 8, so we can choose and it has be closest one. So we can choose 6, 4 which is the closest Euclidean distance. Then n_x greater than that there is no entry because there is no fold over so there is no entry which is going to be having a n_x greater value. n_y less if I choose, 3, 5 will be the next closest entry because 6, 4 has already been occupied. n_y greater, there is no entry.

Now if I, if we compute the distance of 3, 3 from each one of these three options 3, 3 is the least one, (sorry) 3, 5 is going to have the least distance from 3, 3 which is 4. So, we just send over the query to 3, 4. Now come to the 3, 5 so what happens to 3, 5 how the routing table entry will be there. So, for less than x you find out the closest value it will be 2, 4 which is kept here.

And then phir nx is greater than that, then this 6, 4 will be the closest entry, because this is far off. Then ny greater than 8, 6 because that is only now possible entry which available, which is closest. Ny less than it will be 3, 2 actually. Because this is far off if you compute the distance all these are occupied, this one is also far off this the closest we will there. So now when 3, 3 is going to be measured with all these four nodes, 3, 2 will be the closest one and the entry will be handed over to it.

(Refer Slide Time: 15:51)



14

RT of (3,2)

$n_x >$ (6,2)

$n_x <$ (1,2)

$n_y >$ (2,4)

$n_y <$ (2,0)

(3,2) find itself to be closest to (3,3), so it is root ((3,3))

For hash ID (4,1) → Query starting at (2,0)

RT at (2,0)

$n_x >$ (3,2) $n_y >$ (6,2) → other two options already used

$n_x <$ (1,2) $n_y <$ –

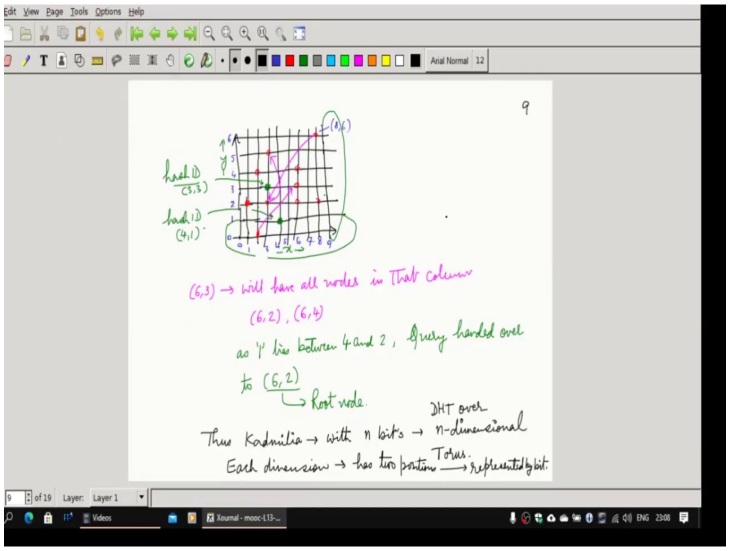
Query handed over to (3,2) → closest to hash ID.

Now 3, 3 and 3, 2 essentially, 3, 2 has to now build up its own routing table, which we will do. Higher side the closest node is this, lowest side the closet node is this, upper side this is the closest note, lower side this is the closest node. So this will be the entry and it will find out how far is 3, 3 from each of these four entries and itself, so it will find this, it itself actually is the closest one so it should be the root node.

Then we can look for another example where hash ID of 4, 1 is chosen and query starts 2, 0. The same thing which I am repeating but now with a different metric. So routing table at 2, 0 what is going to happen? n_x is greater than if I am choosing than 3, 2 will be the closest node, n_x less than 1, 2 will be the closest node, n_y greater than the two already options which were there already occupied, so next best option is 6, 2; n_y less than there is no option.

The query will be straightforward will be handed over to 3, 2 which is ofcourse is the closest. Now 3, 2 entry which is already available here, this is going to compare with all these 4 that we define that 4, 1 is closet to itself, so 3, 2 is the root node in this case. Now this was not the case in the earlier thing there was no Euclidean distance in that case.

(Refer Slide Time: 17:11)



(6,3) → will have all nodes in That column

(6,2), (6,4)

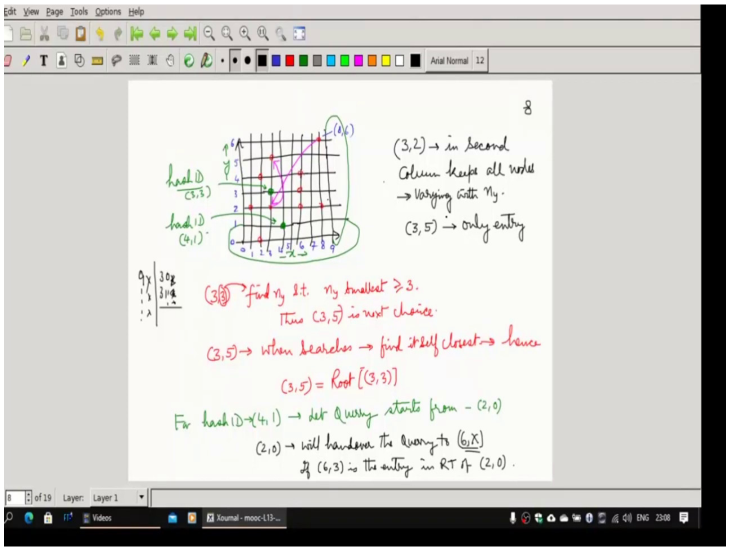
as 'y' lies between 4 and 2, Query handed over to $\underline{(6,2)}$ ↳ Root node.

Thus Kadmilia → with n bit's → DHT over n-dimensional Torus.

Each dimension → has two position → represented by bit.

So, if you go back and check, the root node which we found was for this 6, 2 was the root node while now we are actually having 3, 2 as the node.

(Refer Slide Time: 17:24)



(3,2) → in Second Column keeps all nodes → Varying with n_y.

(3,5) → only entry

(3③) → find n_y s.t. n_y smallest ≥ 3.

Thus (3,5) is next choice.

(3,5) → when searches → find itself closest → hence

(3,5) = Root [(3,3)]

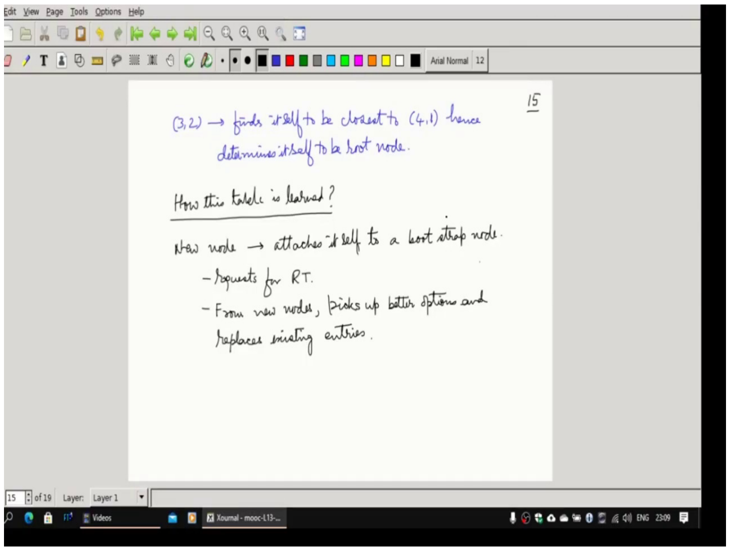
For hash ID → (4,1) → let Query starts from – (2,0)

(2,0) → will handover the Query to $\underline{(6,X)}$

If (6,3) is the entry in RT of (2,0).

And for this one, root node was 3, 5, it was not 3, 2. Now in our case it is Euclidian distance both are 3, 2 because that is, if you look at that is actually closest.

(Refer Slide Time: 17:36)


15

(3,2) → finds itself to be closest to (4,1) hence determines itself to be root node.

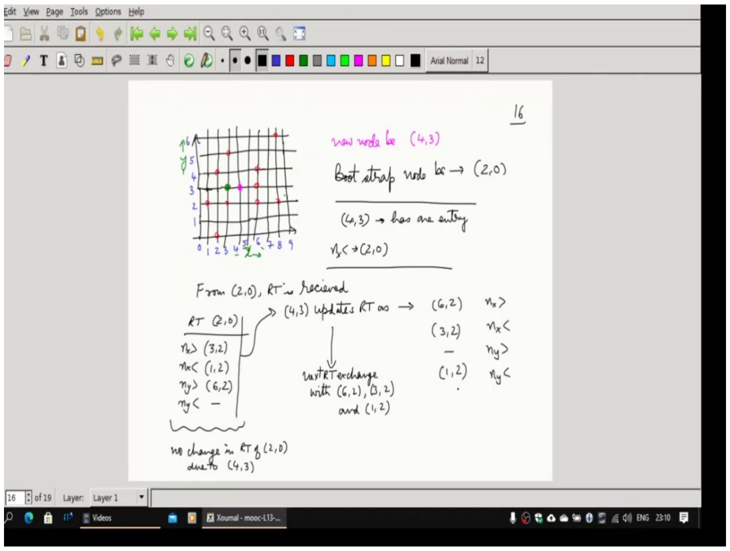
How this table is learned?

New node → attaches itself to a boot strap node.
- requests for RT.
- From new nodes, picks up better options and replaces existing entries.

Now 3, 2 will find itself to the closest and determines itself to the root node. But how these tables are going to be learned, the neighbourhood tables. Actually, you should not call it neighbourhood tables those are normally used in the connotation of (proxy) with the proximity.

So the new node will attach itself to any boot step node and request for its routing table and from the new knowledge, which it’s get its new nodes active information it will pick up the better options and replace the existing entries. If you, if it is better in the existing actually.

(Refer Slide Time: 18:06)



16

new node be (4,3)

Boot strap node be → (2,0)

(4,3) → has one entry

$n_x< \rightarrow (2,0)$

From (2,0), RT is recieved

RT (2,0)
$n_x>$ (3,2)
$n_x<$ (1,2)
$n_y>$ (6,2)
$n_y<$ –

no change in RT of (2,0) due to (4,3)

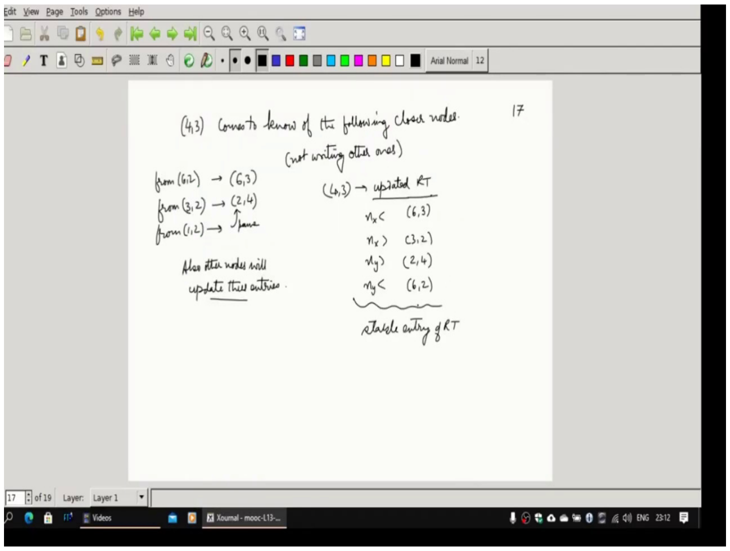
(4,3) updates RT as →
(6,2) $n_x>$
(3,2) $n_x<$
– $n_y>$
(1,2) $n_y<$

next RT exchange with (6,2), (3,2) and (1,2)

For example, new node is 4, 3 it is bootstrap to 2, 0. So 4, 3 somewhere lies here, so 4, 3 is boot strap to this, so only 2, 0 is known to it, only entry is 2, 0 as of now. 2, 0 will now send its routing table to 4, 3. 2, 0 will not be affected by knowledge of 4, 3 because it still is not the better option from for already existing options. So 4, 3 will now update its routing table because it now knows about this, this and this node.

So 4, 3 will pick up larger nx it will pick up this particular node 6, 2, for lower one it will pick up this one. For higher, because its only know of these 3 plus fourth node, nobody else is known so there is no entry for ny greater. For ny less it will again choose whatever the next best possible entry which is 1, 2. So these 3 entries will be learned.

(Refer Slide Time: 18:58)



17

(4,3) Comes to know of the following closer nodes.
(not writing other ones)

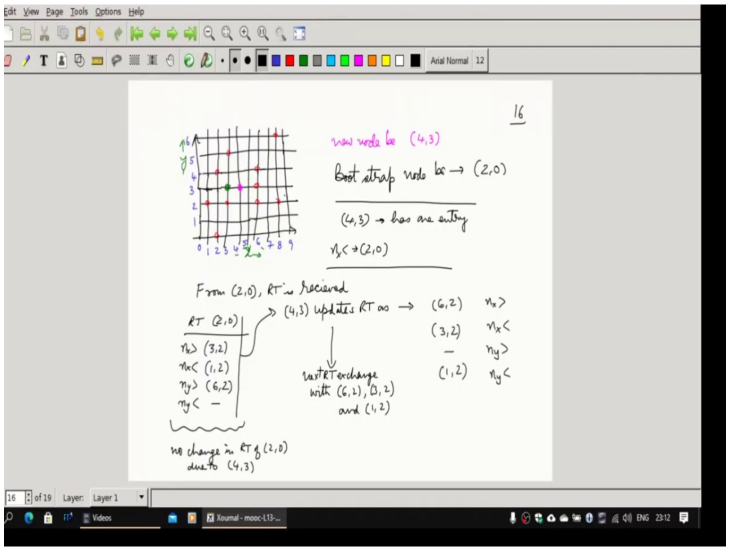
from (6,2) → (6,3)
from (3,2) → (2,4)
from (1,2) → same

Also other nodes will update their entries.

(4,3) → updated RT

$n_x<$ (6,3)
$n_x>$ (3,2)
$n_y>$ (2,4)
$n_y<$ (6,2)

stable entry of RT



16

new node be (4,3)

Boot strap node be → (2,0)

(4,3) → has one entry
$n_x< \to (2,0)$

From (2,0), RT is recieved

RT (2,0)
$n_x>$ (3,2)
$n_x<$ (1,2)
$n_y>$ (6,2)
$n_y<$ —

no change in RT of (2,0) due to (4,3)

(4,3) updates RT as →

(6,2)	$n_x>$
(3,2)	$n_x<$
—	$n_y>$
(1,2)	$n_y<$

next RT exchange with (6,2), (3,2) and (1,2)

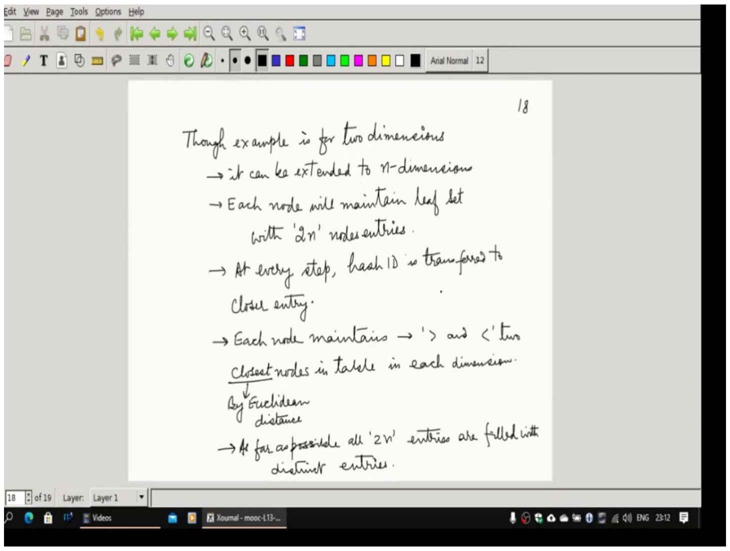
In the next time step it will come to know of new following more closer nodes actually. So, for example 6, 2 will be telling so I am only writing the nodes which are closer. 6, 3 will be (())(19:09) that so 6, 3 will get replaced 6, 2 will no more be there. It will learn about 3, 2 from 3, 2 about 2, 4 so 2, 4 will be learned. From 1, 2 again about 2, 4 only the knowledge will come so that will also be replaced, so this can be observed in the previous figure. So, this guy will tell about this, this person and this person but this is and the new entry which is known is actually this guy.

So 2, 4 is what I have written 2, 4 will be learned from this. Similarly, 6, 2, 6, 2 is going to tell about this person, this person, this and this. So, it will learn about this new guy which can be its own possible replacement entry, this guy cannot be replace entry this nodes he is already aware of. So, based on this you will now replace these entries and after this it will remain stable.

And of course you can see for this guy left side this the one entry, for right side this the one entry, for up either this or this both can be there so it can retain this one. Of course it will come to know of this one also, so if this is going to be with proximity small closer one so you can choose this one actually in that case. And this one is going to be now for the lower y so 6, 2 will again come.

So, this has gone the x plus greater than x and n_x greater than n_x lower than n_y greater than and n_y lower than again this will come. So that is the entry which essentially now gets reflected here, 6, 2 comes here now and this become a stable routing table. Now any this is how the new nodes get inserted, when they get removed again those entries are purged off from the tables and replenished by the better entries even the routing table exchanges are happening.

(Refer Slide Time: 21:00)



18

Though example is for two dimensions

→ it can be extended to n-dimensions

→ Each node will maintain leaf set with '2n' nodes entries.

→ At every step, hash ID is transferred to closer entry.

→ Each node maintains → '> and <' two closest nodes in table in each dimension.

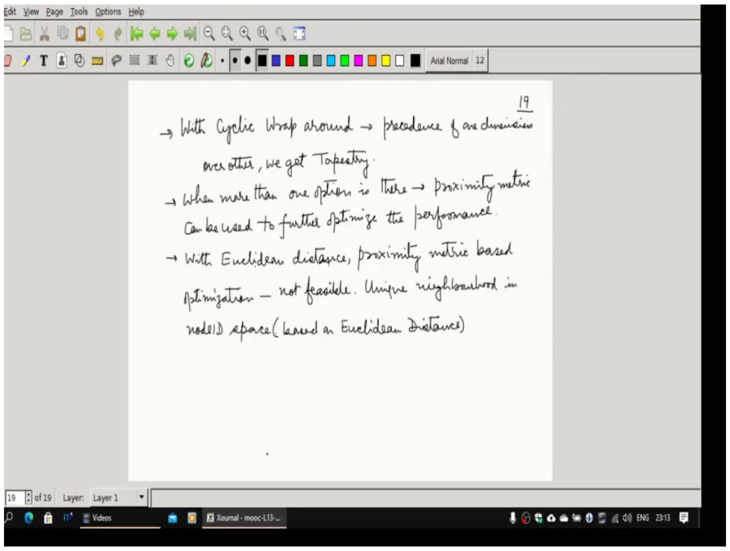
$\downarrow$ By Euclidean distance

→ As far as possible all '2n' entries are filled with distinct entries.

So through this example which was in two dimensions, one thing which is obvious that it can be extended to n dimensions and each node need to maintain a leaf set of 2n nodes entries at most. There can be sometimes less also but two node entries are required most of the times. And at every step hash ID is transferred to a closer entry in terms of the Euclidian distance.

So each node has to maintain greater than and less than entries for two closest nodes in the table each dimension again closest by Euclidian distance. And as far as possible all 2n entry should be filled. So, if you do not get best entry you can find out the second best entry and fill with that.

(Refer Slide Time: 21:41)



19

→ With Cyclic Wrap around → precedence of one dimension over other, we get Tapestry.

→ When more than one option is there → proximity metric can be used to further optimize the performance.

→ With Euclidean distance, proximity metric based optimization – not feasible. Unique neighbourhood in nodeID space (based on Euclidean Distance)

And of course, with cyclic wrap around also this can be done. You can do precedence of one dimensional either doing distance that way, or using Euclidean distance also it can be done on a n dimensional torus. Then, of course whenever more than one option is there use proximity metric to further optimise the performance.

Though in Euclidian distance normally all the nodes which are into the same Euclidian distance only will be the option available to you. So, options will be less so optimization also will be less compared to when you are actually choosing say Tapestry kind of a structure where options are pretty large.

So, this the how the distributor dash tables in multiple dimensions actually work. And this the most generalize framework which can actually exist so almost everything can be explained in this from. So, with that we close the lecture now.

And in the next lecture we will be talking about multi-layer DHT where the multiple services will end up using their own separate layers and a node can participate in multiple DHT layers simultaneously. And of course by while returning the same node which also means there will be multiple routing tables that we will do in the next lecture.