Peer to Peer Networks

Professor Y. N. Singh

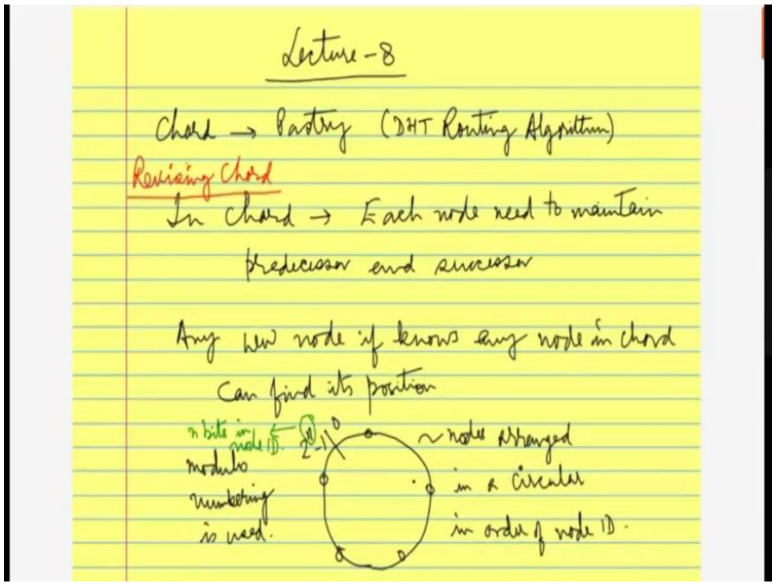
Department of Electrical Engineering

Indian Institute of Technology, Kanpur

Lecture 8

PASTRY Protocol: The Efficient Use of Internet Infrastructure

(Refer Slide Time: 00:13)



Lecture-8

Chord → Pastry (DHT Routing Algorithm)

Revising Chord

In Chord → Each node need to maintain predecessor and successor

Any new node if knows any node in chord can find its position

n bits in node ID.

modulo numbering is used.

2^n-1 0

~ nodes arranged in a circular in order of node ID.

Welcome to lecture number 8 for the MOOC on peer to peer networks. In the previous lecture, I had covered about Pastry, and I have shown you an example that if we just depend on routing table exchanges, then our routing table may not converge correctly. So, one of the essential mechanisms in chord was the use of predecessor and successor both. We can also call it leaf set, Pastry terms it as a leaf set. And with that, we were always able to list with the correct routing table, so convergence was always guaranteed.

So, that is what we had discussed in the previous lecture. But there was a problem in the chord that each node is randomly picking up the node IDs. So, maybe your neighborhood node might be going to some node when you are forwarding a query or trying to publish a key-value pair, which may be all the far off. It may be just the opposite direction on the earth. If you are in India, it the other node; the next neighboring node might be in the United States. And then, the next node may be back to India.

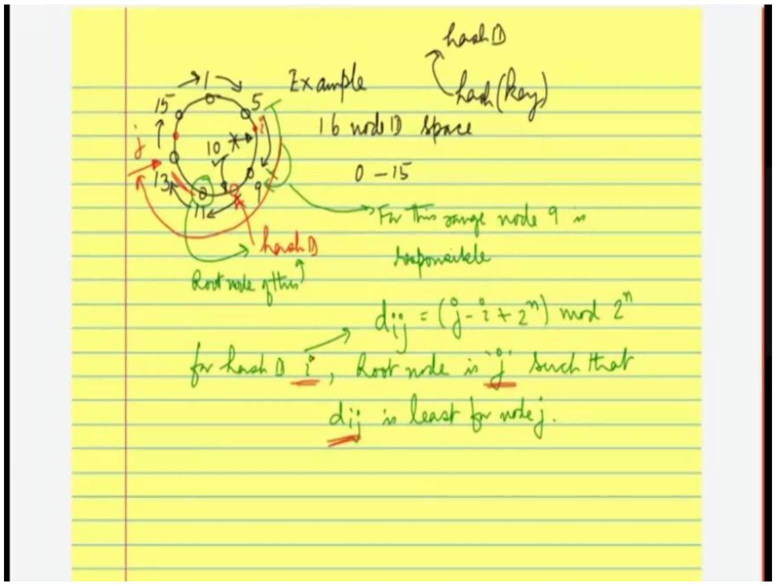
So, proximity is not guaranteed. This basically in the node ID space only we are looking at closeness, and based on that, every time the query, or the key-value, or hash ID for the key, is moving towards the node which is closer in node ID space. But it is not taking the optimal path. So, this was one of the problems; this needs to be resolved. So, that is why the Pastry got invented. We will now be today looking at the Pastry, another DST routing algorithm, which is technically very similar to chord, where the route nodes are defined, except for the minor changes in distance definition.

So, we will be revising the chord first, and then we will move on to Pastry. So just a quick revision of the chord. The chord, in this case, all nodes have to maintain a predecessor, as well as a successor node. And all nodes are supposed to be arranged circularly, as you can see in the figure, so they are placed circularly.

So, this will depend on the node ID. So, if this number is going to be 0, this has to be higher than 0. So, there is no other node which should exist between these two node IDs. They are all arranged circularly, so you start from 0, go all the way to 2 raise power n minus 1.

Suppose there are n bits which are contained in the node Id. And we are using modulo numbering, so we essentially, in this case, once the numbers get finished off, we again go back to 0. So, this circular arrangement is maintained. And in chord usually, we have a mechanism by which if the hash ID lies will be lying somewhere on the circle. And whichever node ID will be immediately after that in a clockwise direction, so that will be the first node that will become the root node. So, I will explain the actual definition of the root node. We had discussed this thing earlier.

(Refer Slide Time: 3:39)



Example

hash ID

hash(key)

16 node ID space

0 – 15

For this range node 9 is responsible

$d_{ij} = (j - i + 2^n) \bmod 2^n$

for hash ID i, Root node is j such that

d_{ij} is least for node j.

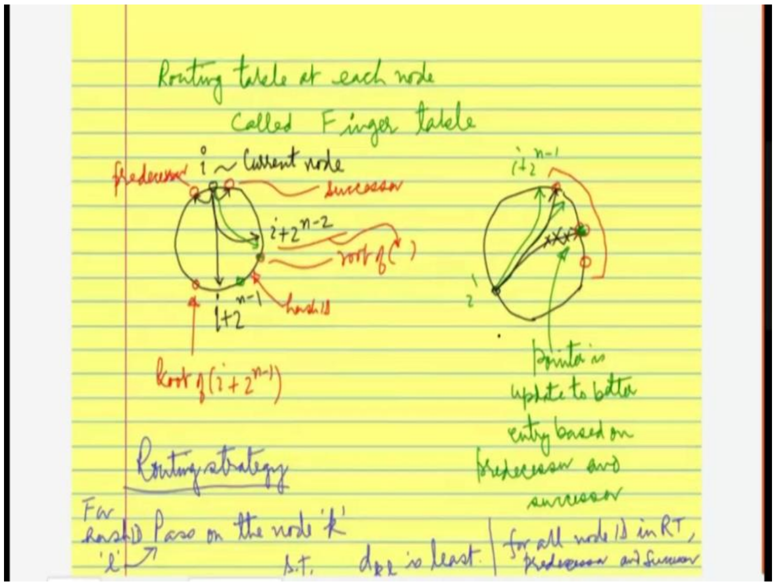
So, this is the way it is going to be arranged. So, I am taking an example of 16 node ID space, so numbers can only be given from 0 to 15. There are some nodes, so when they will be arranged in the circle, after 1, 5 will come, there is no node with a value 2, 3, 4, so then only this can happen. And from 5, there will be 9. So, they are in, remember they are in order. So you are not going to get a 10 here in this position. So, you will not get 10 here. If you want, you will not get 10 here. 10 has to be only after this, so 10 can only come at this location. So, this cannot come here. So, this is going to be correct if I have to insert 10.

So, this is the way things are organized. Now the hash ID, or the key value, so basically there will be a key, and you will be generating a hash, hash function of, the hash of the key. This is what will be your hash ID. So, this hash ID is what will be; for example, hash ID belongs to this particular point in this node ID space, then the root node for this one will be 0 in this fashion. The definition is the node ID that is closest or greater than or equal to the hash ID and smallest in magnitude in a cyclic manner, so that will be the root node as per this definition.

If you have your hash IDs in this range at any point in time, this will be the root node. So, this is how it is going to be happening. We can take this and define this as a, in a mathematical form. So, as a distance metric. This, we had also discussed in one of our earlier lectures. From a point i, so from you take a point i, and to a point j, if you want to find a distance, this distance has to be taken in this fashion. So, this distance will be given by j - i, and this can also be negative, need not be always positive, because, after 15, it is, the circle is completing and coming back to 0, 1 and so on.

So, j - i can also be negative. That is why we can add 2^n. So basically, and then do a modulo operation. So, that will give me the distance always in a clockwise fashion. So, for a hash ID i, the root node will be any node whose node ID is j such that d_{ij} is the least, so you have to choose j such that d_{ij} is minimum for a given hash ID so that that node j will be the root node. So, that what will be the definition?

(Refer Slide Time: 6:27)



Routing table at each node
Called Finger table
Predecessor
i ~ Current node
Successor
$i+2^{n-2}$
root of ( )
$i+2^{n-1}$
hash ID
Root of $(i+2^{n-1})$
$i+2^{n-1}$
i
Pointer is update to better entry based on predecessor and successor
Routing strategy
For hash ID 'l' Pass on the node 'k' s.t. d_{kl} is least. | for all node ID in RT, Predecessor and Successor

So typically, how the routing table will be there. The routing table at each node is called the finger tables already discussed in case of a chord. Usually, this is the current node i. so every node should maintain a successor and predecessor, immediately any node which is smallest in number but is higher than i, that has to be retained cyclically. And a predecessor, the previous one. These two always have to be maintained.

And after that, you can find out the root of halfway across. So, the total number of node IDs is 2^n. So this $2^n/2 = 2^{n-1}$. So, this actually will when I add to i, this will point to diametrically opposite point. So, whatever is the root node of this, this will be the root node; if this, no node exists in this range. So, this will be kept in the table. So, in the finger table. Similarly, for quarter 2^{n-2}, I will add to i and find the root node.

We will keep all the entries. Ultimately, at 2^0 i +1, so the root node of i + 1 will always be the successor. So, the predecessor is always kept for optimization purposes. Otherwise, you will not be able to do, you cannot stabilize the routing table to the correct entries, as we had seen in the previous lecture. And of course, now the routing strategy is straightforward. For hash ID l, so l is a hash ID that you want to find out. And there are many nodes which are going to present.

For example, this i contains this node, this and these three nodes. It will choose a node such that from the hash value, we have to select a node such that from the node listed in my finger table, the distance should be minimum from that node to the hash ID. For example, if the hash ID is supposed at this position, this is the hash ID. So, in this case, I should now hand over the query to this particular node. This node will now further determine if its successor is immediately after this node after this hash ID, so it should be the root node.

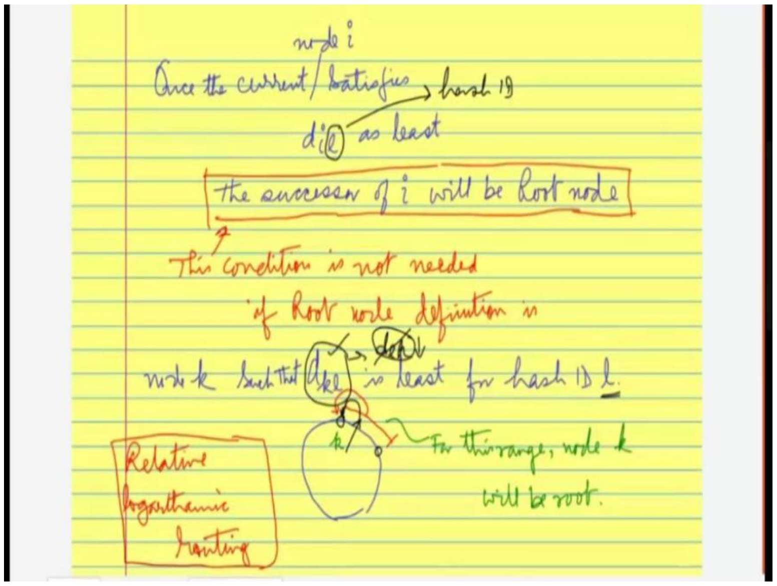
It will then inform the successor that you are the root node, and this is the hash ID, then it will look into its database. Every time you have to use it, you are trying to minimize it; you are choosing k such that d_{kl} is the minimum thing and hand it over. At some point, the node will figure out that it is the closest guy. And then that guy will figure out the successor and hand over the query to him, and tell him that you are the root node. That has to be told him specifically; otherwise, he will look at his finger table, and he will find out somebody else who is closer to that hash ID.

So, routing has to stop here. And this guy will then has to specifically ask him to supply the value pair, value key-value pair from his database. We can modify this definition slightly and make it simpler. I will do that later. For example, I am always pointing here; this is the same thing I had mentioned in the previously recorded lecture that this is pointing here.

So, suppose there is a better pointer from predecessor and successor. In that case, somebody is closer to the halfway across entry, so halfway across the entry. For example, it is pointing to somewhere here, so then n it is now pointing here, so predecessor is at this point, the successor is here, so this remains the root node.

But if my halfway entry is pointing here, so this is going to be, this is i, so this is i + 2^{n-1}, and the finger table contain this nodes entry. Whenever this node sends back the information about predecessor and successor, this node will figure out the better entry to change the pointer instead of this; the pointer will be modified from here. So, this entry will no more be valid. So, this how you keep on optimizing the pointers in the finger table. So, this predecessor, successor, actually do that job. So this we had discussed earlier also.

(Refer Slide Time: 10:55)



Once the current node i satisfies d_{il} as least → hash ID

$\boxed{\text{The successor of } i \text{ will be Root node}}$

This condition is not needed if Root node definition is

node k such that d_{kl} is least for hash ID l.

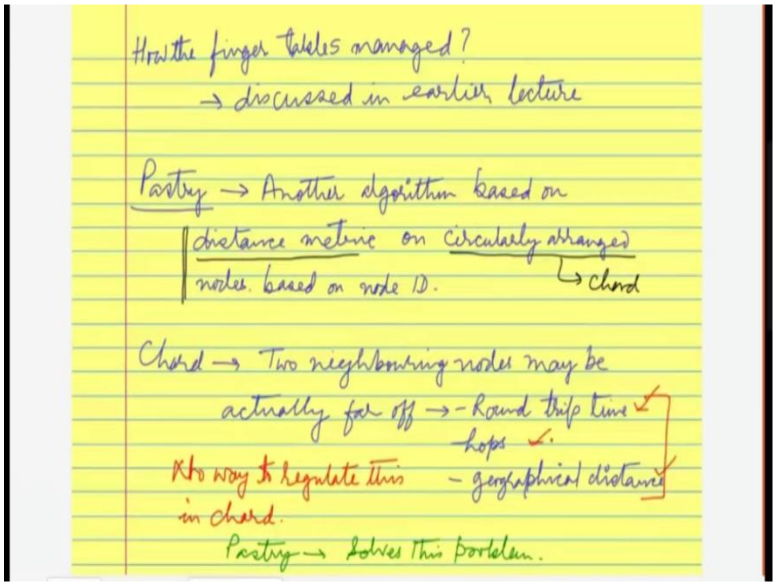
k
For this range, node k will be root.

$\boxed{\text{Relative logarithmic Routing}}$

So once the current node i will satisfy the d_{il}, where l is the hash ID which we are searching this hash ID, this what we had used in the earlier slide, is the least value. The successor of i will be the root node. This is what I mentioned earlier. We can avoid this specific condition if we choose my root node definition slightly differently. We can decide that root node, node k, such that d_{kl}; remember earlier, when the root node was being defined, it was d_{lk}, so we chose k such that d_{lk} gets minimized.

But now we are not doing this; we are doing this one. So, node k is chosen such that d_{kl} is the least for the hash ID l, so that k will be the root node. So, in this case, if this is the range of hash ID, the root node will be the previous guy, the predecessor to, each one of, this hash value, the predecessor is this, so this will be the root node. Now, this is relative to logarithmic routing. Now when I want to, in this case, only numeric ID based search can only happen. I cannot choose based on geometric or actual distances, which are closer to me. I need something different. So, that is what Pastry did.

(Refer Slide Time: 12:20)



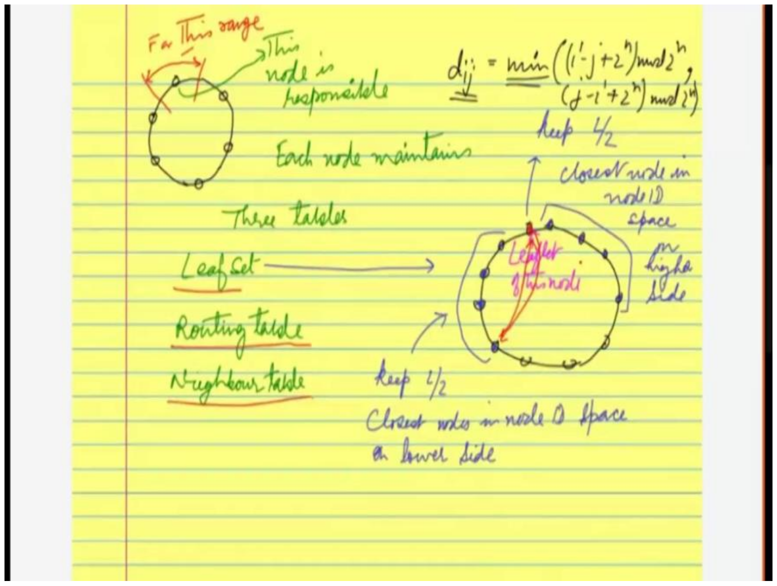
How the finger tables managed?
→ discussed in earlier lecture

Pastry → Another algorithm based on
distance metric on circularly arranged nodes based on node ID.
↳ Chord

Chord → Two neighbouring nodes may be actually far off → - Round trip time ✓
- hops ✓
- geographical distance ✓

No way to regulate this in chord.

Pastry → Solves this problem.


For This range
This node is responsible
$d_{ij} = \min((i-j+2^n) \bmod 2^n, (j-i+2^n) \bmod 2^n)$
Keep 1/2
closest node in node ID space on higher side
Each node maintains
Three tables
Leaf Set
Routing table
Neighbour table
Leader of this node
Keep 1/2
Closest nodes in node ID space on lower side

And, of course, I had talked about finger table management. And this was also part of the previous lecture. We will now use this Pastry; this algorithm uses the proximity metric, or actual distances will also be accounted for. But the root node definition, or the node ID, node ID space, the distance metric is still based on the circularly arranged nodes. So that is the key thing. This is the same thing as what happens in the chord.

So, still, the arrangement is circular. And two neighboring nodes may be far off in case of a chord, even in terms of round trip time, hops, or geographical distance, and this cannot be done in a chord, but Pastry can do it I mentioned earlier. So let us see how it does. So, first of all, what is the root node definition in the chord in the Pastry? If a node is lying here, so the next successor is here, the predecessor is here, so halfway across will divide. This node will be the responsible node for this whole range or called the root node for this ring.

So, it is the absolute distance that is going to be critical here. So technically, your d_{ij} will be given by a minimum of i-j + 2^n mod 2^n, or so this is what is going to be the distance metric now. So, the node closest to the hash Id will be the root node in this sense. This distance represents this, so I am going either this way or going the other way around, either way, whatever be the distance, so i to j or j to i, both I am taking, so this will be the root node.

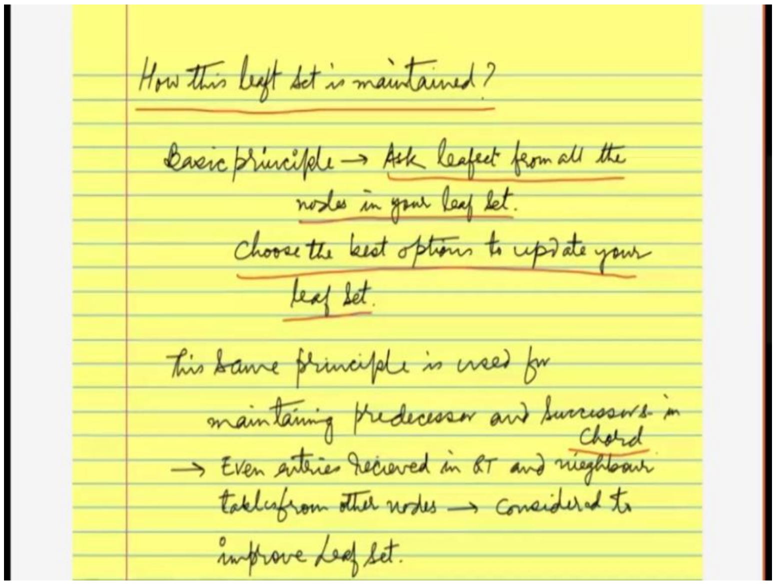
But there can be a problem actually, that a node your hash ID may be precisely between the two nodes, in this case, who will be the one, so both are equidistance. So, in that case, you can choose the next higher entry all the time, but it has to be consistently done with everybody. Otherwise, you will not be able to find the root node. The root node has to be unique in the system. So, each node in Pastry will maintain three tables, on is called leaf set, so this is an extension of just maintaining predecessor and successor in the chord. This is an extension for that. The routing table is now slightly different here. It is not based on finger tables.

And there is one-third thing that is neighbor tables, which essentially now look into taking care of these three distances mentioned here, these three distances are being taken care of, so anyone can choose it. So, but that is being left to the implementation of the algorithm.

So, the leaf set, first of all, let us define what the leaf set is. So, its size will be given by some parameter l. I will tell you what the l will be actually. So, l by 2 nodes, which are higher and closest to the current node, those will be the part of the leaf set higher side, and there is a leaf set lower side, which is l by 2 lower nodes, but which are closest to this current node.

So, this set of, for example, add node, 8 nodes will form a leaf set, where l is now 8 in this case, so 4, 4. So, that is what leaf set is; every node is supposed to maintain it. Even if some node dies off, this guy will always be exchanging leaf sets with all the leaf set members. So, its leaf set will be given to this person. So, this will be changing the given leaf set to him. This guy will be communicating leaf set back to him. If some node dies off, they can always repair it and always maintain 8 nodes in their leaf set.

(Refer Slide Time: 16:26)



How this leaf set is maintained?

Basic principle → Ask leafset from all the nodes in your leaf set.
Choose the best options to update your leaf set.

This same principle is used for maintaining predecessor and successors in Chord

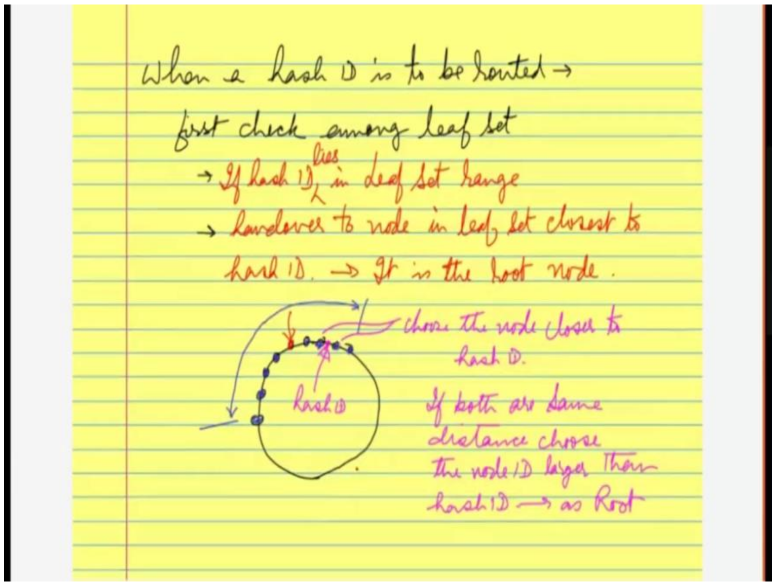
→ Even entries recieved in RT and neighbour tables from other nodes → considered to improve Leaf set.

So the basic principle is to ask the leaf sets from all the nodes in your leaf set and choose the best option to update your leaf set. So that is going to be the key rule, which is maintained by all the nodes. So all the nodes will keep on maintaining the leaf sets accordingly. And whatever node you choose, you start taking leaf set ultimately you will end up in fine building up your own optimum leaf set, and you will stabilize on to that. So, all the nodes reported, you usually do a, if you are doing ping-based proximity, closeness, based explicitly on node ID, there is no ping test here to be done based on node ID.

So, it is a straightforward algorithm in this case. A similar thing is also done in the neighbor table case, but we will use a metric, which is defined by one of these three, or maybe something else whatever the application developer decides. If you note down that the same principle we were using, so leaf set l-value was 2 in a chord. So, in this case, the value of l can need not always be 2. It can be higher; it can be 4, it can be 8, it can be 16, or 32, whatever as an application developed you decide.

Another important thing is that the other nodes send even the received entries, and the routing tables or the neighbor tables are sent from other nodes; even those entries are also considered to make a better choice for your leaf set. So, this is an ongoing process all the time. Leaf set, usually, will not be required, but you should always consider all the new information arriving at you for faster convergence.

(Refer Slide Time: 18:06)



When a hash ID is to be routed →

first check among leaf set

→ If hash ID $\overset{\text{lies}}{\wedge}$ in leaf set range

→ handover to node in leaf set closest to hash ID. → It is the root node.

choose the node closer to hash ID.

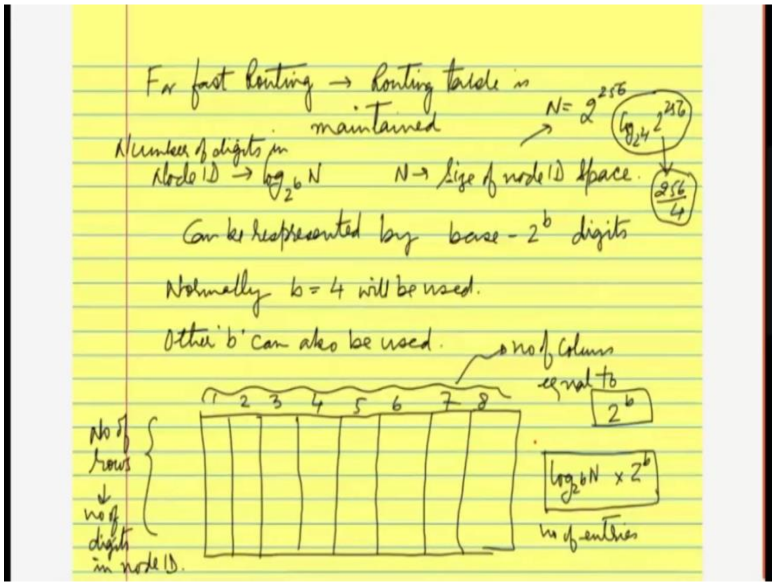
If both are same distance choose the node ID larger than hash ID → as Root

Now when a hash ID is to be routed, so how the routing has to be done? So, routing here is slightly tricky. First of all, you will always use the routing table, and once you reach here and find out you are the closest node, you will start using your leaf set for doing the routing. You get a hash ID; this guy gets the hash ID, which is at this point. And there is no more closer node; this is the only closest node as per the routing table.

I will talk about what closeness is. Then you start looking at your leaf set. So, it finds out this guy is actually between these two. So whichever, it will just send it over to one of them, or it will find out which one is closer. If both of them are equal, it will give it to the higher one as per the consistent rule. I had been following the higher node. Or this node 1, if that is, whichever is closest that we have picked up and entry will be passed on to that.

That guy will again look at his own leaf set and find out he is at the closest node to look into the database for the corresponding key-value pair, or rather he will publish, depending on whether it is a query message or a publish message.

(Refer Slide Time: 19:25)



For fast Routing → Routing table is maintained

Number of digits in Node ID → $\log_{2^b} N$ N → Size of node ID Space

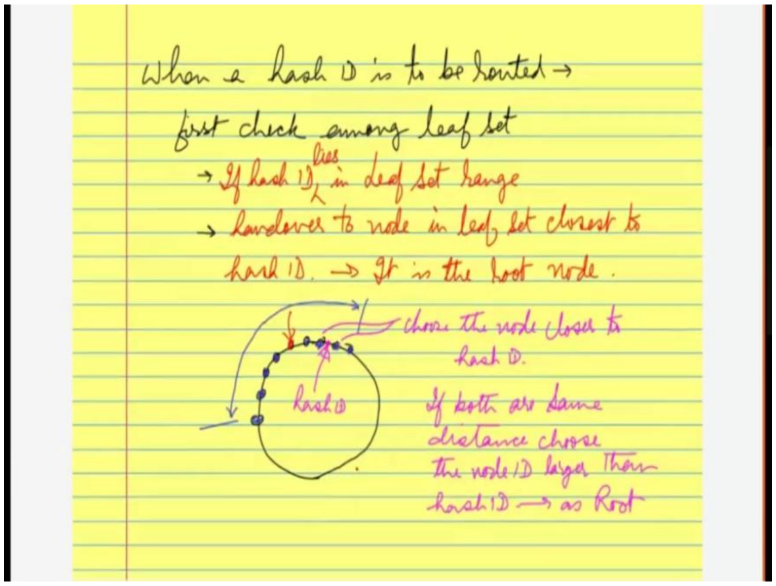
$N = 2^{256}$ $\boxed{\log_{2^4} 2^{256}}$ → $\boxed{\frac{256}{4}}$

Can be represented by base - 2^b digits

Normally b = 4 will be used.

Other 'b' can also be used.

1 2 3 4 5 6 7 8
no of Colums equal to $\boxed{2^b}$
No of rows ↓ no of digits in node ID.
$\boxed{\log_{2^b} N \times 2^b}$
no of entries



When a hash ID is to be routed →

first check among leaf set

→ If hash ID lies in leaf set range

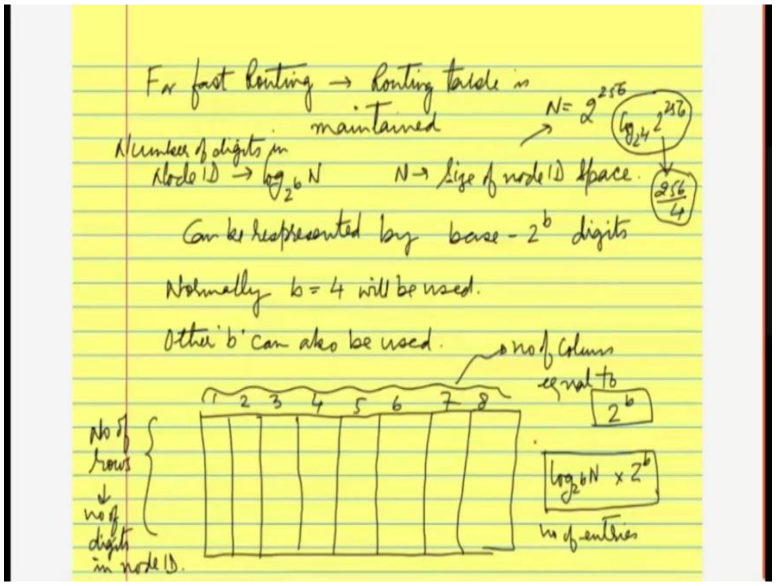
→ handover to node in leaf set closest to hash ID. → It is the root node.

choose the node closer to hash ID.
hash ID
If both are same distance choose the node ID larger than hash ID → as Root

You are now coming to the routing table, how the routing table is built. So fast routing, routing table has to be maintained, because you do not need a routing table, you can work without the routing table, because you can keep on passing through these leaf sets, but every time you can jump at most by 4.

So, you can do 4, 4 jumps, so if there are n nodes away, you have to go, so if there are total n by 2 nodes are there, halfway across if they are uniformly distributed, so, n by 8, roughly those many hops will be required to reach to your root node in the worst-case scenario. So that is not desirable typically when n is very large. So you would like to go into log n steps instead of some constant steps into n.

(Refer Slide Time: 20:10)



For fast Routing → Routing table is maintained

$N = 2^{256}$ $\log_{2^4} 2^{256}$ → $\frac{256}{4}$

Number of digits in Node ID → $\log_{2^b} N$ N → Size of node ID Space.

Can be represented by base - 2^b digits

Normally b = 4 will be used.

Other 'b' can also be used.

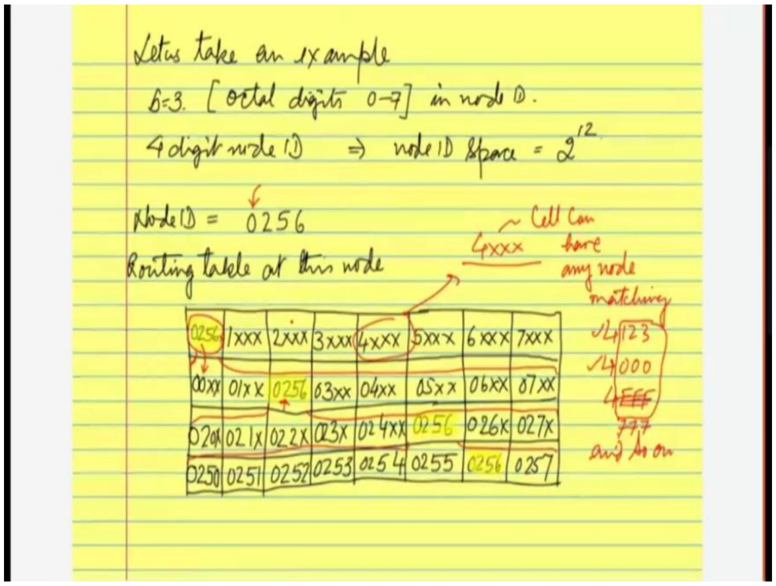
1 2 3 4 5 6 7 8
no of Colums equal to $\boxed{2^b}$
No of rows ↓ no of digits in node ID.
$\boxed{\log_{2^b} N \times 2^b}$
no of entries

So, for this, the routing table is maintained. This is going to be pretty fast. So in the case of n, if it is a size of node ID space, n for example, it is 2^{256}, so 256 bits can represent a node ID. Then the number of digits which will be required will be $\log_{2^b}$ N. With 2^b, b is the number that we have to define. So, this is base 2^b digit has to be used for representing those node IDs. So, I can use that, so b can be 3, it can be 4. Normally b equals 4 is used, but in our example, we will take b equals 3 for simplicity because my tables will be smaller.

We can now even choose 2 that actually can be determined; it can be anything. It does not matter. If you choose b is equal to 1, then we have only two digits 0 and 1 are there, and this whole thing will degenerate to Kademlia. You do not require, in this case, this table will generate to Kademlia, but of course, the distance metric is going to be different in Kademlia and here. So, once I explain Kademlia, you will be able to appreciate that. But as of now, this can be anything. So, the number of rows in the routing table, we have to understand this, the number of rows will be equal to the number of digits in the node ID.

If there is a 4 bit, b is equal to 4 is chosen, so I am now using a 4-bit hexadecimal digit for representing the node IDs. So, how many hexadecimal digits can be there, 256/4? So, those many will be there. And you will end up going to have those many rows, which will be presented here. Depending on, if I am going to use a hexadecimal number, there will be 16 columns; if b is equal to 3, there will be 8 columns. So, the number of columns will always be equal to 2 raise power b. So, 2^4 is 16, so 16 columns will be there for a hexadecimal, base 16 digits if used to represent the Pastry routing.

(Refer Slide Time: 22:22)



Lets take an example

b=3. [octal digits 0–7] in node ID.

4 digit node ID ⇒ node ID Space = 2^{12}

Node ID = 0256

Routing table at this node

0256	1xxx	2xxx	3xxx	4xxx	5xxx	6xxx	7xxx
00xx	01xx	0256	03xx	04xx	05xx	06xx	07xx
020x	021x	022x	023x	024x	0256	026x	027x
0250	0251	0252	0253	0254	0255	0256	0257

4xxx ~ Cell can have any node matching

✓4123
✓4000
~~4EFF~~
~~777~~
and so on

So, let us take an example. So here b is equal to 3, which has been taken, and this can now represent octal digits. The digits can be from 0 to 7. You are not allowed to have 8, 9, 10 and so on in the node ID. And I will take four-digit node ID, which means 2 raise power 3 multiplied by 4. So, 2 raise power 12, those many, the node ID space, which is still actually large.

So, let us take a node ID to 0 to 256 first of all. So what the routing table will look like? So, this will also explain how the routing table gets done. So let us, the first digit will now derive the first column. So, the first digit I will take. So, when the first digit is 0, this node itself can actually be put up here. So, there is no nothing else; only it, the same node entry, will just remain here. Now the first bit is 1, I will keep here. And I can choose anything, anything which can match with 1 x x x.

Which means 4 x x x here; take this example. If the cell, this particular cell, will have this entry, it can have any node that matches this. So, 4 1 2 3 matches, 4 0 0 matches, x means it can be anything. Sorry, I should not have written this F F F here because I am talking about the octal system. So, let us call it 4 7 7. So there is no entry as of now, there is only one node, so I am just leaving it as it is.

So this means there is nothing. Then 2 x x x, any entry which matches with this. 2^9 possibilities exist. Anyone of them can be placed here. But we are going to put a restriction on this what can be placed here. As when doing peer to peer routing, we want to have fewer network resources to be consumed. We are looking going to have some kind of a proximity metric beside which entry has to come in when there are multiple possibilities.

Similarly, for three, anything which is matching. So, each of these blocks can have an entry from 2^9, that much size block. One of them is 2^9, entry is possible. So, this block is representing 2^9 possibilities. Now, coming to the second column. So, 0 remains as it is. So, this will remain at 0.

I am not looking at the second digit. So, the second digit is 0, so this entry has to be maintained here. And then n has to be x x.

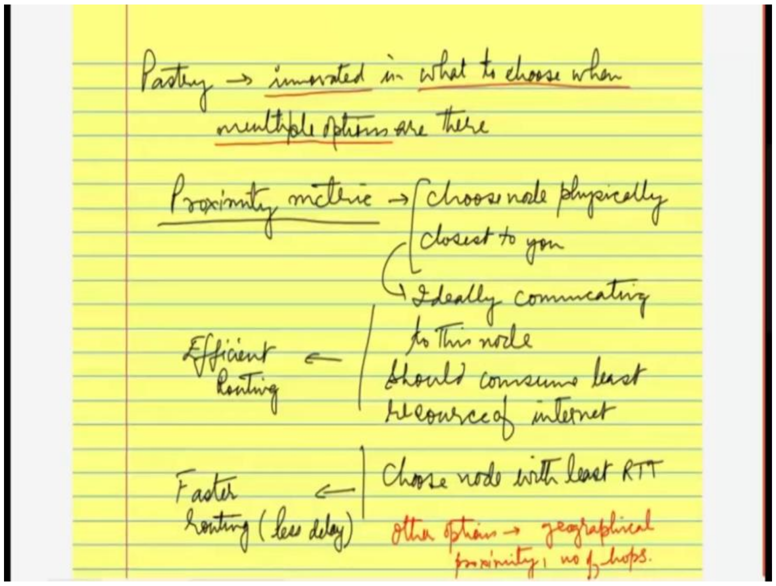
So, 5 cannot come here. It has to be, can be anything. So 0 2 5 6, I will be keeping here at 2 at this place, because of 0 2 matches with this. 0 0 is not matching with this, so it is x x. Anybody come here. This l represents, will be containing one entry out of possible 2^6 entries. So, three bits here and three bits, so 6 possible entries, one of those will be coming here. 0 1 x x and so on, here 0 2, it matches, so 0 2 5 6 I will just put as it is. That is the closest; you have to be closest to yourself.

Then 0 3 x x, 0 4 x x. So, each of them represents a block of 2 raise power 6 sizes in the node ID space. So next entry is, technically, what I have done is, I have taken this particular thing, and I have expanded in this whole row. 0, that is why gets fixed, everywhere in the highest order place. Now in the second place will change 0 1 2, so wherever there is 2, I am repeating this entry. Now, 0 2 gets fixed here. And I am going to expand this into this third row. 0 2 0 will come here. So, I do not have any entry matching this.

There will be total 8 entries which are possible which can come here if they exist. There is currently only one node; that is why there are all left all. So, as I go along, you will appreciate this. So, 0 2 1 x, similarly, it goes on. Here 0 2 5 6 comes in. So, the same node will appear again. And subsequently doing the same way, I can only take 0 2 5 0. Any entry which is in black actually node does not exist. I am going to use the color to identify. But 0 2 5 0 if it exists, will come here. Otherwise, there is no entry here.

All nodes that are different only at the unit place, the lowest octal number place, digit place, so those will be all 8 numbers coming here. 0 2 5 6 exists, so it is going to be present here. I have highlighted with yellow here. So, what Pastry further did, because there were many possibilities, many nodes can exist which can come. It is representing a number directly; this is representing 8. One of the 8 will be coming; here, one of the 2^6 will be coming. Here one of the 2^9 will be coming. So, when you have multiple choice, which one to pick up.

(Refer Slide Time: 27:36)



Pastry → innovated in what to choose when multiple options are there

Proximity metric → choose node physically closest to you

Ideally communicating to this node should consume least resource of internet

Efficient Routing ←

Faster Routing (less delay) ← Choose node with least RTT

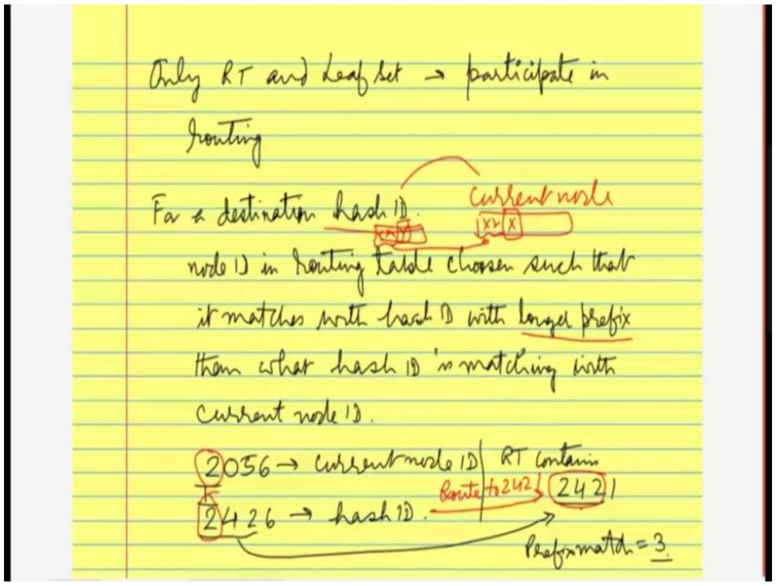
other options → geographical proximity, no of hops.

Pastry innovated here, and they said that we could choose when multiple options are there, and the choice will be based on the physical distance. How this physical distance has to be defined actually? Ideally, communicating to this node we choose should consume the least resources of the internet; that is the idea. So, here the routing will become more efficient; it will become efficient routing compared to chord.

And this efficiency is much higher than a chord. If I want a faster route, I can choose the options based on the least round trip time. I can do a ping and do the ping. Ping is a command used in any machine that can be used to find out a round trip time. The UDP packet goes all the way to the other side and comes back, and based on that, you will figure out whether the time is delayed.

After transmitting many packets, you just make half of it, do the average and take half of that, and that will be your one side delay and choose whichever is the least one. This will make your routing very fast. So other options, not only RTT, you can choose. Your difficult proximity, if you can identify from IP address longitude and latitude, you can do it based on the number of hops you have to use in that case trace route command to find out the number of hops.

(Refer Slide Time: 29:00)



Only RT and Leaf set → participate in routing

For a destination hash ID. current node

node ID in routing table chosen such that it matches with hash ID with longest prefix than what hash ID is matching with current node ID.

2056 → current node ID | RT contains

2426 → hash ID. Route to 242 | 2421

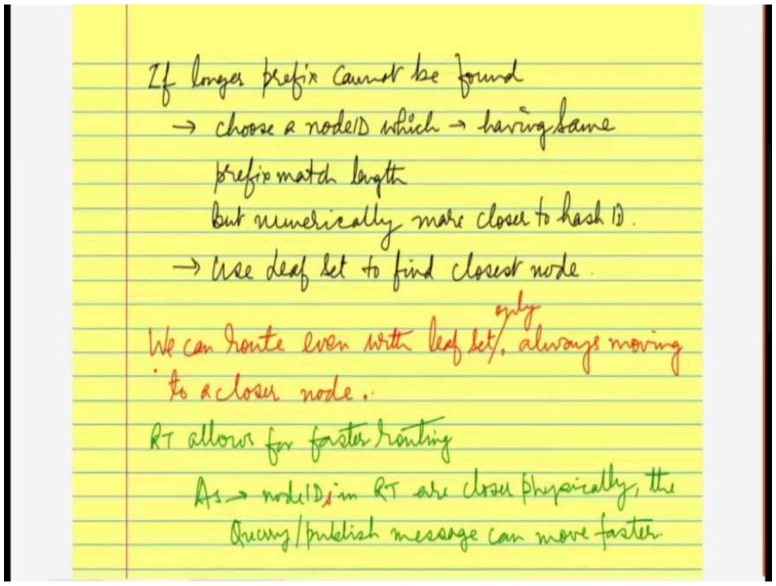
Prefix match = 3

It is essential for routing, only the routing table and leaf set participants; the neighbor table never participates in the routing table; it is not supposed to actually. So, if you have the hash ID being given to you, how the routing happens? Node ID will now search, look at the hash ID. Now hash ID if we compare with the current node, it must be having some bits present here so that some bits will be matching in the hash ID, so maybe these two match with this.

The third one is not matching. These this is different. So, match prefix match is only two digits in this case. So, node ID will be from the routing table; it will search, it will choose such that the, it matches with hash ID in a longer prefix. Here it is matching 2 with the current node, so I need to choose an entry that matches three or higher digits on the prefix side.

So, whatever your current is matching with the current one, it has to be longer. For example, 2 0 5 6 is the current node ID, 2 4 2 6 is the hash ID, and the routing table of 2 0 5 6 contains 2 4 2 1. You can see that it is only matching with one digit with the current node. But with this entry, it is matching with 3. And this is the longest match, so that you will hand over this query for this hash ID to 2 4 2 1. So, the prefix match is now 3; it is not 1.

(Refer Slide Time: 30:35)



If longer prefix cannot be found
→ choose a nodeID which → having same
prefix match length
but numerically more closer to hash ID.
→ use leaf set to find closest node.

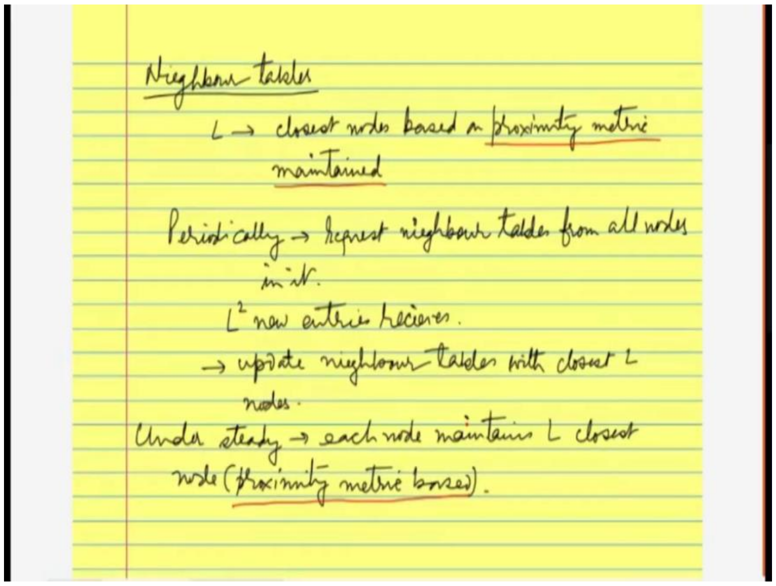
We can route even with leaf set only, always moving
to a closer node.

RT allows for faster routing
As → nodeIDs in RT are closer physically, the
Query/publish message can move faster.

If somehow you cannot find a longer prefix, then, or a hash ID, then it is already matching with you at some places, you cannot find a longer prefix. You can choose a node ID, which has the same prefix match as what it is having with you, but it has to be numerically closer to the hash ID and choose the node. Now, if you cannot, if you end up finding out that you are the only guy who is the closest to the hash ID, and there is no node which is having a longer prefix match, in that case, you have to just switch over to the leaf set based routing.

So, that is for the local optimality. So, you do a faster routing, faster jumps with a routing table. Then, when you reach very close to the root node, you start using leaf sets. So, of course, I have mentioned it here that you can only use leaf sets if you want. But that will be slower. So, these routing tables make it very, very fast. I have explained this same principle in the earlier slide, which you can use for routing the leaf set, any hash ID once you can reach the pair.

(Refer Slide Time: 31:53)



Nieghbour tables

$L \rightarrow$ closest nodes based on proximity metric maintained

Periodically $\rightarrow$ request nieghbour tables from all nodes in it.

L^2 new entries recieves.

$\rightarrow$ update nieghbour tables with closest L nodes.

Under steady $\rightarrow$ each node maintains L closest node (proximity metric based).

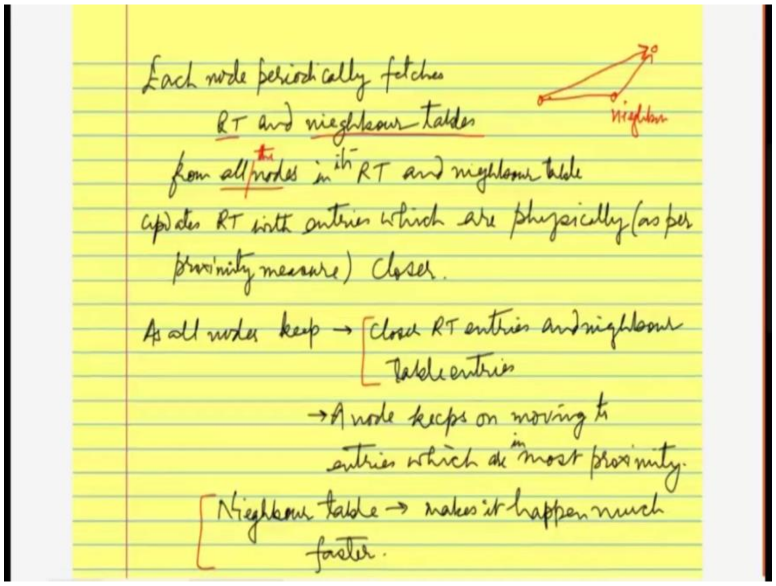
Now, let us come to neighbor tables; what happens to the neighbor tables. You will again choose L closest node based on the proximity metrics, so maybe the number of hops or ping, or maybe RTT, you will select the L closest one. So, there is no node ID thing to be considered here. These are very close to you. If I am in IIT Kanpur and try to do it, I will pick up all L nodes from IIT Kanpur inside me because my round trip time to the outside will be larger.

My hop count will also be larger when I go to the outside world. So, they will become kind of my proximity metric. So this essentially solves a big problem. It makes things efficient; you will appreciate it once we come to choosing the nodes. So, periodically, you will require the neighbor table from all node tables in your neighbor table. So, when each one of your neighbor tables, there are L such nodes you have in your neighbor table, they will give their entry and their entries to you.

You get L square new entries. And now, you will do a ping test if you are doing RTTP. If you are doing several hops, you will do a traceroute to each of them and find out the best L and update them. Your neighbor table will always keep on updating. So, in the beginning, for example, if I am connected to somebody in the US, he gives his neighbor table to pick up somebody closer to me. So, the moment I find somebody closer to me, where the entry is coming from the routing table, or in my leaf set, the moment somebody comes close to me, I will just pick up and hop on to that.

I will keep on replacing it. And ultimately, I should converge on a situation where all nodes in my neighbor table are in my proximity. So, under the city-state, each node will maintain L closest node. This usually can be a separate function, which can be plugged into the peer to peer libraries or middle where we are using, and then the proximity can be used for doing the more efficient routing.

(Refer Slide Time: 34:00)



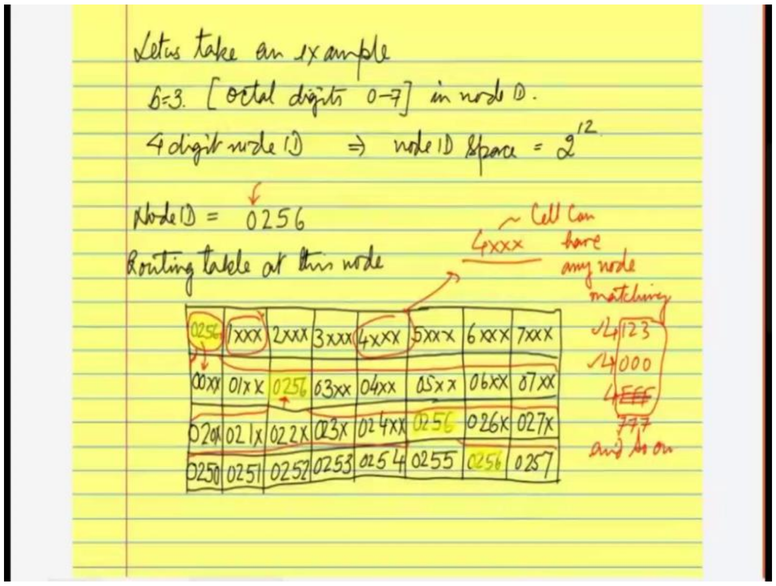
Each node periodically fetches
RT and nieghbour tables
from all the nodes in its RT and nieghbour table
updates RT with entries which are physically (as per proximity measure) closer.

As all nodes keep → [closer RT entries and nieghbour table entries
→ A node keeps on moving to entries which are in most proximity.

[Nieghbour table → makes it happen much faster.

So, each node periodically will fetch routing table and neighbor tables, so remember, it has to bring routing table and neighbor tables, from all the nodes, from all the nodes present in its routing neighbor table. So, whatever is there in your routing table and neighbor table, you will pick up each of them and ask them to send you their routing and neighbor tables. Based on the entries you will arrive at, you will update your routing table entries that are physical as per the proximity metrics are closer to you.

(Refer Slide Time: 34:40)



Letus take an example

b=3. [octal digits 0–7] in node ID.

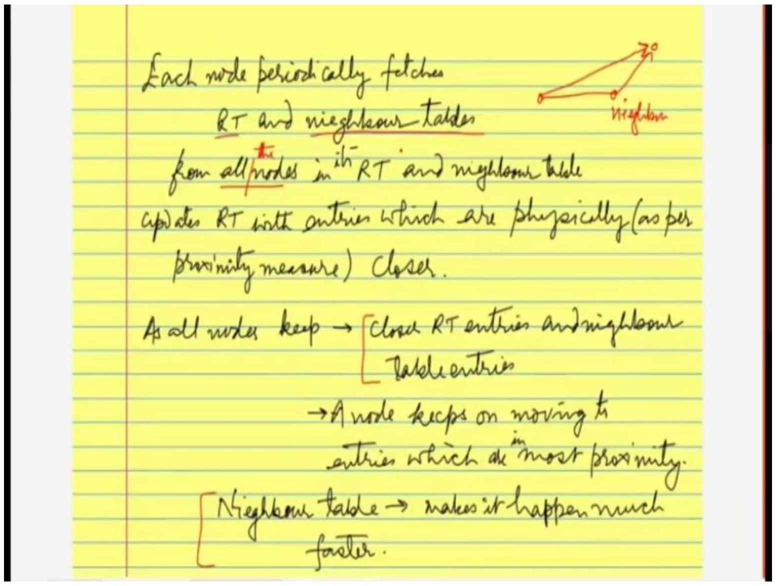
4 digit node ID ⇒ node ID Space = 2^{12}

Node ID = 0256

Routing table at this node

0256	1xxx	2xxx	3xxx	4xxx	5xxx	6xxx	7xxx
00xx	01xx	0256	03xx	04xx	05xx	06xx	07xx
020x	021x	022x	023x	024xx	0256	026x	027x
0250	0251	0252	0253	0254	0255	0256	0257

4xxx
Cell can have any node matching
4123
4000
4EEE
777
and so on



Each node periodically fetches
RT and nieghbour tables
from all the nodes in its RT and nieghbour table
updates RT with entries which are physically (as per proximity measure) Closer.

As all nodes keep → [Closer RT entries and nieghbour Table entries

→ A node keeps on moving to entries which are in most proximity.

[Nieghbour table → makes it happen much faster.

So, you may end up getting, for example, a routing table here; for this particular entry, you may get, say, ten possibilities once the exchange happens. Once the ten possibilities come, you will pick up one of the guys who is the closest to you. And put that here. So, whenever you are doing the routing to a more extended domain, for example, this is a very long hop as far as the numeric distance is concerned. But the guy is physically close to, physically closer to you.

All nodes will keep closer routing table entries and neighbor table entries; that is what happens. And a node will keep on moving to the entries, which are better and better. And neighbor table makes it happen much faster. The routing table actually will very quickly; they will try to find out. For example, you are a new node; you are only building up your neighbor table. So, you do not know in your routing table, which is the closest one. He says maybe you got a far off node and is giving his proximity-based node possibilities to you.

So, you do not know; you are not getting closer nodes in your routing table. But when you get it, even you will ask your neighbor. Neighbor will give his routing table, which is based on proximity. This is already optimized. When you provide it to somebody closer to your neighbor, he must also be closer to you than somebody who is arbitrary. So, this leads to a much faster convergence because of that. So, at this point, let me close this lecture. And in the next lecture, we will look at an example for this ST routing.