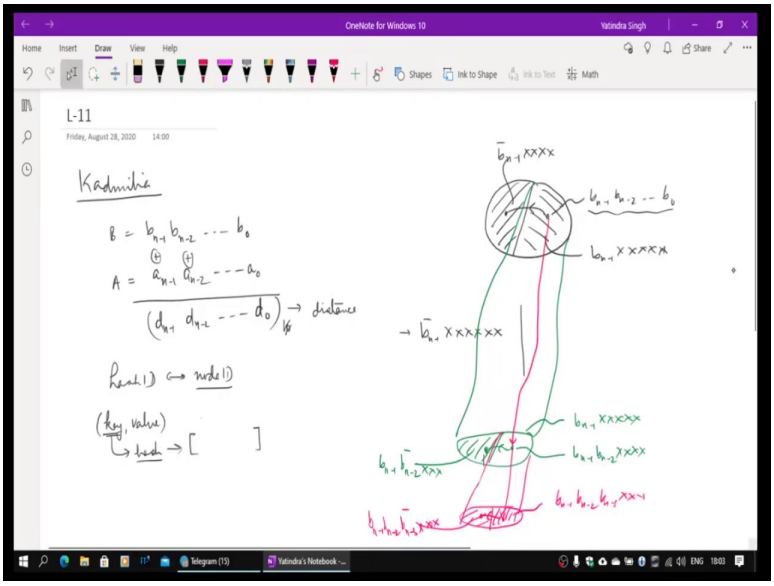
Peer to Peer Networks
Professor Y. N. Singh
Department of Electrical Engineering
Indian Institute of Technology, Kanpur
Lecture No. 11
Tapestry: An Evolution of Kademlia

(Refer Slide Time: 0:16)



This is lecture 11 in the peer to peer networks MOOC. In the previous lecture we have talked about Kademlia, today we will be evolving that Kademlia into a another algorithm called Tapestry. So, Kademlia was algorithm where the nodes were actually being given, node ID and for example a node b will be identified by a node ID which will be represented in binary form as where b_0, b_1,…,to b_{n-1} are the bits they will have only values 1 and 0.

So, any two nodes, I can find out the distance between them through an XOR method so we will do a bit by bit X oring and whatever is the resultant which will come we will find out the numerical value. So, for a node a for example, if this is the bit pattern for the node ID.

So, in this case if I want to find out the distance between a and b, I will now do the X oring, so b_{n-1} and a_{n-1} will be XORed and I will get a d_{n-1}. Similarly, b_{n-2} and a_{n-2} will be XORed, I get d_{n-2} and so on. So, whatever is the numerical value of this base 10 that will be the distance actually.

So, idea here is that any hash ID, hash ID is what you generate for searching for a key. So, normally there is a key and value pair this is what stored in a distributed hash table. So, for key you will be now doing the hashing and this hash will generate the same thing, the same number

of bits cells which are used in node id, so they have to be somehow mapped to the node ID but we are considering cases where they are exactly same and then whichever node is and this hash id hash ID, node ID.

So, we will now search for a node whose distance is least from the hash ID and that node ID will be the root so that should store this key value pair. So, that is fundamentally is the principle. Now, let us actually before I move on to Tapestry let me explain another way of understanding it that how we are actually recording, how we are creating the routing table entries.

So, if this is the node ID, the whole node ID space I am actually by forgetting this node ID space so maybe your node for example node B which is going to be there is here, so this is going to

b_{n-1}, b_{n-2},…, and so on till it b_0.

So, what will happen is we will actually find out this half which is now going to be represented by b_{n-1}xxxx. So this particular half is or the b_{n-1}xxxx, so the current node is one of the members of this whole set, other part is this.

So, essentially if I want to actually find a hash ID which is going to start b_{n-1} bar so if it is 1 this has to be 0, so hash ID is starting with this certainly nobody here in this set will be closer. Somebody in this set has to be closer or will be the root nodes. So, we will hand over. Only one node from here will be now kept in the routing table, so that is why the first entry will always stand out to be b_{n-1} bar and this can be anything.

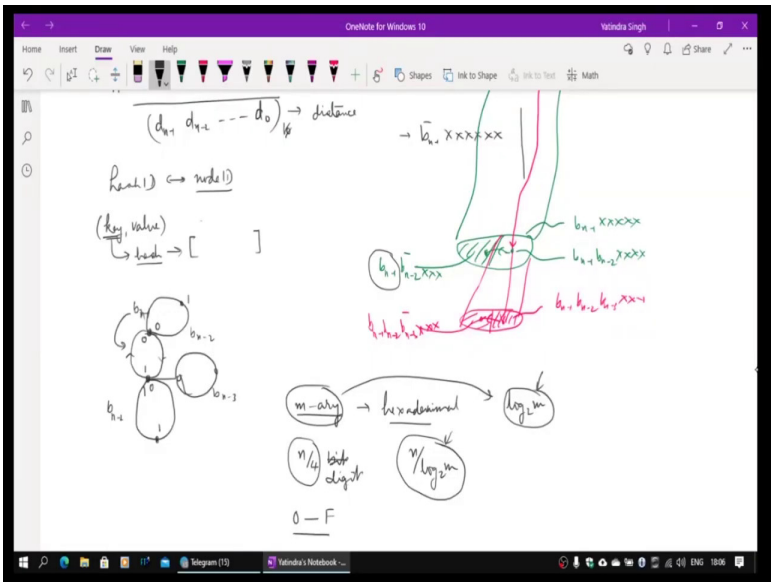
So, any one of these can be picked up and of course if we use proximity routing we will choose out of these whichever one is going to be having the lowest value of proximity matrix so maybe round trip time delay.

So, this will be the first entry. Now, once this has been done so I can take this particular part and then of course expand it further and I get a new space now this space is actually bn minus 1 and so on I can further partition it, now one of these nodes will be $b_{n-1}b_{n-2}$ xxxx something and other partition will now represent $b_{n-1}b_{n-2}$ xxxx.

So, any one node here has to be fitted in this. Now, every time I am doing a binary bifurcation so you can actually keep on doing this exercise again and again so I can again take this and then further do a split so one side will now be $b_{n-1}b_{n-2}$ b_{n-3} xxxx..

And about this particular node which was here which actually came at this point and the same node will be coming here somewhere here, the other side will be now $b_{n-1}b_{n-2}$ b_{n-3} xxxx. So, we can actually now keep on doing routing, every time we will keep one node from this partition and these will be the first entry in the first row, this will be the second entry, third entry and so on. That is how the Kademlia actually works.

(Refer Slide Time: 6:03)



Now, in this case this is a binary system. So, what we are actually having is in the first go, I am actually now searching only for first bit, so first bit is being organized in a ring technically so this b_{n-1} so this is either going to be 0 or 1 so if it is hash ID is matching with this I will keep it here. You will be one of those. You will be one of the places here, there will be only one other guy, so if you do not match it to u you just send it over to other person, that is why XORing will work because there are only two persons in the ring. So, if it matches with this guy you hand it over to this person, this person will now actually have another ring which is going to be with the second digit, so this is a b_{n-1} ring, this will be b_{n-2} ring actually.

((Refer Slide Time))(06:39)

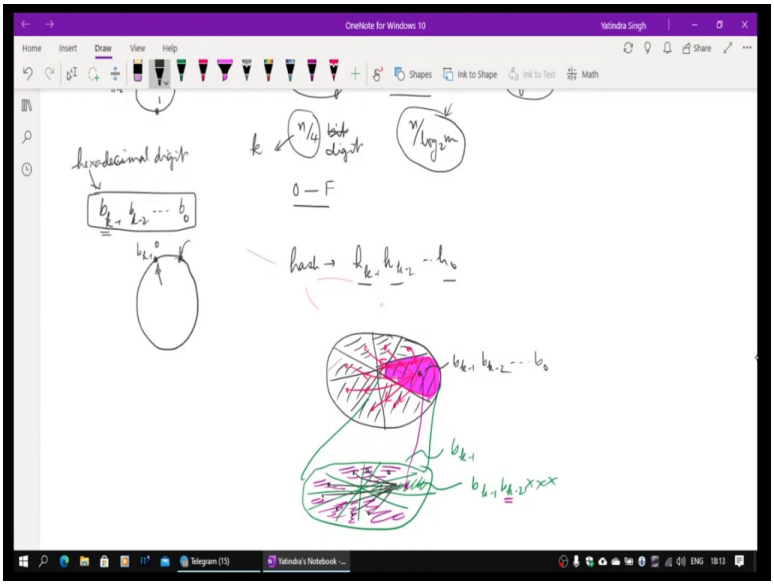
There will be another b_{n-2} ring also sitting here with two values, one value will be 0, other will be 1 so this will be again 0 and 1. So, this should be internally 0 and 1. I should keep only one value here. Then I will again compare with b_{n-2} go to one of them. So, if it comes here then there will be another ring which will be present which will be based on b_{n-3} .

So, it will be basically this b_{n-1} is 1, b_{n-2} is 0 and then the two value, so you kind of create, every time you move to a another ring and you keep on becoming closer and closer to the route node.

Now, this ring instead of this binary digits, I want to use something else, I want to use m-ary digits. So, for example hexadecimal. So, in that case if these are n bits here I will be requiring now n by 4 (n/4) bits, in general for m-ary digits, I will be requiring $\log_2 m$ or I should actually call it $\log_2 m$ bits will be required, you will be requiring the total bits, so n divided by $\log_2 m$ ($n/\log_2 m$) symbols will be or digits will be required.

So, for hexadecimal system which is 16, which is 16-ary thing. log 2 m will be 4 so that is why you require n by 4, I should not call it bits, I should call it digits, hexadecimal digits, so each digit can take a value from 0 to F, after 9 it will be A B C D E F, for 10 11 12 13 14 and 15.

(Refer Slide Time: 8:46)



So, now once we do this I can do the similar exercise in the ring actually. So, I can now form a ring whatever is your current node, current digit, so maybe your digit I will now represent it by not again b, it will be n by 4, so I can call this number as not by small n, but I can represent it by k, so b of k -1, b of k-2, these are hexadecimal digits now.

So right now I am just representing b by another form, so these are with hexadecimal digits. So, if your value b_{k-1} is here, so first what you can do is, you can find out your hash ID, hash ID will also be of the similar form, so it will now contain h_{k-1}, h_{k-2} and so on h_0, each one is now hexadecimal digit.

You have to now find out, in earlier case it was 0 or 1, so either you are here or there will be other opponent, so now on this 16 digits can be placed here, so I can now put this value is going to be 0. So, essentially this is going to be your own entry but then you will require 15 more people who are going to at the same place have different digit actually. So, we can explain this thing like you have this node ID space because I am using now 16-ary system so I will now partition this into 16 parts.

So, one part is yours where you are actually putting b_{k-1}, b_{k-2} and so on b_0. So, with this b_k same particular hexadecimal digit all node ID's are going to be in this set. But there are 15 other possibilities so I will now have one possibility here, one possibility here, one possibility here, so node ID space has been divided into 16 various parts, so I will pick up one guy from each one of them and I will keep a record of that in my routing table.

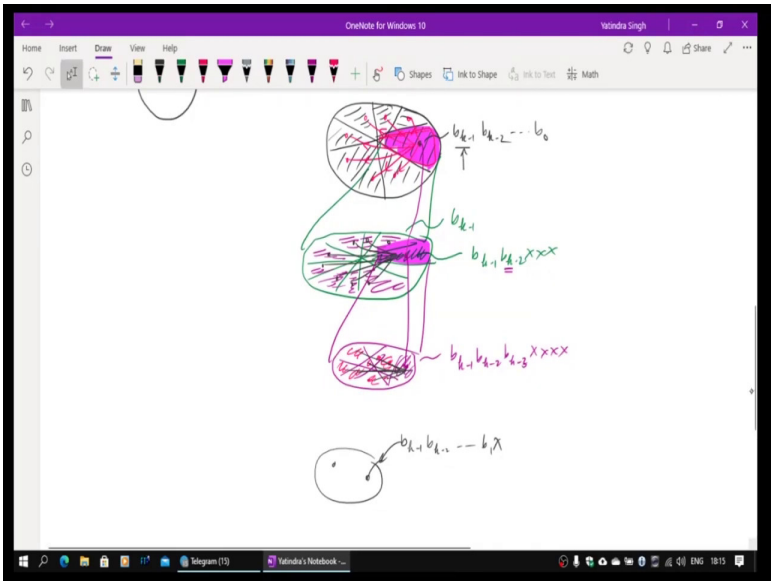
So, this guy will now have a pointer to this, have a pointer to this, have a pointer to this, they have to be 16 I have actually shown less number here, so it is actually 3 plus 3, 6 plus 7 these are octal system currently but it does not matter, for 16-ary system there are have to be total 16 partitions, one in which you will block. So, here you are going to have 7 other entries in the routing table with the hexadecimal system you are going to have 15 other entries.

Now, whatever is your routing table area, whatever is your partition, this partition can be taken, so this partition now is this one and this partition now can be expanded further like this so everybody here will be having b_{k-1} but b_{k-2} can be of different value, so again I will do a partitioning, I may not write do 15 but there actually 16 partitions, one of them will be b_{k-1}, b_{k-2} xxxx all of them.

So, one of the nodes this particular node, so this particular node will now will be coming in, will also be present here in this partition. So, every time my search is being reducing by 1 by 16, so I am doing faster conversions compared to every time earlier in Kademlia we are doing half every time. So, now other partitions can take this b k minus 2, can take various values so I will now choose if it taking a different values. Again 15 other partitions will be present and then you will be having nodes sitting inside them, so you need to probably take only one each so that if ((Refer time))(12:56) close up then internally the way it is maintained in tables this guy also must be maintaining

So, for each partition there will be one node which is picked up and which is going to be put in the table and this will be, this pointer will be maintained for each one of these entries. Now, this we will keep on doing it recursively actually, again till we go to extend where you will only have 16 guys actually sitting in.

(Refer Slide Time: 13:32)



So, this table for example will be only this much part will be there so again this can be expanded, this can be expanded further and I will have again create 16 partitions. And now this one is now representing b_{k-1}, b_{k-2}, b_{k-3} and something so let this be that particular thing.

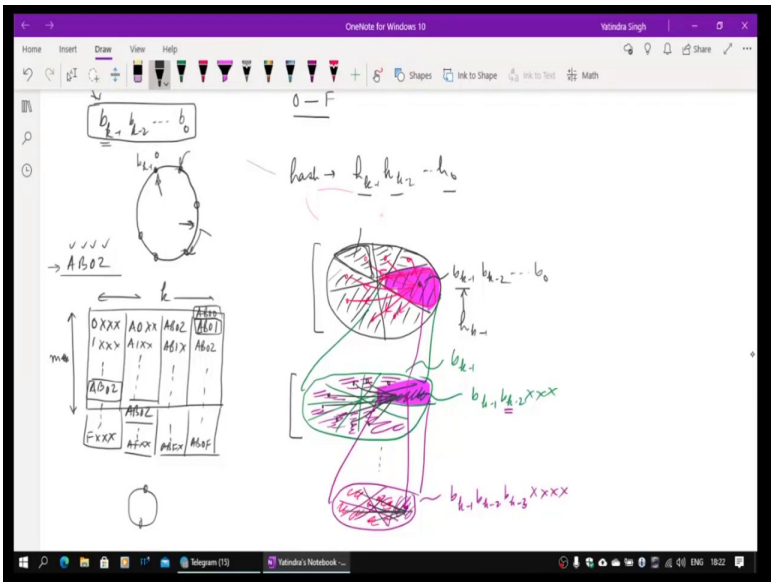
The same node which was present here will also be available here at this point in this partition. And from other again I will choose a different one node from each one of these partitions will picked up and connected in the table that will become a column actually. So, every time when I am creating these 16 entries I am creating a column in routing table, so these 16 things are the rows and when I am actually keeping these entries this will be in first column, this will be second column, this will be in third column and so on.

That is how this keeps on happening in the end ultimately you will have an area where there will be only 16 possibilities entries which can be happening so this will be b_{k-1}, b_{k-2}, and so on, till b_0, so b_0 sitting somewhere here, so actually this node is going to be everything, but if you put this thing as x there will be only 16 possibilities which can exist.

So, it will be b_1 x, there are 16 nodes, some of them will be there, some of them will not be there, so correspondingly either entries will be present in the table or not present. So, at every step whenever you are going to start with for hash id you will look with this particular bit, we will arrange this thing in a cycle, we will arrange this thing in a circle. So some nodes will be present, some will not be present. So, remember if there is no entry in this partition, there is no node in this partition then only that particular node will be missing, even if there is one single node that will be there in the routing table.

As you actually move, as your columns move on the right side as I am becoming smaller and smaller then there will be chances that some of the slots will be vacant in the routing table. So, now if I am searching for any hash, so h_{k-1}, so I will search from each one of these based on the first bit, there is some and if my h_k for example is here, h_{k-1} lies here but there is no node sitting, so that thing is blank, so normally in that cycle we will again use that ((Refer time))(16:04) routing table strategy and we will give it to the next guy, so next person. So, all hash ID which is belonging to this partition will be remapped onto the next person. So, now let us look at how the routing table will be created.

(Refer Slide Time: 16:19)



The routing table will look something like this at a node, so I am now going to take every small size, I am now taking this one, I am actually taking hexadecimal four digit thing. So, let the node be AB02 and then I am going to create four columns because there are four digits, so this will be four columns, so k digits, k columns actually. And this will be equal to is m-ary system there will be total has to be m-1 because for this current node I need not k.

But if I keep 1 also m also, then one of the entry will be occupied by this. So, let us see what is going to happen. So, AB02 so the first digit is going to be very fair, so this is actually for this case, first one. So, I will be actually having 0xxx, some entry has to be maintained from there, 1xxx has to be done. At some point A will come, A is from its own partition, I am talking about routing table at this node, so it has, it should keep itself. So, that is a special thing which is going to be there and then this will keep on moving ultimately you will have fxxx from each partition, if there is somebody exist you will keep it otherwise entry will not be there.

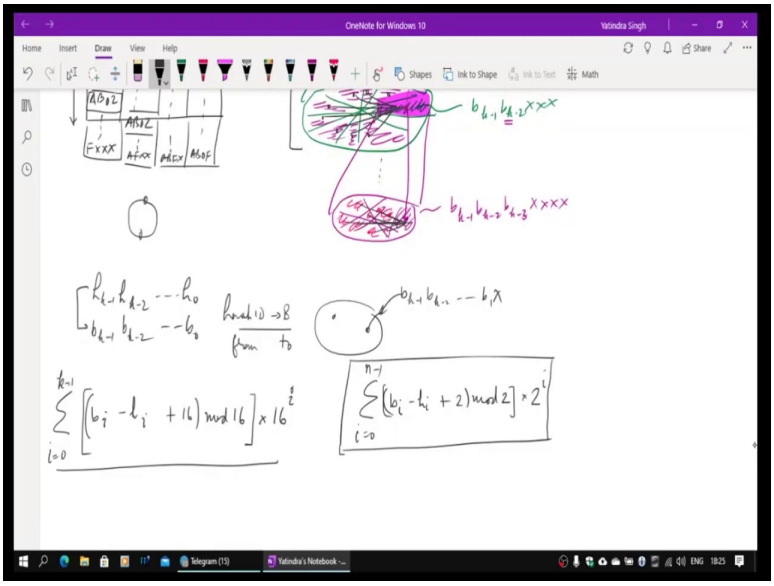
So, now AB02 is now I will now, second column I am now talking about this particular case. So, A has to be as it is, so I will be talking about A0xx, anybody from this partition basically this is sub partitions now. So, then A1xx and so on, I will have AB02 and then it will go Afxx. And similarly now the fourth one AB02 will come in the first case itself, AB1x and so on ABfx will be there.

So, similarly in the last column, now we will be actually again going to look so AB01 has to be there, if this node exist the entry will be there, otherwise it has to be empty. AB02 will be there and AB00 has to be there, sorry this I have not put, so these two will be conditional if they exist they will be there this will be like last one.

Because only 16 nodes are there and then we will have AB0f so whichever nodes exists they will be present. So, again the rule is if I am actually searching for any entry, I will look with the first digit and this will be, first digits will be arranged in the cyclic fashion, if that digit there is no entry for that I will just find out the lowest entry is higher than or equal to the hash ID and will pickup that node and move there.

So, now every time in earlier case in case of Kademlia we were doing binary thing either 0 or 1 and based on that keep on moving further and further and further. Now, here instead of that binary thing I am now doing it with the hexadecimal thing, we can also so the similar thing using octal system also. This what actually Tapestry is, so what will be the distance that will come as a question.

(Refer Slide Time: 19:56)



So, the distance metric if your hash ID is going to be h_{k-1}, h_{k-2} and so on h_0, node ID is going to be b_{k-1}, b_{k-2},…, b_0. So, in this case now remember this distance is now asymmetric actually because I am using circular arrangement of the digits and If hash ID there is no node is there for

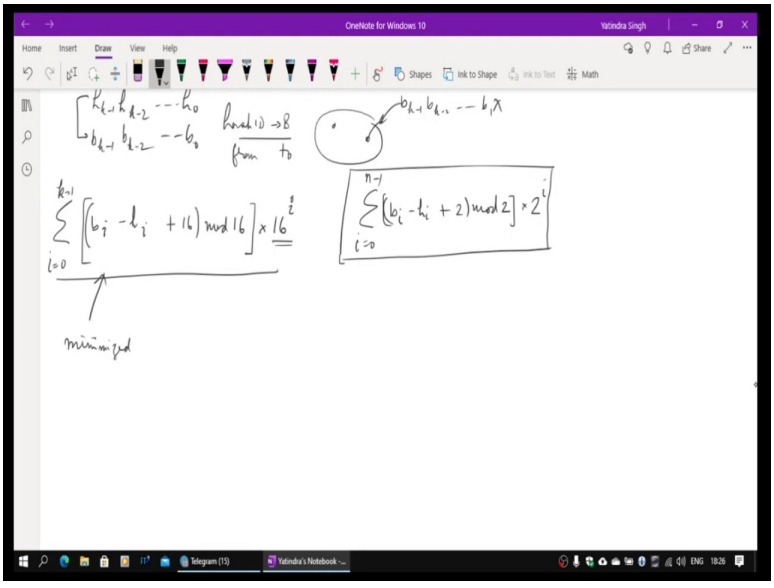
that correspondingly then I am going to the next guy in a clockwise fashion. So, this is what we are actually doing.

Now, in this case the distance from the hash ID to the node ID need to be minimized. So, every time they will do that and then of course I can now write that from hash id to b, the distance is from hash id to b actually, so this is the distance I am talking this is from, and this is to, so it will be b_{k-1} h_{k-1}, now I need to do a modulo operation it will be sixteen, I can take a modulo that will be the distance on that circle.

So, I can now generalize this instead of k. I can now write down I, so I will call (b_i -hi + 16), I can do summation from i is equal to 0 to k-1, so this is now modulo 16 thin., We are doing and then I need to multiply, so I will be multiplying by 16 raised to power 5. Now, see this is very similar distance as I have mentioned earlier. So earlier I have said for Kademlia I will be able to go from distance from h to b_i is going to be b_i -h_i these are bits in this case so that is why modulo 2 operation is good enough.

And I did a summation here and 2 raised to power i, now i goes from 0 to n-1 that was the distance for Kademlia. So, we can this essentially tapestry is nothing but extension of Kademlia in that sense.

(Refer Slide Time: 22:44)



And then we are trying to now minimize this distance, every time. Now, since I am doing 16 raise to power i, once I have minimized on the highest order digit now I do not have to bother

because lower orders actually will not going to make any impact, while in a cyclic arrangement this actually was not true, so that is why in Tapestry you are essentially doing the only match based moment and once the match based moment cannot be made any further, you start doing moment based on the numerical closeness or which we were actually using leave set there.

So, here you do not require a leave set because the distance matrix is this. So, now we need to, we can actually look into an example of how to manage this kind of system and then what more innovations actually can be done. So, the example I will be doing in the next video.