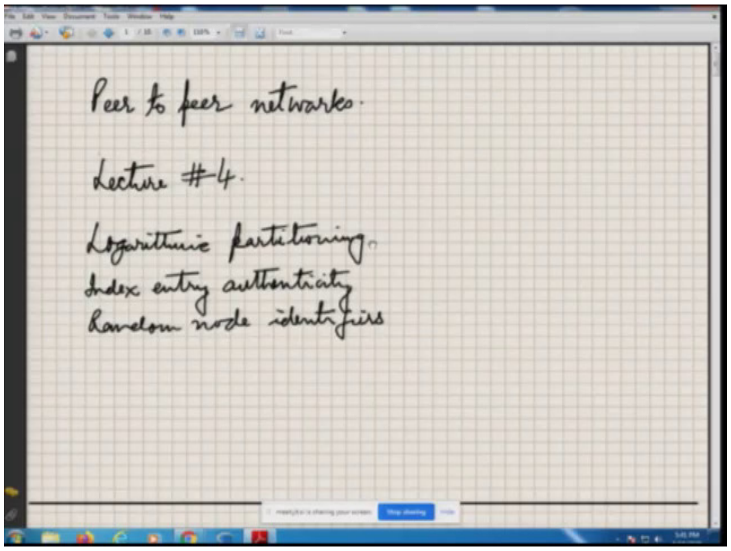
Peer to Peer Networks
Professor Y. N. Singh
Department of Electrical Engineering
Indian Institute of Technology, Kanpur
Lecture-4
Logarithmic Partitioning of Node ID Space and Index Entry Authenticity

So, welcome to the lecture 4 in the MOOC on peer to peer networks.

(Refer Slide Time: 0:18)



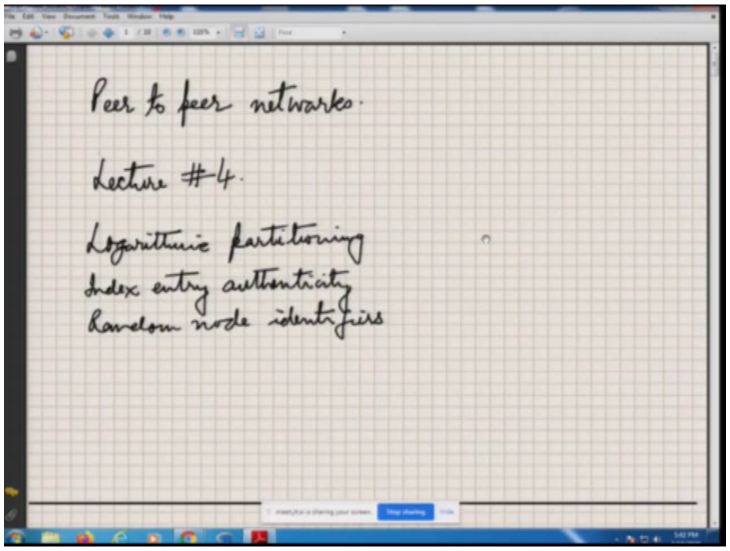
Peer to peer networks.

Lecture #4.

Logarithmic partitioning.
Index entry authenticity
Random node identifiers

So, earlier we had talked about distributed hash tables. We had also talked about what is Chord-based routing, a circular organization based on the node IDs. So, I will be further actually moving ahead and then talk about a few more things. So, today we will be talking about logarithmic partitioning.

(Refer Slide Time: 0:47)



Peer to peer networks.

Lecture #4.

Logarithmic partitioning
Index entry authenticity
Random node identifiers

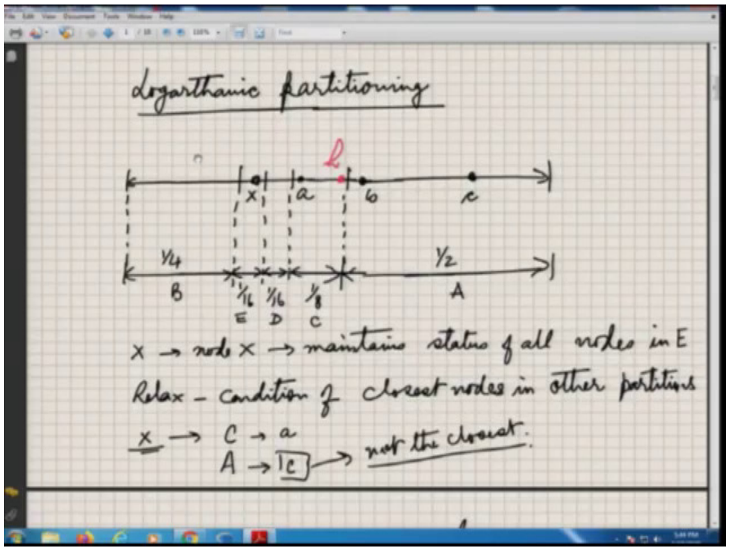
So, I had mentioned this thing earlier. This always ensures that no node is disconnected and network will never get partitioned into two different halves, which they cannot communicate back together again, this will never happen. And then we will be also talking about how the index entry authenticity has to be maintained.

Because it should not happen that if you remember, we have been talking about Voice over IP. So for your phone number, some node ID, anybody should not be able to arbitrarily publish the phone number to the node ID mapping. In fact, we do not use for phone number to endpoint address, we actually use phone number to node ID mapping and in the second step, we go from node ID to endpoint address.

In this index entry authenticity has to be done and then we will be also talking about how the node identifiers can be randomized. How we can force it. Because, there is a denial of service which can happen, people can choose node identifiers on their own arbitrarily. So for your phone number, whoever is the root node, if I just happen to choose the closest node ID to that one, so I will then become the root node, and whenever actually somebody comes to me for your phone number I may give a endpoint, wrong endpoint address I am a rogue node there or I may choose not even to tell see there is this guy does not exist.

I can technically black you out, nobody will be able to communicate to you or I may actually route your call to somebody else which is not correct. This is taken care of in case of SIP-based telephony because SIP servers are proper servers which are authenticated and they are being resolved in both ways through DNS servers. In this case, in peer to peer system, DNS kind of thing does not exist. It is a distributed responsibility handled by DHT table, DHT network. So, let us first come to what is the logarithmic partitioning.

(Refer Slide Time: 2:53)



Logarthamic partitioning

x → node x → maintains status of all nodes in E

Relax – condition of closest nodes in other partitions

$\underline{x} \rightarrow C \rightarrow a$

$A \rightarrow \boxed{c} \rightarrow$ not the closest.

So, as you can see, I have actually represented a namespace by a horizontal line, this is the whole range of the namespace from one endpoint to another point. Now, in this case, let h be the hash value which I want to find out who is going to the root node. So node which is closest to the hash value will be the root node.

So obviously, you can see B should be the root node. So and then I have actually, for example, talking about the node x, x is going to have a routing table or a neighbor table, and it will maintain the status of in every partition 1 node and whatever partition in which it exists all the nodes in that. So, you can see, I can actually divide this whole thing into two partitions.

The first partition is this half from here to this middle and second partition, which I call a, so which is half of the people. But since node x does not lie in this part, so I am not further partitioning it, so b and c are the parts of this.

And then, I look at this particular partition. So, this can now further, we further divide it into half and half so, one fourth here and one fourth here. x lies in this so, I keep this thing as intact. So, one node from this one fourth partition will be there in the routing table at x, one node from this partition a will be present in x. Now, this partition will be further divided into two parts. So, this will become one of them will become C. x is not in that, so I will come to this part and this is further partition to half so E and D.

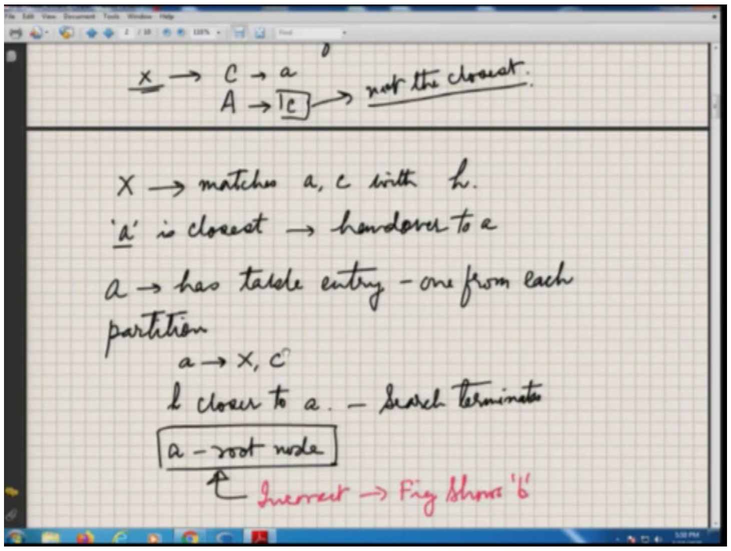
x actually now remains in E there is no more further partitioning being done this is the smallest partition, but this technically depends on what kind of DHT you are using. If you are using tapestry so smallest partition will, can have at most 16 nodes and all 16 nodes. They can be only 16 nodes, so each node will have 15 node entries for that partition. And I will be talking about Tapestry at some point of time later in one of the future lectures and this will become more elaborated at that point of time, but, just to clarify why logarithmic partitioning actually works.

Now node x will maintain the status of all nodes in E that is important because of the lowest partition. So, whatever be the number is the lowest partition is consists of 16 nodes, so all 15 other nodes have to be kept. Whoever was are there with you in that partition, you have to keep track of them, and then you have to keep, now x should also keep one entry from D because this next actually similar order of partition, one entry from c, one entry from b and one entry from a which can be any one of the b or c. For simplicity, I have taken only four nodes present in this DHT network as of now.

And remember, I had when I talked about this, I had put up a condition that x will be taking a node which is closest to itself from every partition. So b should is what should be made the entry normally but we will relax this will not keep this actually. So let us not let us just choose any arbitrary node from every partition to be kept with me, then what happens? So, x will have in partition C, it will maintain a as the entry, this is the only entry available so it can only keep a.

There is no nobody in partition B, so there is no entry for B partition for A partition, b and c are two possibilities. And because I have not kept this I have relaxed this condition of using a closest node, I am going to choose c here. Just I wanted to show why it is important to keep the closest node to yourself from each partition. There is a reason for that, so I am just wanted to explain that reason. So, it chooses c and it is not the closest one, b is the closest one to x from partitioning a. So, I have just made this deviation. Let us see what happens because of this. And remember, I have to now choose figure out this hash value h, h has to be closest to somebody and based on that I have to figure it out.

(Refer Slide Time: 7:13)



$\underline{x} \rightarrow C \rightarrow a$

$A \rightarrow \boxed{c} \rightarrow$ <u>not the closest.</u>

$X \rightarrow$ matches a, c with h.

'<u>a</u>' is closest $\rightarrow$ handover to a

$a \rightarrow$ has table entry - one from each partition

$a \rightarrow x, c$

h closer to a. - Search terminates

$\boxed{\text{a - root node}}$

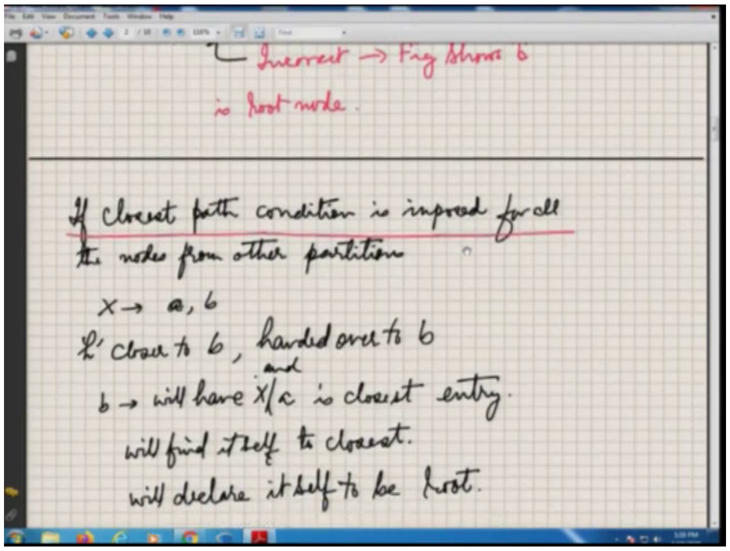
Incorrect $\rightarrow$ Fig shows 'b'

So, now what it will do is x will match a, c with h a as well as c both will be matched with h. So, what will happen is, a will be found out to be the closest entry for h, x has only two entries a and c so it will compare h whether h is closer to a or closer to c, h is certainly closer to a, so it will hand it over to a. And a will also have table entry from each partition. So, a will have again a will do a 1 by 16th partition here. So, there is nobody else in other 1 by 16 partition, a will have its own entry as itself.

One entry which will be there from E plus D partition, this will be one together thing for a actually. So, A will have x and c as the partition. Again, I am relaxing the same condition here. So a is not need not have b here, suppose this can happen, because I am not mentoring this condition I am not forcing it. So, now, how the a will route, a will now search h is closer to whom, to me a, x or c. And it will find out that a itself is the closest to the h.

So, it will declare since I am the closest match, so, I am the root node which is actually not true, this is incorrect. Figure shows that the b should be the root node not a. So, let us see, how this can be corrected.

(Refer Slide Time: 8:40)



Incorrect → Fig shows b is root node.

If closest path condition is imposed for all the nodes from other partition

X → a, b

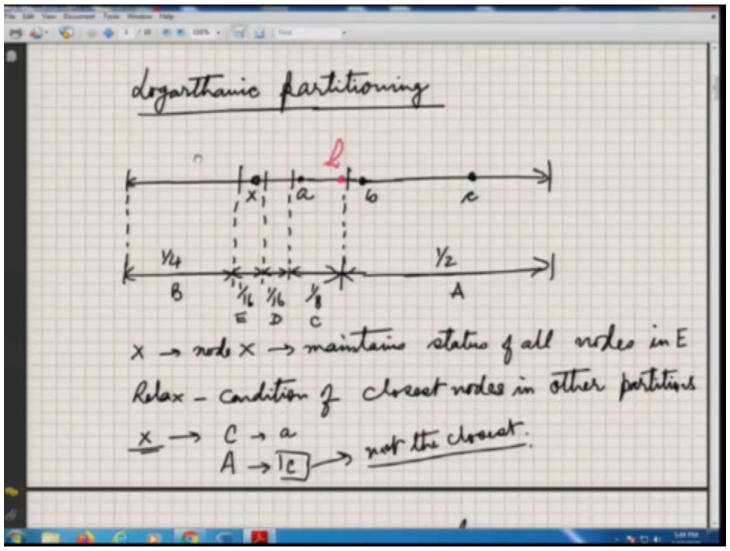
h' closer to b, handed over to b

b → will have X and a is closest entry.

will find itself to closest.

will declare itself to be root.

(Refer Slide Time: 8:52)



Logarthanic partitioning

X → node X → maintains status of all nodes in E

Relax – condition of closest nodes in other partitions

X → C → a

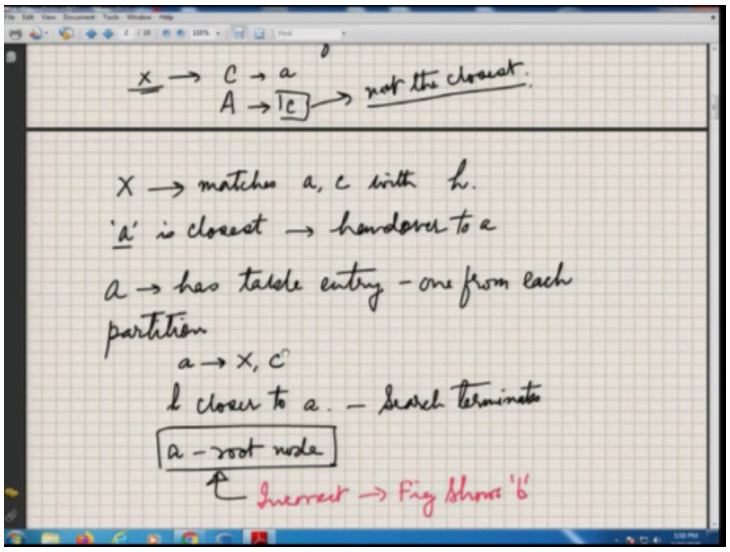
A → $\boxed{c}$ → not the closest.

If I impose the closest path condition, closest path condition is imposed for all node IDs, all the nodes from other partition. In that case Xxis going to keep a and b it cannot keep C, whenever it will come to came up, it will find out about b it will immediately replace c by b actually. So a and b could be the only entries.

This happens because of the neighbor table exchange continuously, you keep on improving, finding out the better nodes and better nodes will replace better entries will replace the older entries actually. So, h will be closer to b and it will be handed over to b. Now b will do the search and we will have x and a as a closer centuries, b actually should have a and c because there will be half partition here, this whole thing will be one single partition. So, I think this is incorrectly written here.

So, b will have a and b, a and c to be the entries. So, it will simply figure out that it is itself is the closest, so it will declare that I am the root, which is correct entry actually now. So that is the reason why you need to choose the closest path condition should be there for every node from other partitions, so whenever you are making a selection. This ensures that things always converge correctly.

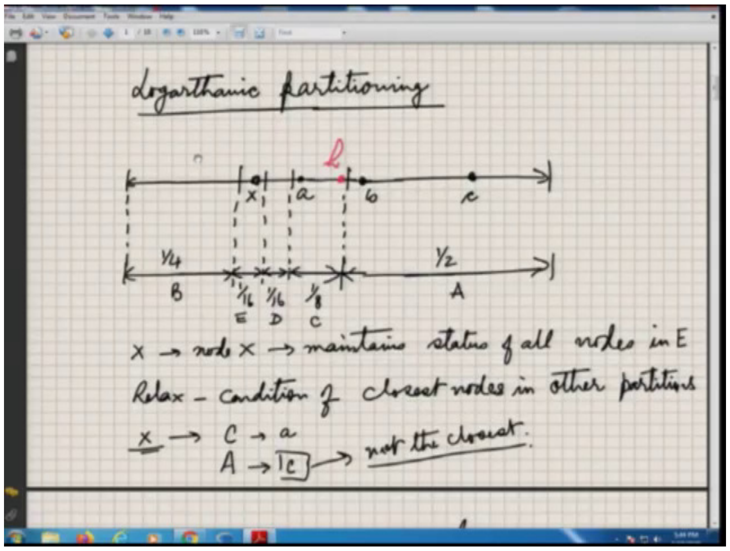
(Refer Slide Time: 10:05)



x → C → a
A → $\boxed{c}$ → not the closest.

X → matches a, c with k.
'a' is closest → handover to a
a → has table entry - one from each partition
a → X, C
k closer to a. - Search terminates
$\boxed{a \text{ - root node}}$
Incorrect → Fig shows 'b'

(Refer Slide Time: 10:10)



Logarithmic partitioning

X → node X → maintains status of all nodes in E
Relax - condition of closest nodes in other partitions
x → C → a
A → $\boxed{c}$ → not the closest.

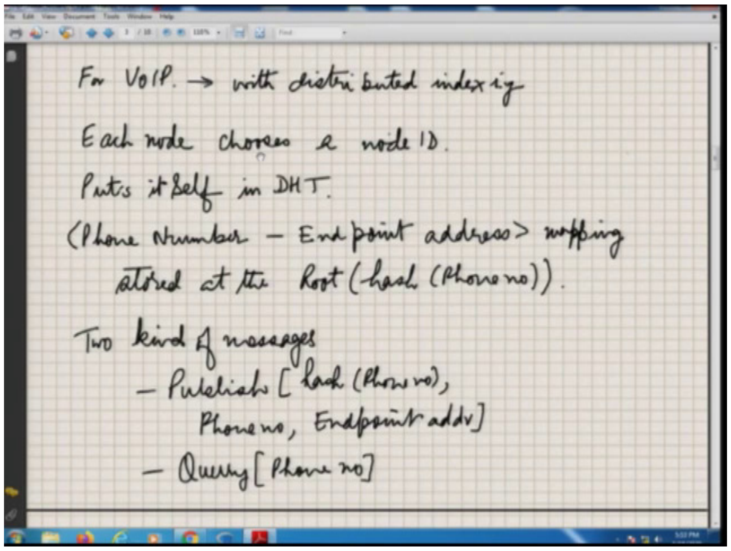
Now, this also means, one of the important thing if now node dies off then what is going to happen. So, if the node dies off then what is going to happen? So, important thing is that x is maintaining all entries here. So, it actually all that neighbor table is being giving it to handing it over to other guys. So he is maintaining somebody who is closer to him, closest in this particular partition, in this half partition and in this half partition.

If x dies off other node entries which are there present here will be available, after some time and a will then again start pointing to that. All your neighbors you are actually aware of so if your dies off your neighborhood information, other people will figure it out and the people who are your closer neighbors will be known to others and people can connect to that. They will never get isolated actually.

Keeping track of only one guy from the remote node is fine, if c fails, so there are other nodes that are actually having knowledge of c. So, c should be connected to somebody here, it is not an arbitrary connection. So, this cannot be arbitrary thing. So, even if c fails some other node which is present here will start communicating. So, network will never actually get partitioned in this scenario.

In fact, this logarithmic partitioning-based routing table is actually a very good option even for unstructured networks. So, currently I am talking about structured ones. So, this is how the network topology or routing tables are being maintained continuously.

(Refer Slide Time: 11:34)



For VoIP. → with distributed indexing

Each node chooses a node ID.

Puts itself in DHT.

(Phone Number – Endpoint address> mapping
stored at the Root(hash (Phone no)).

Two kind of messages
– Publish [hash (Phone no),
Phone no, Endpoint addr]
– Query [Phone no]

So, now let us come to Voice over IP, the original problem with which we started. And with distributed indexing, each node will choose a node ID that is one a very important thing, otherwise you cannot build up a DHT, you have to have a node ID. And you will put yourself in that DHT. And what normally you will be putting is phone number to endpoint addressing. Actually it need not be endpoint it has to be a node ID. I will explain why this node ID is required later on, as we come to the, there is a reason for doing that, it will be, I will be discussing that in next lecture.

But currently let us assume it is an endpoint address. Somehow if you know this, you will be able to make a phone call to that guy directly by after mutual authentication. So, this phone number to endpoint address mapping will be stored at the root of hashed function, hash value of the phone number.

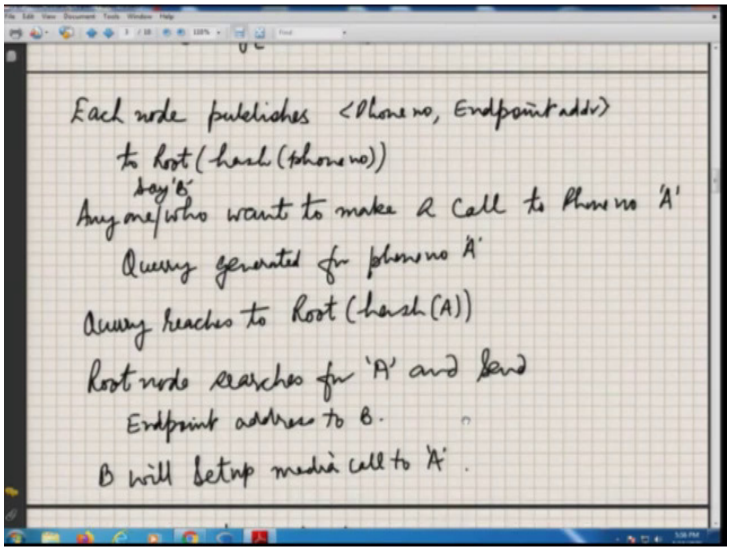
So take a phone number, pass it through a hashing function, if a 256 bit node ID is 256 bit hashing function will be used, you will get 256 bit number the node which is closest to that depending on the distance function. If it is a chord-based system, it is basically a node ID which is higher than this but if it is smallest, higher than or equal to this, that will become the root node or it can be other way around, it can be highest node ID, which is equal to more less than or equal to hash of the phone number.

That also can be depending on how you choose the distance function. So, now the functions once this, this is what has to be put into the DHT by each node after you have tested in the node ID. It will first of thing it will do a publish, so everybody has to tell in there, this should hash table that this is my phone number and is my endpoint address this information has to go. Now anybody who wants to connect to me will do also do a query.

So there are two kinds of messages which will be pushed into the system one is publish. So this publish message will keep on moving till it reaches to the root of the hash of phone number. Once it reaches there, so node will figure out that it is the root node, it will use this and it will keep that entry inside the database.

Somebody can make a query for this phone number. So, we will compute the hash of this and keep on routing until it reaches to the root node, root node will then search for this entry, phone number and endpoint address in the database and send it back to the guy who originated the query. So he will know that what the endpoint address of the destination phone number is.

(Refer Slide Time: 14:06)

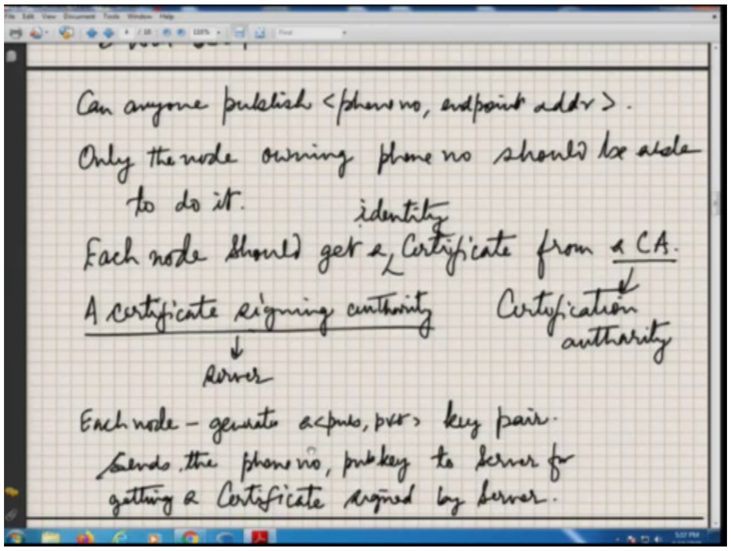


Each node publishes <Phone no, Endpoint addr>
to Root(hash(phone no))
Anyone/who say 'B' want to make a call to Phone no 'A'
Query generated for phone no 'A'
Query reaches to Root(hash(A))
Root node searches for 'A' and send
Endpoint address to B.
B will setup media call to 'A'.

So each node publishes the phone number and endpoint address to the root of hash of phone number. So anyone say B who wants to make a call to phone number A, he will generate, a query will be generated for phone number A actually. So query will be routed, will reach to the root of hash of A, A is the phone number of the destination. The root node will now search for A and send the endpoint address back to B and B will set up then a media call to A essentially.

So if you note down in the SIP server-based telephony, a similar thing was there but you were using SIP server there, you were telling the destination, SIP server was communicating to the SIP server of the destination which was then calling the ringing, response was coming back and then both were knowing each other's IP address and they were talking to each other they were coming, they were setting up a media call. Same thing is going to happen here.

(Refer Slide Time: 15:00)



Can anyone publish <phone no, endpoint addr>.

Only the node owning phone no should be able to do it.

Each node should get a identity Certificate from a CA.

A certificate signing authority $\rightarrow$ server

Certification authority

Each node – generate a<pub, pvt> key pair.

Sends the phone no, pub key to server for getting a Certificate signed by server.

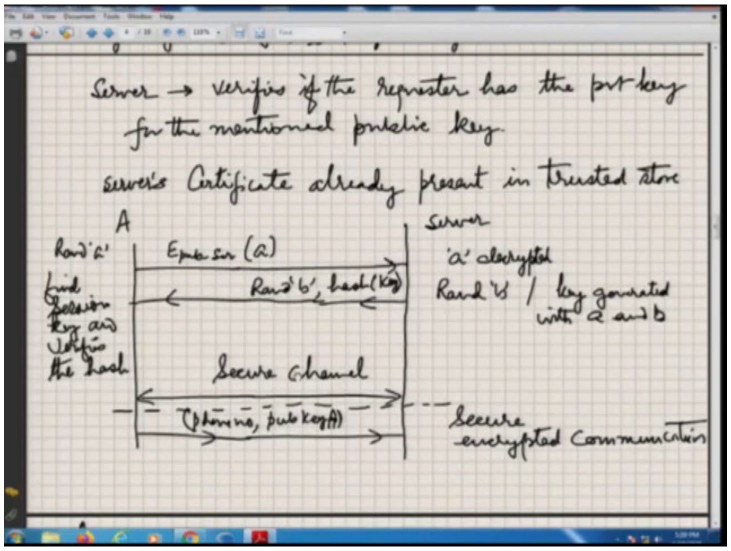
Now question is can anybody publish phone number to endpoint address? I think it should not be permitted. So otherwise, maybe for your phone number, I may put somebody else's endpoint address. So only the owner of the phone should be able to do it. And when I say phone number, it need not be phone number, it can be email id also for making a voice call, it has to be unique identity, and which should be verifiable.

So the way it happens is each node should get an identity certificate from a Certification Authority. We call it CA. We also call it authentication server in our parlance, and this server will actually generate. I mean digitally sign the certificate. Now what the certificate will contain, how actually it happens. Each node has to generate a public and private key pair. It will send its own phone number and public key to the server for getting a certificate sign. A certificate should be created and signed by that server.

Now one thing which is important as I have told earlier, this server certificate is available in trusted store of every node which is participating in the network. So when I use, somebody else offers me a certificate. I can verify the digital signatures of that certificate. So, and if that is verification is viable, is possible, it happens then I know that this certificate was created by this Certification Authority, because nobody else knows the corresponding private key. He's the only guy who knows and his certificate is available in my trusted store.

So, normally we actually when we distribute the application that time itself we embed the public certificate of the Certification Authority. So, this is done in almost all peer to peer systems.

(Refer Slide Time: 16:49)


Server → verifies if the requester has the pvt key for the mentioned public key.

server's Certificate already present in trusted store

A
Server
Rand 'a'
E pub ser (a)
'a' decrypted
Rand 'b', hash(Key)
Rand 'b' / key generated with a and b
find session key and verifies the hash
Secure channel
(phone no, pub key A)
Secure encrypted Communication

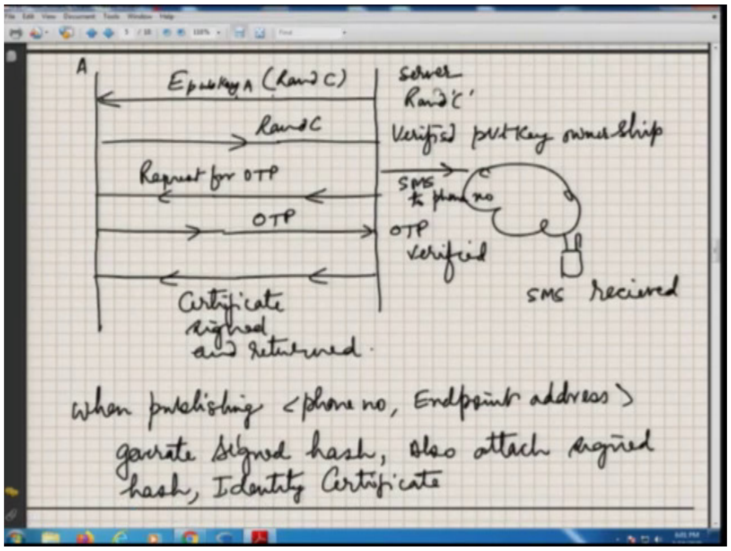
Now server needs to verify if the requester who has given me his public key and as well as a phone number and is asking you to sign a certificate, I should figure out that whether he holds a corresponding private key or not, that is very important. So, one possibility is he actually gives me a signed request to generate a certificate, so I can verify to the public key then I know he is the owner, it cannot be anything arbitrary.

Otherwise I can generate get a certificate generated for somebody else by giving something, so that certificate goes in that original guy does not have the private key I can create a denial of service attack otherwise. So in this case, we have to understand the server certificate is already present in the trusted store of each user. Now, this is a process which will be followed that no day who wants to publish his entry phone number two endpoint address mapping will generate a random number A, it will encrypt with the public key of the certificate server or security authentication server, sign that thing so nobody can figure it out only the server can decrypt it because he has the private key.

A will decrypted it, it will generate a random number B and it will generate using A and B a session key, security session key. It will then send the random B it need not be encrypted and a hash of this key which is generated will also be sent.

Once it comes back here, A and B both will be combined by A to generate the key and hash will be computed to compute or verify the hash value. If the tempering happens, this hash will not match. So, that is how the two random numbers are generated by two different people and every time it is going to be a different session key, so replay attack cannot happen in this case. And this session keys there, you have a secure channel now. On this the A node A can send its phone number and public key for encrypted communication. So on this for generating a certificate. Still server has not verified whether A has the corresponding private key or not, so it has to do that.

(Refer Slide Time: 19:02)



when publishing <phone no, Endpoint address>
generate signed hash, also attach signed hash, Identity Certificate

So what it will do is it will actually send server will generate a random number C, it will encrypt the public key of A, it goes back, remember it is a secure channel now. So rand C can be returned back, if the rand C is same, so they A actually has the public key, has the corresponding private key. So first verification is done, then the server will send an SMS to the phone number, it can be an email also email-based OTP also, if the email is the unique identification being used.

So SMS will be received on the phone and then it will also then send a request for submission of OTPs. He will enter that OTP here, OTP comes back; OTP will be matched and verified. So guy owns the phone number. So I can actually allow the client to act on behalf of that number, actually, that is how it is done or it can be email ID. So then OTP will go via email in that case. So now the certificate will be signed, a certificate will be created having phone-number, having a public key and then that will be signed by the private key of the server.

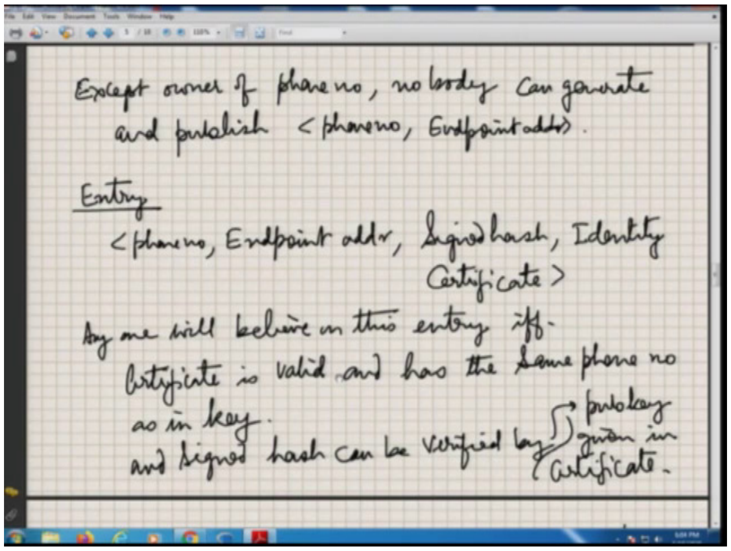
Anybody can verify this certificate because each every client is supposed to have the public, has the certificate of server, and hence the public key of the server actually. So, this is Certification Authority. So when publishing a phone number endpoint address, a node will also generates the signed hash using its private key, which is only known to itself. It will also attach its identity certificate signed by the certificate authority.

I can always verify the signed hash using the public key of client A, I can verify the authenticity of client A's public key by using the certificate which is signed by Certification Authority. So even if somebody actually tries to put a kind of rogue entry phone number endpoint address, but he cannot actually forge the digital signed hash, digital signatures, he cannot actually forge upon the certificate. So, people will be able to figure it out.

So, it does not make sense to actually corrupt the entries, only the original guy will be able to put his entry there. So, only thing root node has to ensure or anybody who makes a query should ensure that whatever his phone number given here, the certificate should have, that identity certificate should have same phone number, it should be verifiable, using public key I should be able to verify the signed hash.

If all the three things happen, the entry is valid and I can use that for making a phone call. And except the owner of phone number nobody can generate and publish this phone number to endpoint address now. So I have ensured that no rogue node can exist and they cannot tamper with the database, distributed database.

(Refer Slide Time: 21:39)



Except owner of phone no, nobody can generate and publish <phone no, Endpoint addr>.

Entry

<phone no, Endpoint addr, Signed hash, Identity Certificate>

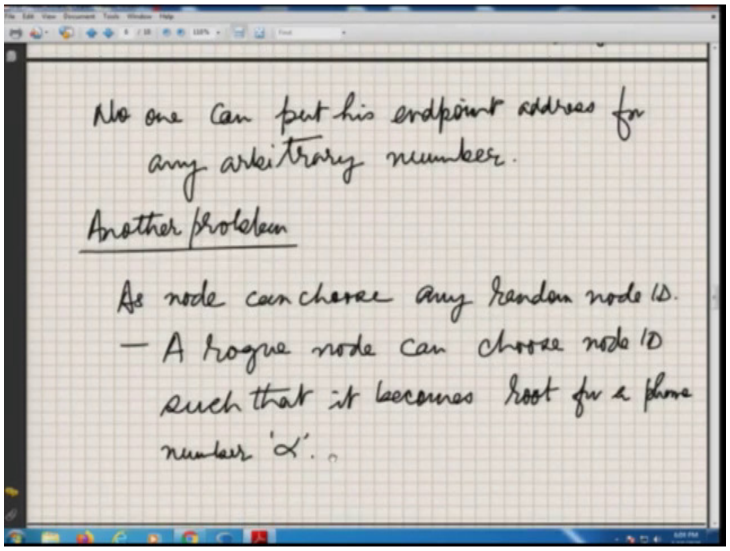
Any one will believe on this entry iff.

Certificate is valid and has the same phone no as in key.

and Signed hash can be verified by pub key given in Certificate.

And so normally the entry will look something like this a phone number and point address, signed hash and identity certificate. All this will go in database not only the phone number and endpoint address, as I have mentioned earlier. And anyone will believe on this entry if and only if certificate is valid, and has the same phone number as in the key. And signed hash can be verified by the public key given in the certificate.

(Refer Slide Time: 22:19)



No one can put his endpoint address for any arbitrary number.

Another problem

As node can choose any random node ID.

— A rogue node can choose node ID such that it becomes root for a phone number 'α'.

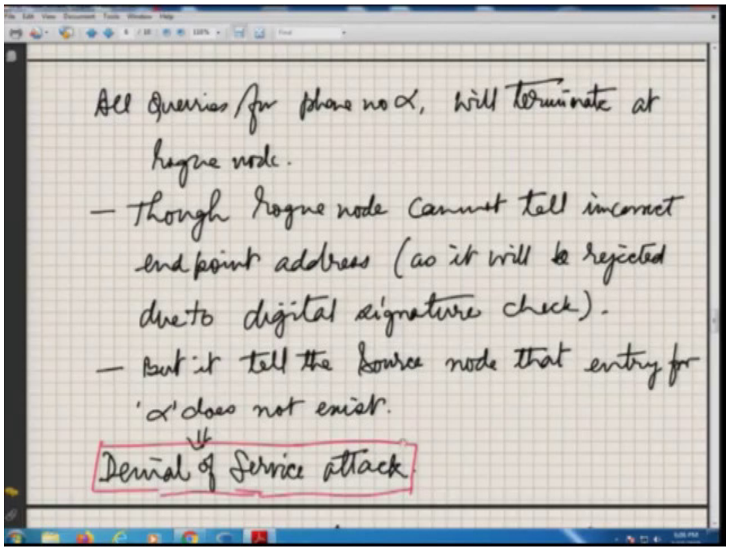
So no one can put this endpoint address for any arbitrary number. So we have taken care of this problem, but we do have another problem actually. This is also one of the key things in all peer to peer systems and all, in fact, whether you go for block chain or you go for N-Talks kind of app, so all those are actually using their public keys as the node ID. So they have corresponding private keys also.

So normally, node IDs are being used to encrypt the information, send it back, so use a private key, then only you can get it. So you cannot arbitrarily choose. You have to generate both private and public key. And which is a random number generation process, so you cannot arbitrarily choose a public key, it is through a random process, so you will be just simply randomly placed.

So you cannot choose yourself to be placed close to a hash of certain key and start doing a denial of service to the people who are trying to connect to that guy. So, that is actually not feasible. So, the problem actually again, let me specify a node can choose any random node ID as of now, and rogue node can then choose the node ID such that it becomes root of a phone number say alpha. I want to block it, say is a some big shot and I want to ensure that nobody should be able to talk to him; I just choose my any random node ID if I can, and it is going to be such that the hash of alpha is closest to that.

So all queries will terminate to me and I will say this guy does not exist. Of course, I cannot temper, I cannot route it to some other endpoint address that is taken care of through digital signatures but I can at least deny that this guy does not exist. And there is no way you can verify. So, this actually can be taken care of in some mechanisms.

(Refer Slide Time: 24:16)



All Queries for phone no α, will terminate at rogue node.

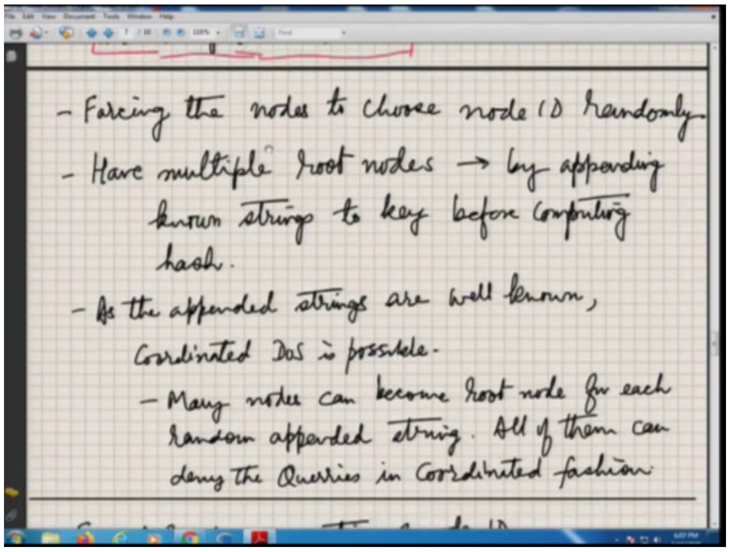
- Though rogue node cannot tell incorrect endpoint address (as it will be rejected due to digital signature check).
- But it tell the source node that entry for 'α' does not exist.

$\Downarrow$

$\boxed{\text{Denial of Service attack}}$

So, this denial of service attack has to be taken care of.

(Refer Slide Time: 24:20)



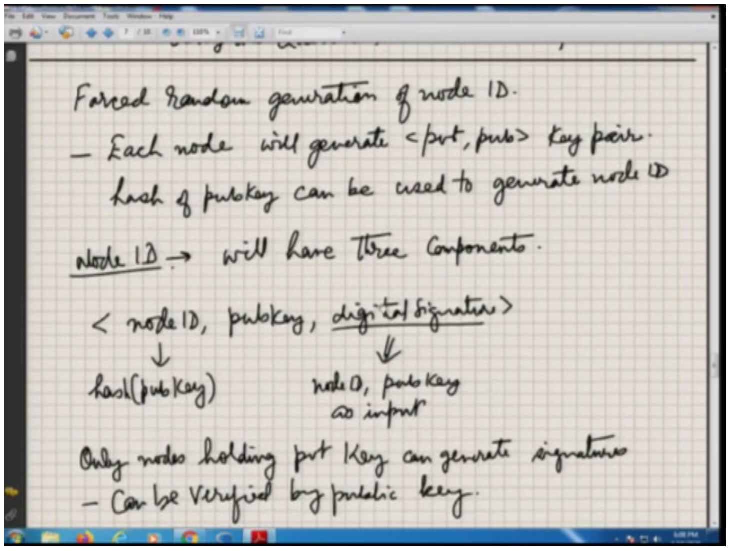
- Forcing the nodes to choose node ID randomly.
- Have multiple root nodes $\rightarrow$ by appending known strings to key before computing hash.
- As the appended strings are well known, Coordinated DoS is possible.
  - Many nodes can become root node for each random appended string. All of them can deny the Queries in Coordinated fashion.

So, the way it can be done is, so either I should force the nodes to choose their node ID randomly, that is what I think is a preferred route or I can also have multiple root nodes. So, you can only place yourself at one root node handles me that you cannot do it at other places.

And this will be done by appending a known string to the entry whose hashing is done. So for example, phone number dash copy one, so dash copy one I have attended compute the hash, you get a different root node.

I say dash copy 2 being appended, I will get a different root node, and both of them can keep the entry. So I can search with dash copy 1, after being appended or dash copy 2 dash copy 3 and so on. So unless rogue, a set of rogue nodes placed itself at all these possible root nodes of copy 1, copy 2, copy 3, which is actually very much viable. So this is not a good solution actually, by appending some string and trying to find out create multiple root nodes because somebody who is attacker can actually create, put itself at closest to those particular root nodes. So then coordinated denial of service attack is going to be feasible.

(Refer Slide Time: 25:29)



Forced random generation of node ID.

– Each node will generate <prt, pub> key pair.

hash of pubkey can be used to generate node ID

<u>Node ID</u> $\rightarrow$ will have Three Components.

< node ID, pubkey, <u>digital signature</u> >

node ID $\downarrow$ hash(pubkey)

digital signature $\downarrow$ node ID, pubkey as input

Only nodes holding prt Key can generate signatures

– Can be verified by public key.

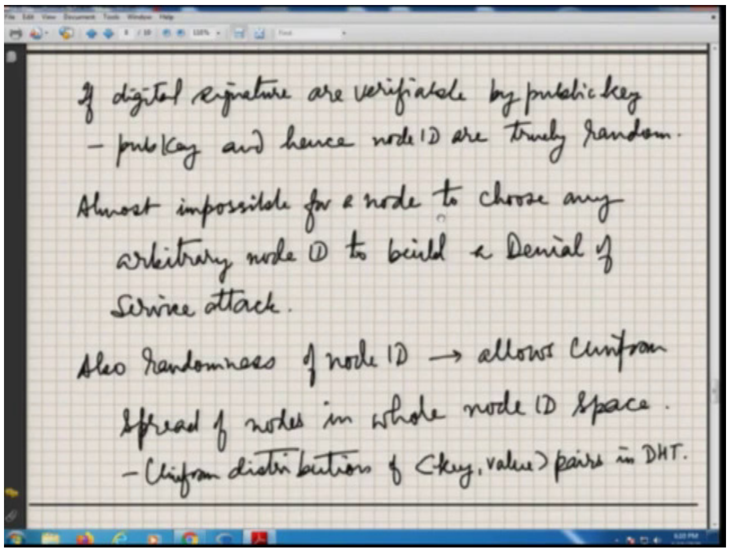
Only way is now to force random generation of node IDs. So each node needs to generate a private and public key pair. Now this is a random process, you cannot choose your private public key because they both actually come together. If you know public key and then you want to choose a private key that is not feasible. If you know private key, you want to find out public key not feasible, both comes in together in the process. That is why it is random.

And hash of this public key can be used to generate node ID if the hash value is larger. So for example, you are using 2048 bit public key, 2048 bit private key, so then, but your node ID space is 256 bit so you have to do further hashing of the public key to 256 bits and that will be your node ID. So node ID and it now will not have simply a node ID or a random listing, it will have three components, it will have node ID , which is basically hash of the public key a public's also has to be present. And I can sign these two things together as a block using the corresponding private key.

Now, the question is I can always use this public key and verify the signatures are correct or not correct. And it means this node it can only be generated by a guy who has the corresponding private key to this public key actually, nobody else can generate this. And we also ensure because these digital signatures are there, and they can be verified, it is generated through this process, public private key pair generation.

So it is the randomly generated stuff you can, it is not arbitrarily placed. So, once you force it, so you yourself cannot, you cannot place yourself anywhere arbitrarily in the network. So, this actually means you cannot actually now do denials of service attacks, so that probability reduces drastically.

(Refer Slide Time: 27:06)



If digital signature are verifiable by public key
- pubKey and hence node ID are truly random.

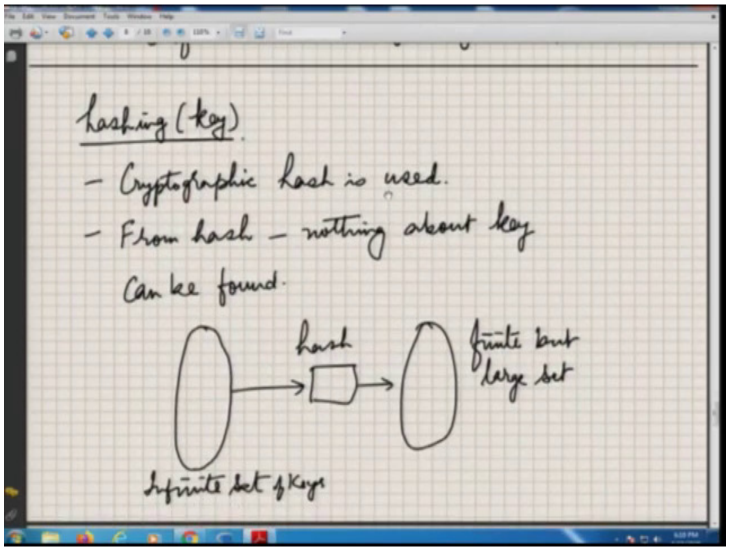
Almost impossible for a node to choose any arbitrary node ID to build a Denial of Service attack.

Also randomness of node ID → allows uniform spread of nodes in whole node ID space.
- Uniform distribution of <key, value> pairs in DHT.

And of course, if digital signatures are going to be verifiable by the public key, so public key and hence the node ID are truly random, that can be insured. So that is a key thing. And almost impossible for a node to choose any arbitrary node ID to build a denial of service attack, so we solve this problem.

Also the important process that randomness of node ID allows a uniform spread of nodes in the whole node ID space. This actually means uniform distribution of key-value pairs in the DHT otherwise there will be unbalancing which can happen, so it will be more balanced system. So that is another added advantage which will come because of the randomness.

(Refer Slide Time: 27:47)



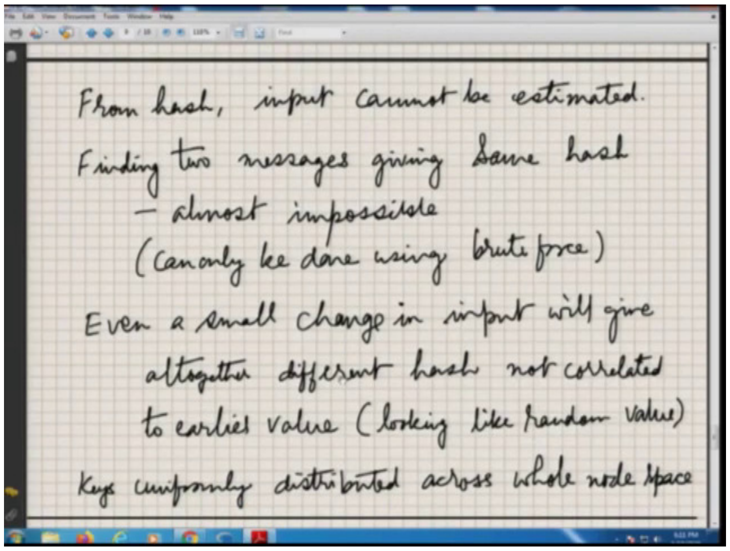
hashing (key)

- Cryptographic hash is used.
- From hash – nothing about key can be found.

And for hashing of the key, we do not use the hashing which we had done earlier. In our previous lectures I have talked about hashing, but was kind of doing modular operation or something, no that is not done now. In most of the practical cases, we use something called cryptographic hash. From the hash value, you will never be able to figure out what is going to be the value of key. So, from hash I will not be able to figure out what was the key whose hash was computed.

So, normally in finite set of keys passing through hash function I will get finite, but large set. So, 2 raised to the power 256 is extremely large number. Normally, there will be many entries which we may up to same entry, but it is very difficult to figure out from this entry the hash value find out what was the original entry whose hash was computed. This is a cryptographic thing.

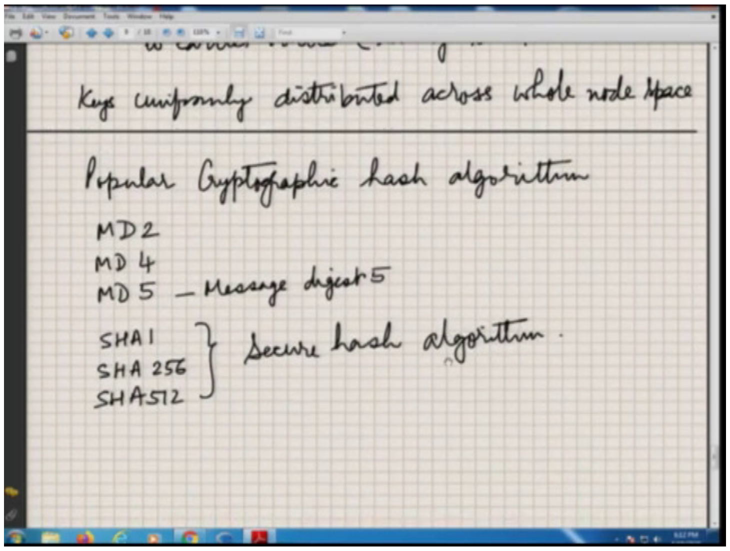
(Refer Slide Time: 28:46)



From hash, input cannot be estimated.
Finding two messages giving same hash
— almost impossible
(Can only be done using brute force)
Even a small change in input will give
altogether different hash not correlated
to earlier value (looking like random value)
Keys uniformly distributed across whole node space

So, from hash your input cannot be estimated, and finding out two messages which gives the same hash value is also almost impossible. This can only be done through brute force, but it will take too much of compute time to do this job and even a small change in the input will change altogether a different hash value. You get, it is not correlated with the value which has been computed before the hash is being estimated. So, and keys will now be uniformly distributed across the whole node space. So that is the advantage of this.

(Refer Slide Time: 29:24)



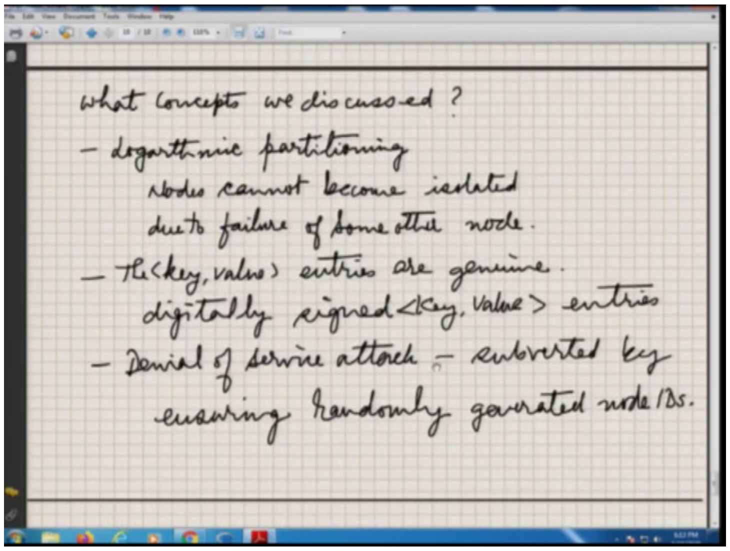
Keys uniformly distributed across whole node space

Popular Cryptographic hash algorithm
MD2
MD 4
MD 5 — Message digest 5
SHA1
SHA 256
SHA512
} Secure hash algorithm.

These are the popular cryptographic hash algorithms we have been using SHA 256 in our case now, in our Brihaspati-4 implementation which we have been doing. So MD2, MD4, MD 5 that is a message digest algorithm and Secure Hash Algorithm is SHA, SHA1, 256 and

512. SHA 1 can be easily cracked by brute force technique it has been shown. So you can find out 2 messages, which will lead to same hash value, so SHA 1 is actually normally not used. A SHA256 or SHA512, in fact SHA512 is more preferable.

(Refer Slide Time: 30:00)

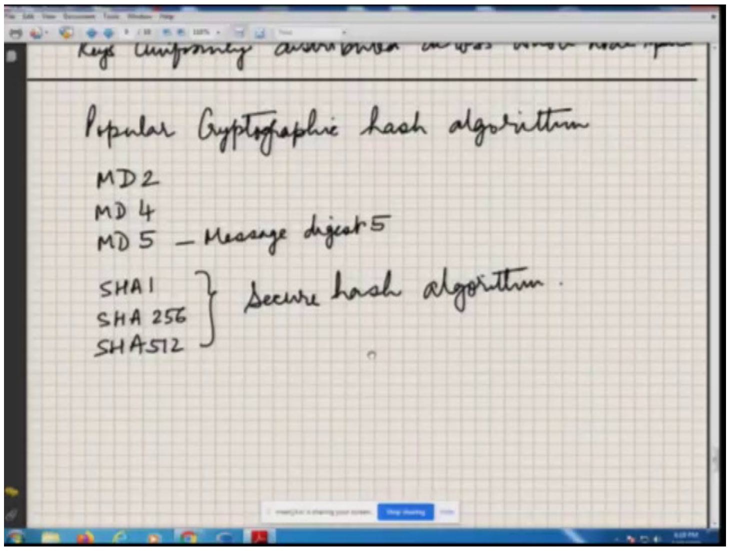


what concepts we discussed?
- logarithmic partitioning
  Nodes cannot become isolated
  due to failure of some other node.
- The <key, value> entries are genuine.
  digitally signed <key, Value> entries
- Denial of service attack – subverted key
  ensuring randomly generated node IDs.

In today's discussion, what we had discussed is logarithmic partitioning. Nodes cannot be isolated due to failures of some other node, because every node knows about all the neighbors, so my neighbors also know about me. So, even if I die, they will still communicate to others because and other partitions, they are connected to the also the closest node I am also connected to the, those guys. So, they will inform about me and update will happen automatically.

Somebody else who is closest in my partition to others will actually, so, that is ensured in this case. And we also discussed how the key value entry pairs are genuine, they are digitally signed entries now, and how to ensure that we do or deny, we take care of denial of service attack, which is done by ensuring the random generation of node IDs. We will now look into the next lecture about how the actually Voice over IP is going to function (())(30:57).

(Refer Slide Time: 31:01)



Popular Cryptographic hash algorithm

MD2

MD 4

MD 5 – Message digest 5

SHA1 }

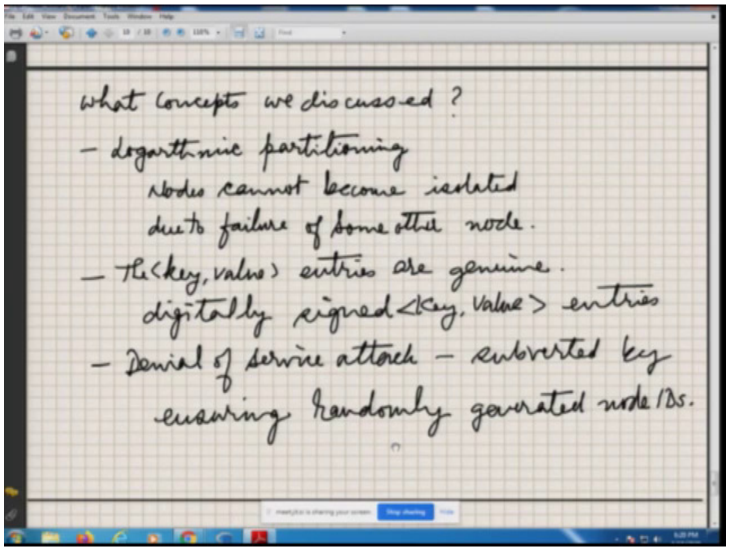
SHA 256 } Secure hash algorithm.

SHA512 }

So, some of the popular cryptographic hash algorithms are MD2, MD4, MD5, SHA1, SHA stands for Secure Hash Algorithm, MD stands for Message Digest and 2, 4, 5 are the versions stop then SHA256 means 256 bit hash value which will be coming out, SHA512 means 512 bit hash value will come out and these are all cryptographic hashes.

So, even minor change in the input will make a lot of drastic change in the output which is computed. And normally a larger value SHA512 or even higher SHA1024 kind of algorithm actually should be used, not the smaller ones because, now with the kind of computing power people have they can be easily, you can by brute force you can figure out an input which will generate the same hash value. So, people actually have moved to SHA512 or even higher now.

(Refer Slide Time: 31:57)



what concepts we discussed ?

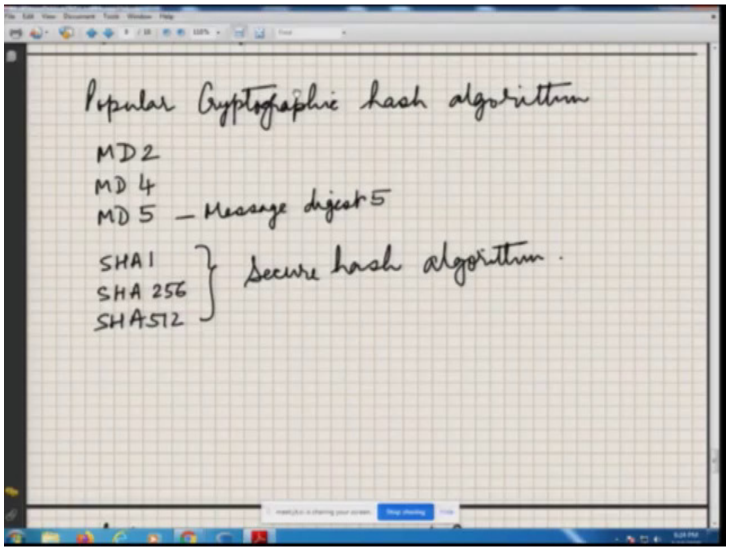
- logarithmic partitioning
  Nodes cannot become isolated
  due to failure of some other node.
- The <key, value> entries are genuine.
  digitally signed <Key, Value> entries
- Denial of service attack – subverted key
  ensuring randomly generated node IDs.

So, in the lecture for what briefly we have discussed, all the concepts one is the logarithmic partitioning. So, this ensures that node will never become isolated, each node will actually have the information about all the neighbors, which are there in this the smaller smallest partition for which it is, to which it is belonging.

And it only knows one node from other partitions every time. So even if you fail, other guys in your partition knows about you, they also knows about the same node in the other partition which is closest to them so, they can update them that you are no more there and the other guy can update based on whatever information these guys will be sending it to them.

So, you will never get partitioned even if a node dies off because of the structure of this network. Second thing which we understood today is that the key value entries have to be genuine and for that they have to be digitally signed. So, that is what we keep. So, popular cryptographic hash algorithms, there are many of them. So, some of them I have written here.

(Refer Slide Time: 32:58)



Popular Cryptographic hash algorithm

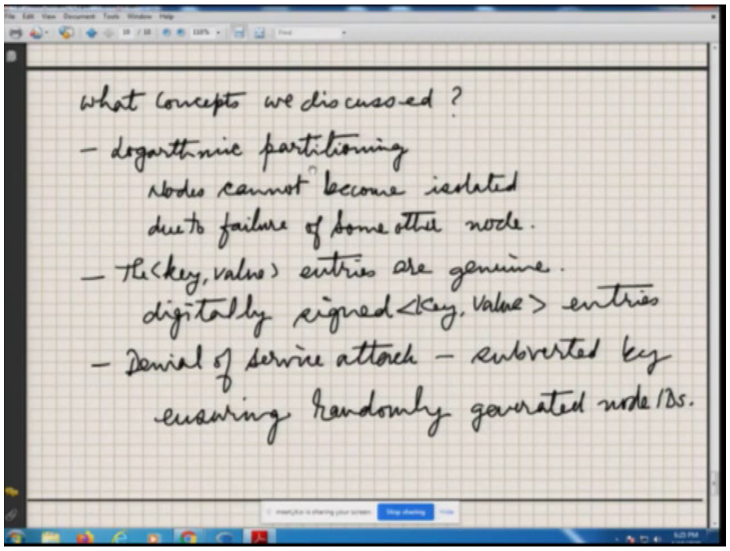
MD2
MD4
MD5 – Message digest 5

SHA1
SHA 256
SHA512
} Secure hash algorithm.

These are MD2, MD4, MD5, MD stands for Message Digest, MD5 had been very popular, I had been using that earlier. And but now most of the places we have upgraded to Secure Hash Algorithm SHA version. So we have SHA1, SHA256, SHA512. SHA1 is not very popular now, not many people use it, so most of the time people are actually moving to the algorithms which give larger bit sequence. So, even SHA1024 algorithm is going to is now more popular.

So, we use either SHA512 or SHA1024 for security reasons, these are not, this is not easy to identify an input instance, which will give a corresponding hash value for this but for the lower values, it becomes far easier. It all basically because you have too much of compute capacity, too much of time and it is very easy, easy actually one can do it now. So, with that we close the lecture, but summary quick summary of what we had done today.

(Refer Slide Time: 34:07)



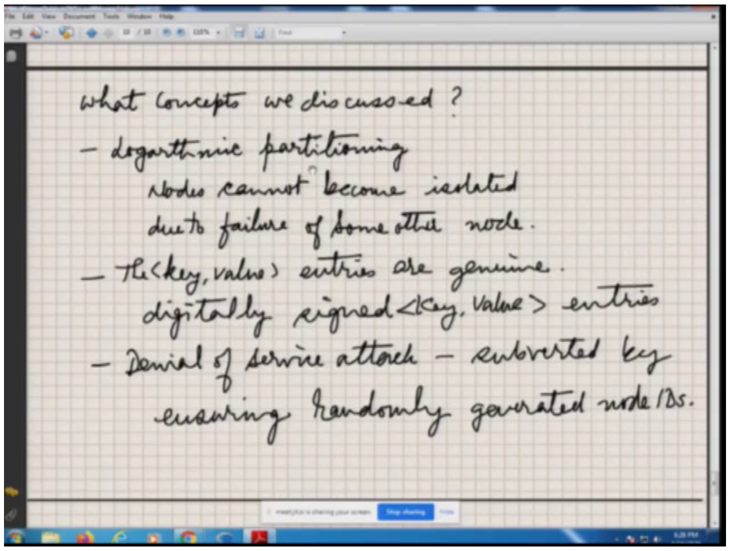
what Concepts we discussed ?

– logarithmic partitioning
  Nodes cannot become isolated
  due to failure of some other node.
– The <key, value> entries are genuine.
  digitally signed <key, value> entries
– Denial of service attack – subverted key
  ensuring randomly generated node IDs.

We, I had talked about logarithmic partitioning. So, every node is knowing all the neighbors which are very close to it actually all the neighbors in the local smallest partition is already known to them. And it actually keeps track of the closest node in the other partitions, which is being also done by all your closest neighbors. So, even if you die off, so, your closest neighbors are also connected to the same nodes in the other partitions.

They will update that you are no more there and the new updated values from your neighborhood will be available to all the nodes in the other partitions and correspondingly, they will actually then update their entries. So, that will be happening, because they are again connected to their partitions closely so they will be also exchanging neighborhood tables. All the nodes there also will update the entry because of your getting removed from the system. So, network will never get partitioned that is being ensured with this mechanism.

(Refer Slide Time: 35:05)



what concepts we discussed ?

- logarithmic partitioning
  Nodes cannot become isolated
  due to failure of some other node.
- The <key, value> entries are genuine.
  digitally signed <key, value> entries
- Denial of service attack – subverted key
  ensuring randomly generated node IDs.

And we also looked at the how the key value entries are kept genuine. This is done through digital signatures and because these are signed through identity certificates issued by Certification Authority. So, nobody else can actually put an entry. Your node ID can change, your IP address can change but your identity certificate remains the same, your private key remains the same, public-key remains the same. So, this signed entry becomes a valid entry.

But of course, as an endpoint address probably we should not keep IP address port number kind of things. I will actually explain in the next lecture how this is being handled. We still normally should actually keep node IDs there of the destination node, but then we require not only publish and query messages, there are other kinds of messages also which are required. I will list down all those messages in future lectures.

And also we also learned about how to subvert the denial of service attack by ensuring that all node IDs are randomly generated. So that is being done through again, public-private key pair generation. A user can be connected to multiple nodes with the same user ID, basically same identity certificate, but different node IDs can be present. So that actually is technically feasible with this kind of design. So we will look at it in the next lecture on Voice over IP thing with which we started, and we will discuss how Voice over IP will actually work in such a scenario with all the basics, which we have learned so far.