Peer to Peer Networks
Professor Y. N. Singh
Department of Electrical Engineering
Indian Institute of Technology, Kanpur
Lecture – 25
P2P Mailing List Services: An Alternative Design

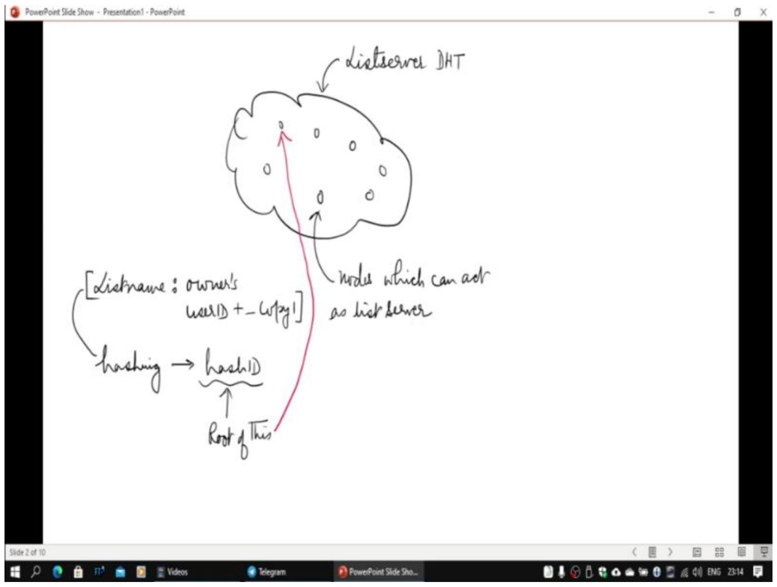
(Refer Slide Time: 0:15)

Lecture - 25

Something more on P2P mailing lists.

In the earlier video, we have discussed peer to peer mailing list implementation; but, some of the discussions were missed out. So, in this video is I am discussing something more on this peer-to-peer mailing list.

(Refer Slide Time: 0:32)

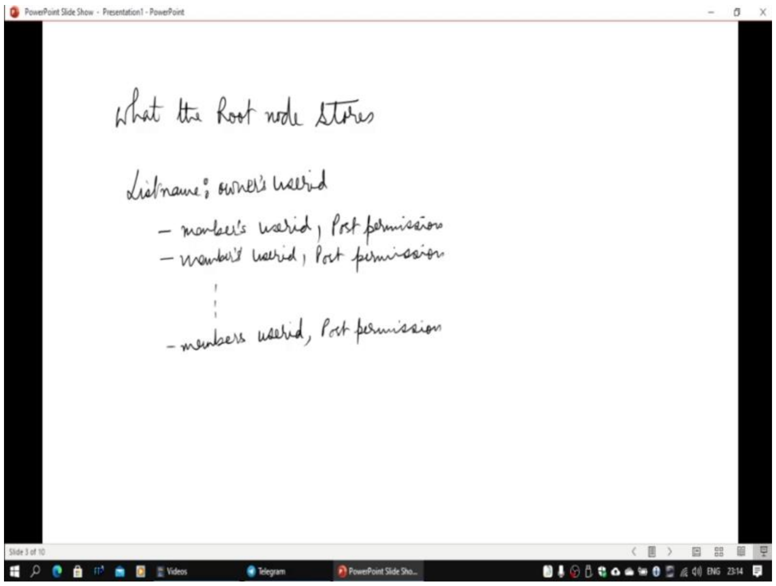

Listserver DHT
nodes which can act as list server
[Listname : owner's mailID + _copy1]
hashing → hashID
Root of This

We discussed in the earlier one that there was a list server DHT, so all the nodes that can act as a list server will register in that DHT. So, any node which wants to find out that want to generate a list. He will create a list name; he will also append his owner's ID the way it has been shown here.

So, this will be a colon which will be used and then, of course, a copy 1; the reason being I am implicitly assuming that there were two copies maintained for every data structure for the list. So, this for resilience purposes, the way we had done in the discussed in the earlier videos; same way it is done here. So, I am not mentioning that when I say copy 1 computing hash, it goes here; it will generate a copy 2 actually, which will go to another root node; which intern will maintain the copy 1.

So, if a node goes off automatically, the copy 1 structure will move to somewhere else. A new node comes in that should become the root node; then all the data structure will move, so all that will hold. With this, we do compute the hash of this particular structure; we will append copy 1 here, we will get a hash id. We will find out the root node for this; let this root node be this particular node. These are all these black circles that I have shown inside this list server DHT; these are the nodes, which can act as list servers. I have shown it node ID space, and these nodes have formed a list server DHT.

(Refer Slide Time: 2:04)



What the Root node Stores

Listname; owner's userid

- member's userid, Post permission
- member's userid, Post permission

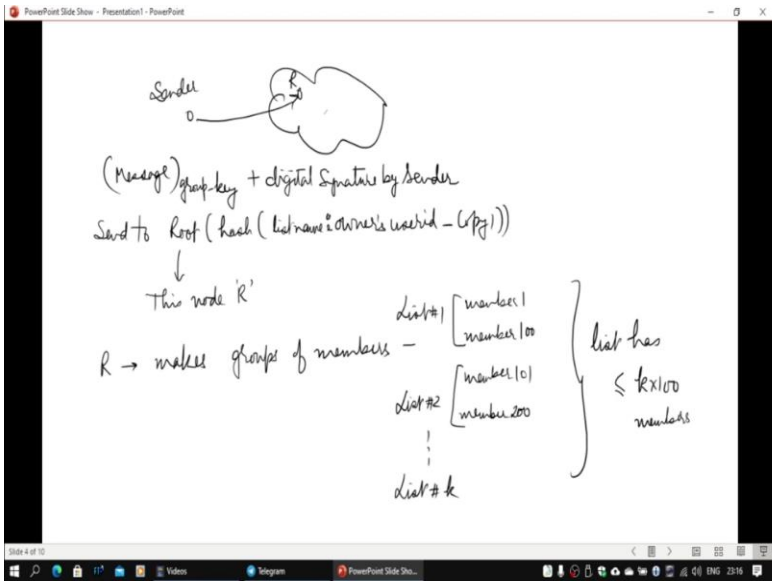
⋮

- members userid, Post permission

Now, what this root node is going to store? So, when the owner is going to publish. You are the guy, so I am creating a list. It will create a list data structure that will be, which will look something like this. It will contain the list name, owner's user-id, and when you do underscore copy1, compute the hash and root; you will come to the root node. And you will, this particular structure will be published there; it will contain all the members' user-id, whatever number which are there. Whether they have permission to post or not? Whether it will require moderation or it is a moderated free.

And of course, this whole list needs to be signed by the person who has created the list. The owner's user-id should have signed this particular list; otherwise, the root should not maintain this data structure. So, this was not stated earlier explicitly, but I am stating it now.

(Refer Slide Time: 2:55)


PowerPoint Slide Show - Presentation1 - PowerPoint
Sender
R
(Message)group-key + digital Signature by Sender
Send to Root (hash (listname : owner's userid – copy1))
This node 'R'
R → makes groups of members –
List#1 [member1 member100
List#2 [member101 member200
List#k
list has ≤ k×100 members
Slide 4 of 10

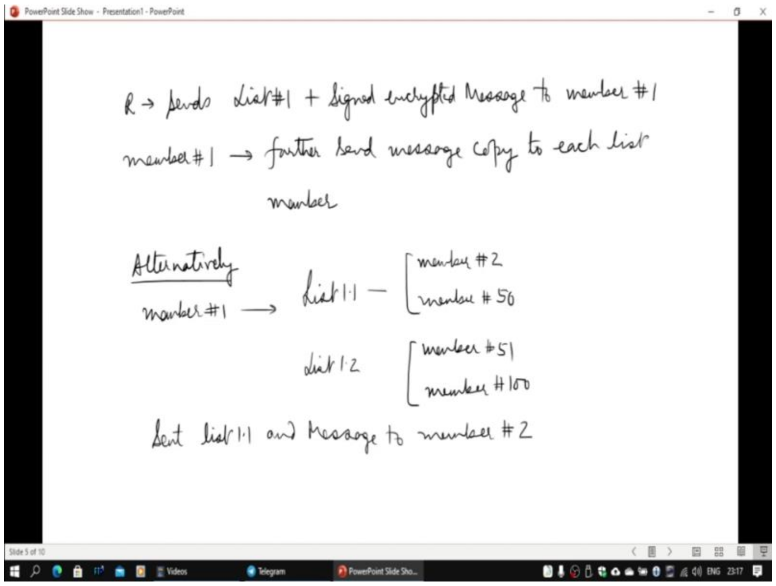
Now, a sender wants to send a message to this list; how can this be done? So, I have discussed this thing in the previous video. So, it will create a message, and as I mentioned earlier, that will be a group key. So, there is a key for the whole group, which the group owner will be sending or dispatching to all members.

When the membership is going to be updated; that time, it is going to generate a key and send it. Of course, when you remove somebody from the list, you remove it from the root. But, if somehow that guy still gets a message, he will decrypt it; unless you send/change their group key and then communicate to everybody.

Anyone who wants to post will take the message; it will encrypt with the group key. It will attach its signature to identify who has sent this particular message to the group. And then, it will again compute the list name colon and the owner's user-id; we will append a copy 1 and compute the hash. And to this root node, it is going to send this whole message.

The root node in the list server DHT can now send it to all members of the list. So, let us call this node R; so what R will do is it is going to have a huge list. For example, say K into 100 kinds of the list to create K groups, each one hundred. And of course, once it has been done, the message is there, so this is discussed in the earlier one. So, I am going to today talk about an alternative method also.

(Refer Slide Time: 4:26)



R → sends List#1 + Signed encrypted Message to member #1

member#1 → further send message copy to each list member

<u>Alternatively</u>

member#1 → List 1·1 — [member #2, member #50]

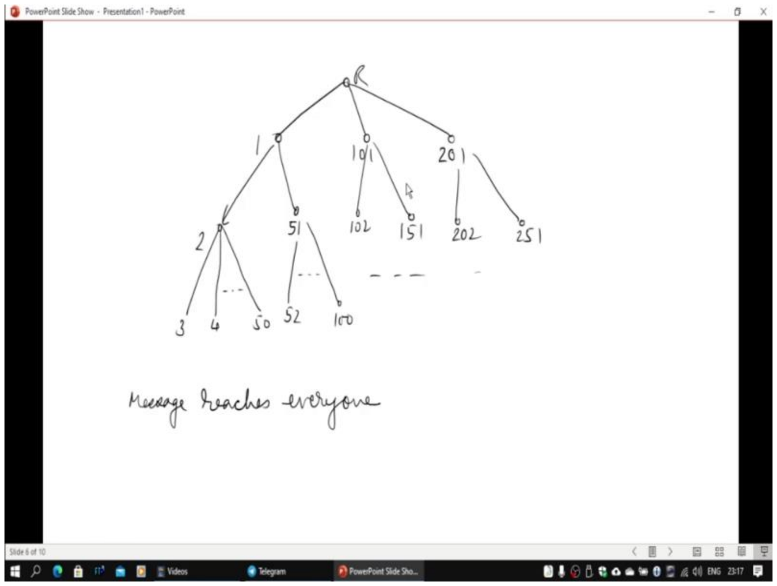
List 1·2 [member #51, member #100]

Sent list 1·1 and Message to member #2

So, I will send list 1 the complete list, plus signed encrypted message to member 1; so, member 1 will further send the message copy to each of the list members, so from member 2 to member 100.

Alternatively, member 1 can further split the list, so like this, member 2 to 50 is the first list, member 51 to 100 in the next one; and then it can send this detailed list 1.1 and the message to member 2. And member 2 will further then look at this whole list and send it to 3 to 50; member 51 will be getting a message again from this member 1, along with this whole list, and he will further split it from 52 to 100 members.

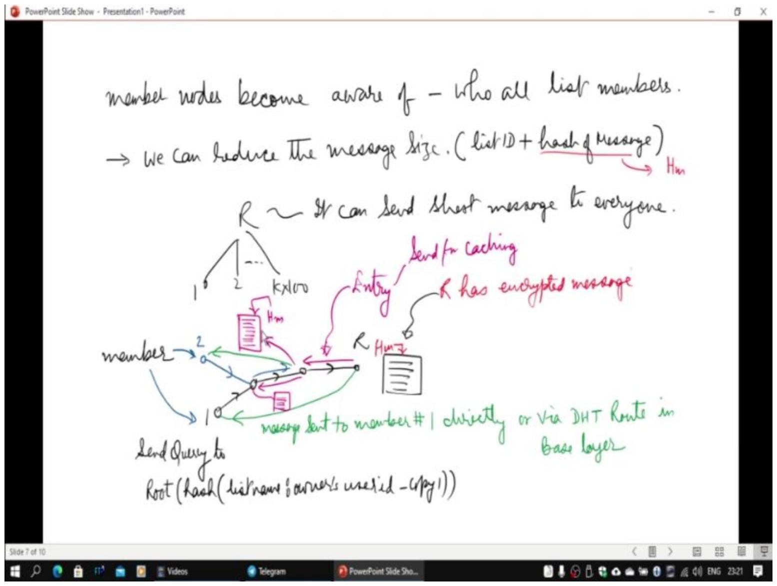
(Refer Slide Time: 5:10)



Message reaches everyone

Now, this will look something like this; I have sent it to 1, 101 and 201 along with the list to who; or this is a list of all the children to whom the messages have to go. 1 has further solved it out and gives again a sub-list of the children to whom the message should go. And given it to the two of the nodes, which can do the further for, 2 is giving it all 3 to 50. 51 is giving to 52 to 100, and so on, this 2 is for even for these; so message reaches to everyone. So, this is what I have talked about in the previous lecture.

(Refer Slide Time: 5:43)



member nodes become aware of – who all list members.

→ We can reduce the message size. (list ID + hash of Message) → Hm

R ~ It can send short message to everyone.

R
1
2
Kx100
Send for caching
Entry
R has encrypted message
Hm
R
Hm
member
2
1
message sent to member #1 directly or via DHT Route in Base layer
Send Query to
Root (hash (listname & owner's userid – copy1))

The problem here is that each member can become aware of who all are the list members; it may not be the whole list, but even the partial list will be available to them. Sometimes it is not desirable; in fact, in most of the list server implementations. It is only available to certain people; not everybody can see who all are the members; unless you permit it in the list. So, we need to have an implementation that needs to be changed.

So, what we can do is why I was not sending it to all K into a hundred nodes; because the message was large and sending it would have taken time processing and sending. So, the alternative way is that R can keep the message with itself, so R has kept a message, for example, here. And will compute the hash of that message, and this R now sends it to each member; a message that will contain the list id and hash of the message content.

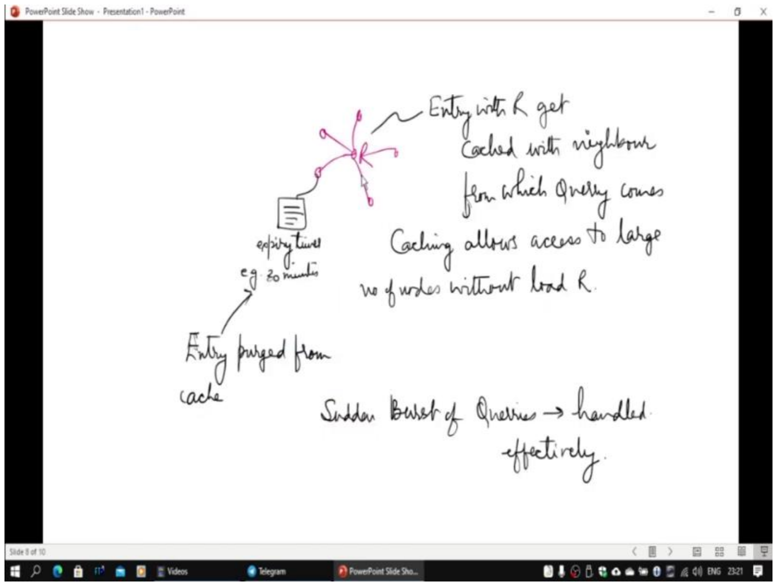
So, the message is not going to be sent, so everybody will come to know that there is a menu message, which has arrived in this list id and this the hash value. Now, here we can play a trick around, so member 1, for example, sends a query that I want to get this message now. So, it knows already list id, which is going to be list name colon owner's user-id, it will append copy1; compute hash, find and it will send to this particular destination or request that I want to retrieve this message with this hash; with the Hm which has been given. The message will reach the root node, and the root node will go either through a DHT root or directly through the endpoint address that will send the message.

So, member and member1 as a list member has got the message now; and he can read the message. Now, we need to do something more because if there is going to be, say, 10000 members; there all of their queries will be coming to R, and R is needs to serve it. If suddenly all of them become active and all want to access it, it will be a bottom line. So, we need to use another phenomenon called caching here. So, when it sends to 1 will also know that from which the root query arrived. It will also send a copy of this particular message; remember it is encrypted, so it is safe.

And the hash value goes through the previous node from which the query has arrived, and it will be cached here, so the cache is shown here. There is an index the hash content hash, which is used as an index entry, a key entry, and there is an encrypted message; this is now available with it. Now, suppose member 2 also wants to get access to it; it will send a query, query travels, it reaches here. There are already things in the cache, so every node, wherever a query arrives before forwarding towards, the root will search if it is already there in the cache.

If it is not there, it should only send it towards the root node; otherwise, you serve the message from your cache itself. It will now search for this content hash, it will take this message out, and it will send it directly or through DHT root to member 2. And it will also then send a cache entry for caching towards the node from which the query arrived. What happened as more and more people are trying to access it, so more and more neighbours around R will start having the cached entry for this particular message.

(Refer Slide Time: 9:03)



Entry with R get
Cached with neighbour
from which Query comes

Caching allows access to large
no of nodes without load R.

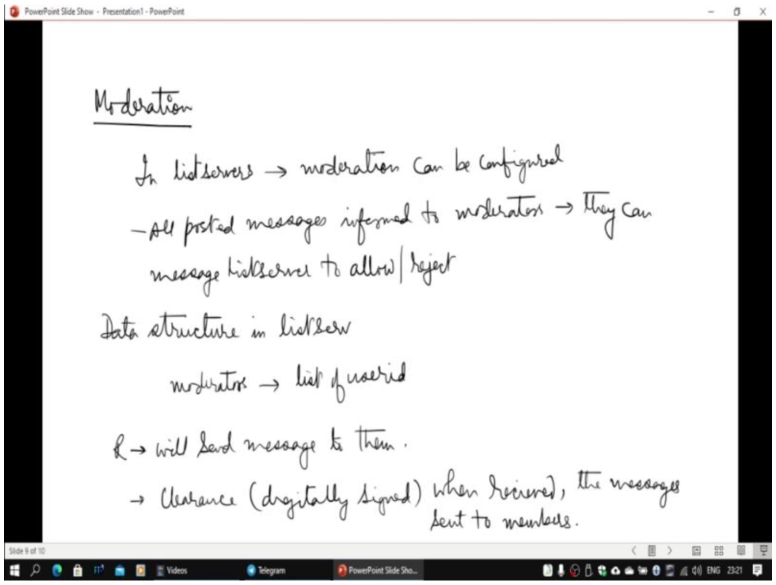
Entry purged from
cache

Sudden Burst of Queries → handled
effectively.

This essentially means that R all the neighbours will start having the entries, and more people will access, it will start this spreading out more and more. And this caching will allow a large number of nodes in a brief time without loading the root node; can have access to the message.

And we usually for this cache messages you can keep expiry timer of say 30 minutes. When the load reduces, not many people are accessing; this will automatically purge from the entry. So, whenever a sudden burst of queries happens, those can be handled very effectively with this method.

(Refer Slide Time: 9:38)



Moderation

In listservers → moderation can be configured

– All posted messages informed to moderators → they can message listserver to allow | reject

Data structure in listserv

moderators → list of userid

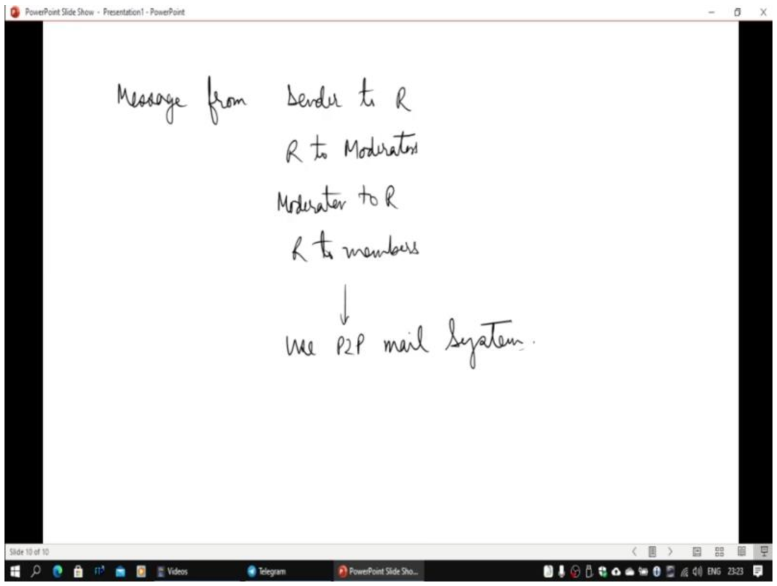
R → will send message to them.

→ Clearance (digitally signed) when received, the messages sent to members.

And of course, one more feature exists, which is moderation; I had also not discussed anything about moderation in the earlier case. So, moderation the moment that comes into the picture, and moderation is configured in them. So, all posted messages which are not permitted to go through without moderation; they will be just held up in the root node R., And R is now supposed also to maintain a list of the moderator, let us say another data structure in the list server; which need to be now populated by the owner. The owner will say these people are the moderators, so he creates the list; he creates the moderators.

And whenever a message comes, that needs to be moderated clear; a message needs to be sent by list server or that root mode to all the moderators listed. So, all the moderators will get it, and once the moderator will get this particular list. It is then to clear it, but if they give a clearance message or give a message to reject it or drop it. The method has to be digitally signed so that the list server can be identified by who/which particular moderator is responded to, and then it will act accordingly. So, tomorrow, for example, if somebody has done a wrong moderation, the owner can find out who did this wrong thing; and he can be removed from the moderator list. So, once the clearance is received, the message will be sent to all the members.

(Refer Slide Time: 11:09)



Message from Sender to R
R to Moderators
Moderator to R
R to members
$\downarrow$
Use P2P mail System.

We need to understand that message has been sent from sender to R in this case, which is a root node or the list server. R has sent the message to moderators, moderators to again send a message back to R; R has sent it to members, or R has sent it to members who have further send it to members.

All these messages are not direct transactions; they all use to peer mailing system to send messages because that is important. Usually, when you are sending it to a list server, so it may be off; but in that case, somebody else will be acting as a list server for you.

The list server is always alive, so it can directly peer-to-peer transaction; we do not require a mail storage system. But when you are sending to moderators or sending it to members; then you will be using peer to peer mail system for transacting or transferring the messages.

This was something additional that I missed up in the previous lecture, so I wanted to clarify, so I had done this in this particular one. In the next video, we will be looking into a new peer-to-peer web; and now the peer-to-peer search engines are built using this whole formalism.