Peer To Peer Networks

Professor Y.N. Singh

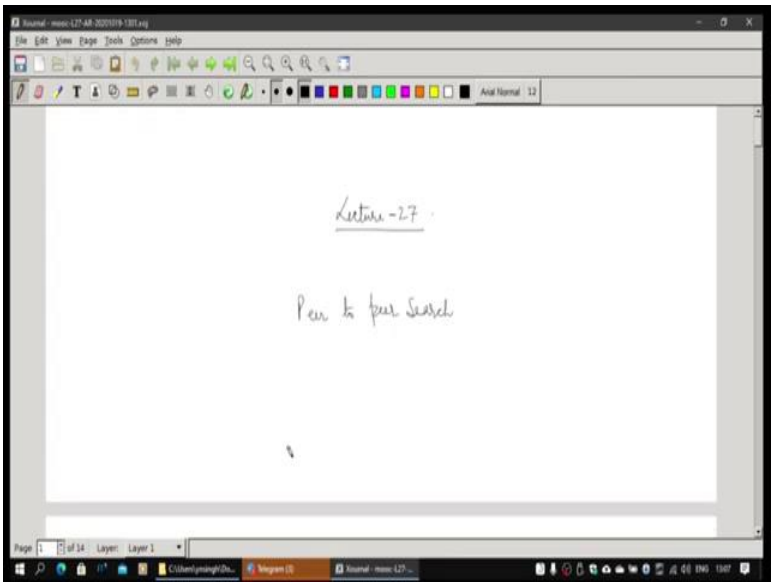
Department of Electrical Engineering

Indian Institute of Technology, Kanpur

Lecture 27

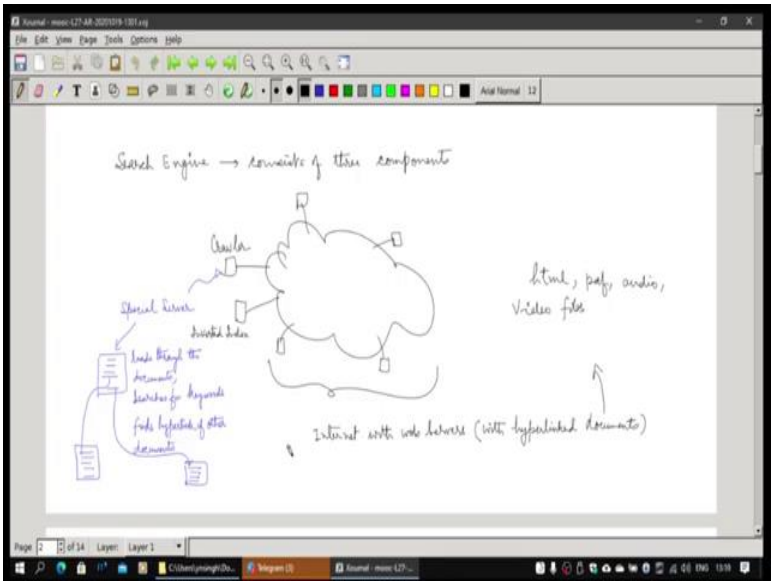
P2P Web Search Engine: A Basic Design

(Refer Slide Time: 00:17)



Welcome to lecture number 27. For the peer to peer look in this. In this lecture, I will discuss peer to peer search mechanisms and how we can create them. So, this is like implementing your own Google. So, Google is a search engine; you put in some keywords and know which all websites are related to that particular keyword.

(Refer Slide Time: 00:46)



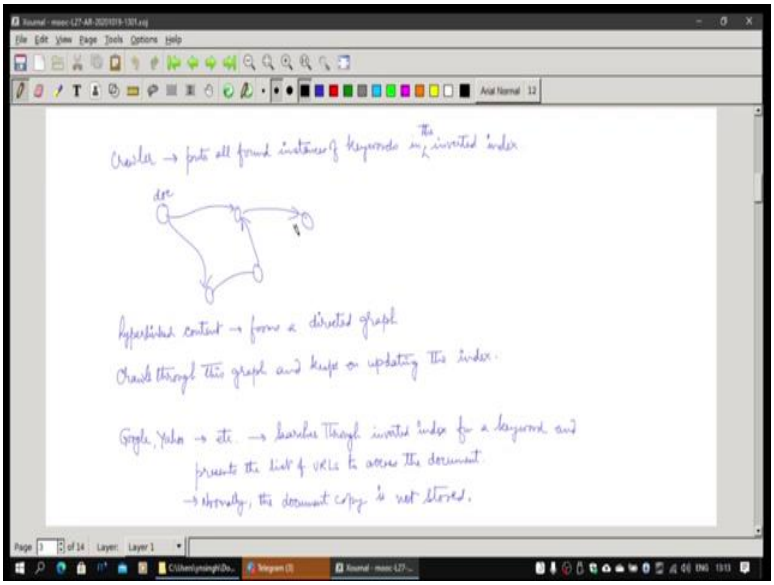
So, in any search engine, typically, the way it happens is; it consists of 3 components, so there will be a crawler. But these are the particular kind of server which special software goes to a website, and then it knows at least a few documents and goes to that document. That document will have hyperlinks to a lot of other web pages all across the sites.

We passed through the page sign out other links, and if the page then it will figure out what is present in that. It will then create an index entry for that, and it will say, okay, this for this keyword, this is the document URI. URI is Universal Resource Identifier, which means what kind of transport HTTP your HTTPS or FTP or telnet any kind of protocol can be put a colon and then, of course, the machine's machine's domain name where it was found and the directories structure to the route of their webspace. So that will be documented, and that will be kept in a database. And we call it an inverted index. So, this crawler will go there, and from that page, it will go to end the page, then to another page, so it keeps just roaming around to the pages on the internet and keeps on updating the inverted index being maintained.

So we assume that the inverter will now have a hyperlink text document for which it will create an index and have PDF it can have audio it can have video files. Now, crawlers are being built to scan through PDFs, scan through audio, scan through video files, and do an image search. So indexing of any kind of content they should be capable of doing, but they just keep roaming around one place to another and creating the index. So, that is how the search engine works

typically. So these three components are crawler, inverted index and the internet with web servers.

(Refer Slide Time: 02:54)



Usually, the crawler will put all found instances of a keyword, so naturally, every crawler will have a keyword: its indexing. It will say this document is there and as a document will be linked to this particular document referring to them and those documents referring to some other documents. When you search for it now, remember a document created later will always be referred to as a document created backward in time.

A document that is in the past cannot refer to a document in future. You usually pick up the latest whenever new documents are edited. They have to be searched for the existing keyword essentially, and their entries have to be created. We have to find out the backward references and then check any new document. This is any new document identified by this. The current one I am crawling through then is updated; otherwise, it will be older documents.

If you do this, these documents will form a hyperlinked or directed graphs. This direction will always be from present to past, mostly unless we change the older document later and add a current document. So, these hyperlinked contents will form a directed graph, so the crawler will simply pass through this graph; keep on moving around, keep on moving around. So it may not

be searching through a whole tree, so a document can, for example, link to, say, 10 other documents and each one of those 10 is further linked to 10 other documents.

So, as the depth of search happens in the tree, what will happen is your depth is the 100 discovered those many documents would be there in the lowest layer. Now, the crawler will be overwhelmed with that, so, naturally, they will keep on doing it randomly. They try to maximize the probability of finding out some new information that can be put into the index.

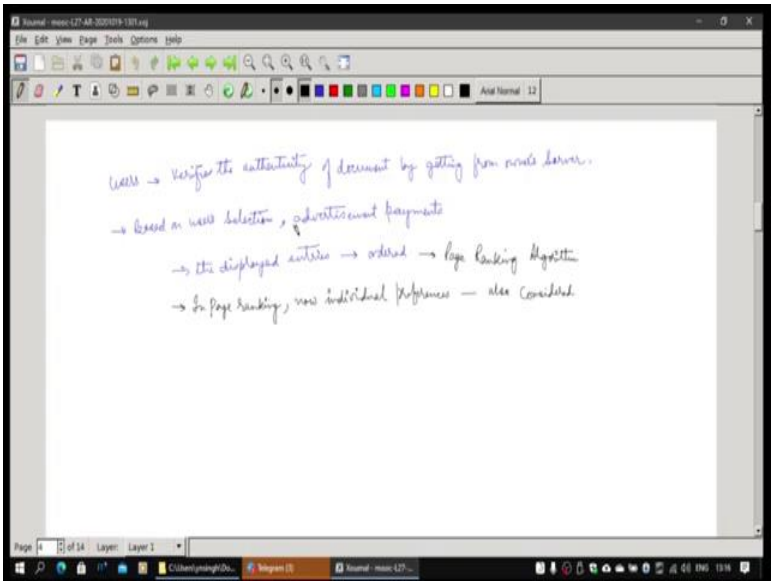
So, the crawler technically is crawling through this graph, representing distributed documents linked to each other. Google, Yahoo etcetera all these does this crawling business, and they create their index. When you go to a Google website and input your keyword, this takes the keyword and searches through the inverted index. We call it an inverted index because I am not creating that entry based on the document, but based on the keywords; I am creating the entries.

So, if a keyword is also repeating multiple times in a document, I will also find out the document's location where that keyword is being appeared. So, this is a keyword-based search, and this has to be done through crawler only. So, the forward index is when you create, which is not an inverted one, which is where you create an index that is a forwarded index at the beginning of the book.

In the back of the book, what you create is an inverted index. It will look into the inverted index from a keyword and give you a URL list to access the document. This implies Google does not, or crawler does not store all the documents it actually can what it requires to use space so what does typically is it only just updates an index forgets the document move on to the next one and keep on doing it like this.

Document copies are not stored. I mean is not too many people have archived the internet at various points in time. You can go back, and you will see the deleted pages that are feasible. But this unendingly puts a lot of pressure on the storage. So, people will keep on doing it when you have to start forgetting the past for that it is available.

(Refer Slide Time: 06:30)



So, we usually need to verify the authenticity of the document you are getting from a URI. So, Google why it is not keeping a copy; if it keeps and you get it from Google, you will say Google has given me the copy how you verify the ownership of that document's authenticity. It is always better to get it from the owner's server, so Google usually will not fetch it for you unless it is in the cached form. We will have this challenge because we will do it in a way with the servers, in our case, in peer to peer system.

So, that is a reason you search from inverted index go that user server and fetch it from there, and you know this is a user server, so you are having more faith in this.

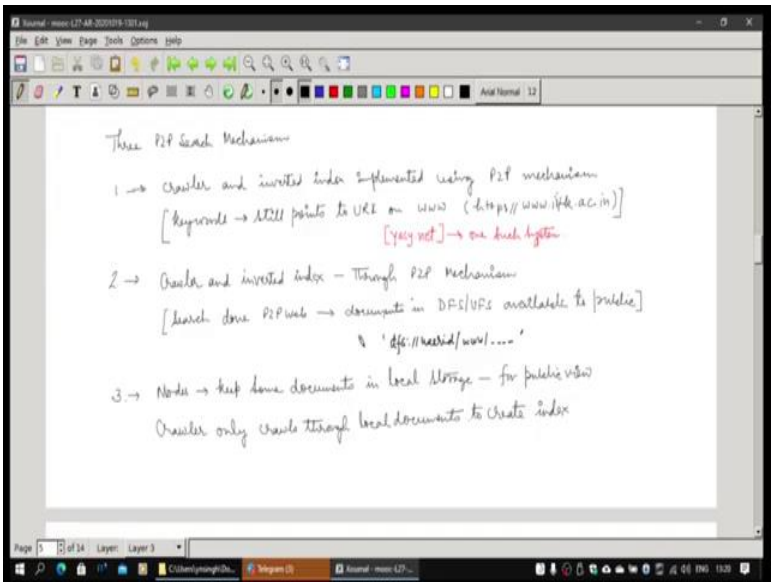
Then news coming from Google servers directory is not authentic than the news coming directly from the newspaper seller. Google and Yahoo also do something interesting, something more on this; it is not merely a search. And that is essentially where the biases can come into the picture. They look at how you are browsing and what you like and what you do not like. So whenever you search a keyword, they will now try to organize all the inverted index entries. They will filter it as per your liking, as per your profile.

And of course, they will also decide if somebody has paid for individual pages to be given a higher rank, so they will always show depending on the keyword that information also to use. For example, your search for a computer on Google, and after some time, you start getting the

advertisements at all other places. So it gives a keyword specific top two entries are technically advertisement. So that is how they make money.

They have to make money to survive anyway. So displayed entries will get ordered in some fashion depending upon how many people are using it, which is popular and more own profile and the advertising industry payments. So, page ranking is the one contribution that was there and what took off Google to what it is today. So, because people like and their individual preferences are being accounted for in the page ranking system. Pages are ranked accordingly.

(Refer Slide Time: 09:03)



P2P search when I want to do now. I am technically now trying to replace Google, but now I am doing it because I am the peer client who will be doing this job. So, I need to do now crawler and inverted index. These two need to be implemented; since there are no servers, they need to be implemented using peer-to-peer mechanisms. Of course, the search space can still be the same internet to have keyword and URI, URI can still be the same HTTP colon. So, this idea was already thought about, and yacy.net is one such system that is available. You can download and install it.

So peer clients together do crawling and forming the inverted index, and they only give you the URL of a server on the web from which you can fetch the document. They do not again store any of the documents. So, that is a worth type one kind of peer to peer search mechanism. So,

crawler and inverted index in peer to peer system but what you are searching in the overall web space which is already existing pre-existing one.

There is another type which we can create where crawler and inverted index are again implemented through peer to peer mechanism. So, we search for our document the peer to peer web, which has been discussed in one of the earlier videos. So, any user can now create a DFS:// something like that, and then, of course, search in that space DFS:// whatever is the user ID and then WWW. So every user is now creating in this space, their web, peer to peer web. So, we can search through this and then create the index of whatever has been published by people.

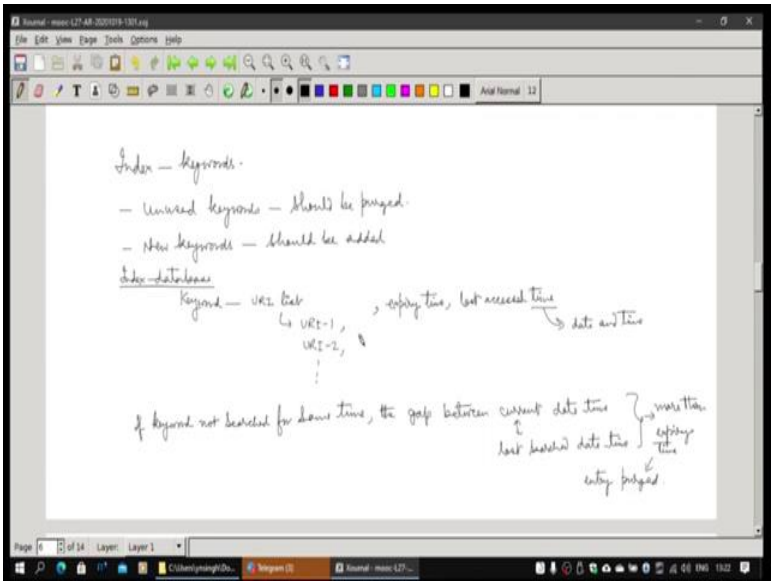
The good thing about this peer-to-peer web is it is still authentic because users digitally sign it instead of the domain name. Now the user ID or the email id will be where the system will be located. The web space will be routed now for every user, and we create an index for that. Now, this cannot be crawled by is not crawled by currently by Google or somebody else, so this needs to be done. And even if the user is off, you can still get the answers or documents, which are verified documents and indexes will be maintained in the same way.

So, this is like creating a DHT layer of the inverted index, and whenever you find the document for keyword mapping, you just publish it as a DHT publish. It will go to a corresponding route of a hash of the keyword and copy one, and there the URI and this mapping will be stored in the database. And copy tool will automatically get created from there. There will always be two copies maintain for resilience purposes again.

And the third possibility is that we do not use the DHT; you are a node. You have some document to make it available to the public, but only you restore it; it is not in your DFS web. It is on locally on your document space. And you can now also have your local crawler. So, you only want to know what kind of index entries you need to crawl for. So, your crawler will only be looking at crawling into your own space and creating your local index.

Now, how the queries will reach you? You are not a there is no DHT mechanism here. So, in this case, we will be going for an unstructured search mechanism.

(Refer Slide Time: 12:46)

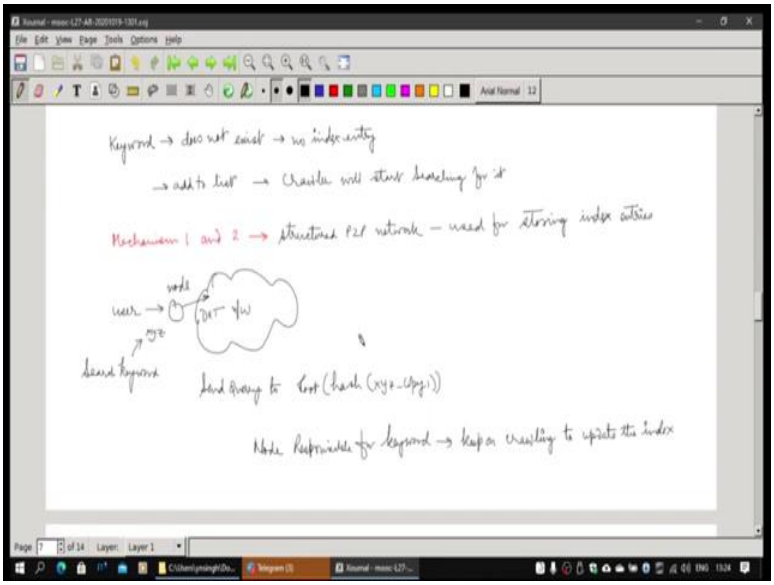


Now, coming to the index keywords, how those will be handled. So, there will always be unused keywords. Some keywords have not been used for many, say many years, 5 years or 6 years. What should be done; should I still maintain the inverted index? Probably, no. Many unused keywords should be purged if they are not used for sure for more than a certain period, and there is a possibility of having even new keywords.

There is a new keyword. There is no entry in your inverted indexes; you would not give any results, but then the crawler now learn this keyword and start crawling with this keyword and then start essentially populating the inverted index for this. Similarly, the crawling rate for different keywords can also be modified depending on the usage frequency of various keywords, which can be different. So, Google, Yahoo and everybody also must be doing a similar thing.

Any keyword that will search any index database will typically have a keyword and the list of URIs that need to be referenced for this keyword. And it will also then have an expiry time of this keyword and last access time. So, I think every server, every DHT, every node in the DHT will keep on looking into this and will keep on monitoring if the current date and last search date time, the gap between them is more than the expiry period. Entry should be purged to create the space actually for more inverted index entries. So that is what everybody will be doing.

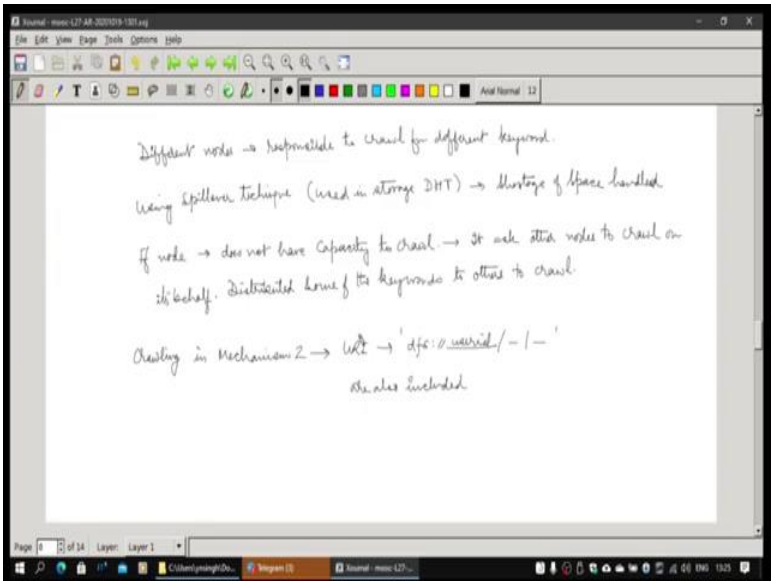
(Refer Slide Time: 14:20)



And if the keyword does not exist, there is no index entry you just simply add to the list, and the crawler will start searching for it. So with mechanisms 1 and 2, search space in the whole worldwide web or your DFS thing. We can use a structured system, structured DHT, for storing the index entries. And DHT structure peer to peer network means DHT based system is peer, structured peer to peer. So user, for example, searches for keyword XYZ, you will compute the hash of XYZ underscore copy one and send it to the root node of that, so it means the query will keep on moving around and into a DHT network and it will go to the responsible root node.

And this root node is also responsible for doing the crawling for this keyword. So, crawling is also not been distributed. If it does not have the capacity, it can always ask somebody else to crawl on its behalf and then publish the entries back to it. And this can also have a mechanism whereby it can start publishing this. If it cannot have a storage space, we can use a similar storage DHT layer mechanism. Where the entries can be distributed across everybody, and this does not burden people.

(Refer Slide Time: 15:41)

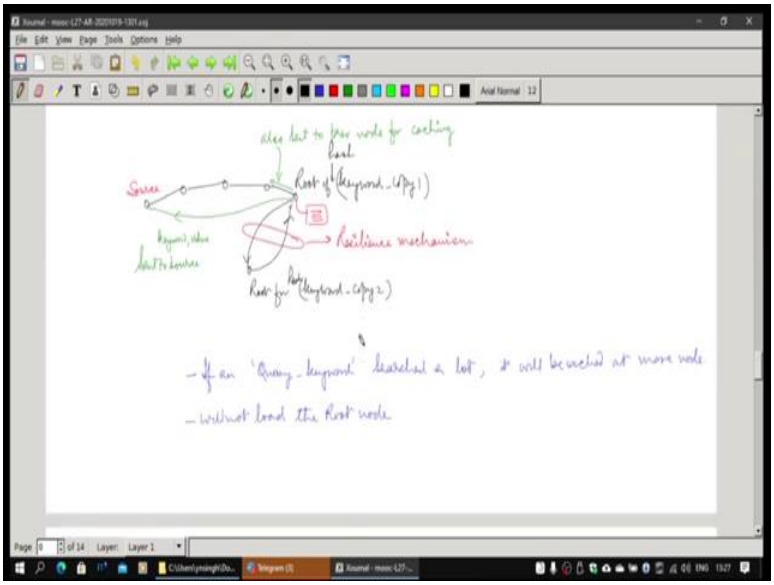


So this implies that different nodes will be responsible for crawling for different keywords, and you can use the spillover technique as used in storage DHT about the shortage of space to be handled. Of course, we do not require a reputation because there is nobody thing, it is a collaborative thing, so it is mostly philanthropy.

If people do not want to contribute, it is going to be very limited in that sense. But a corporation can put their servers as a DHT and then provides space for doing this. Internally you might guess most of the search engine companies are so internally for their servers they are creating very similar systems. And of course, if a node also cannot crawl, it can ask other nodes to do it on his behalf, as I mentioned.

And mechanism 2, the crawling will be done in our DFS space. The URI will be of DFS colon slash, slash user ID slash something of that form, so those will also get included, which are not there in mechanism 1. So yacy.net only provides for mechanism 1. One should try installing it is available on the snap store of Ubuntu. If you have an Ubuntu machine, you can download it and directly install it from the snap store and run it.

(Refer Slide Time: 16:48)



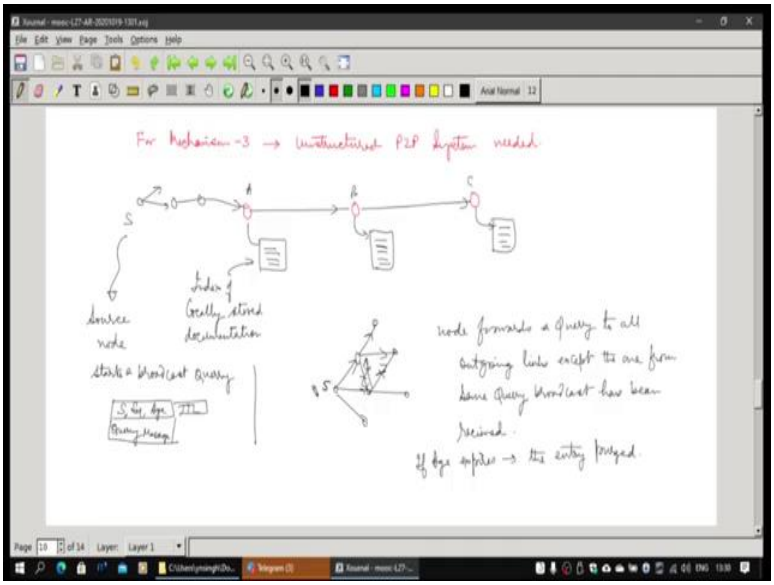
Yacy dash search is a command by which you start with the command line, then you can then start browsing and are not doing it on Google, so the advertising biases are not going to be there in this case. The way it will happen is whenever you are going to search, a source will be sending a keyword. So what will it be doing? It will be sending keywords via the DHT route. It will go to root 1, which is the route of a hash of keyword copy 1.

So, this particular root node will also be publishing a backup copy, which is a root for hash of keyword copy 2. So, there will be essentially doing a resilience mechanism by monitoring each other and republishing periodically. So whenever the query reaches it, it will send the keyword and value to the source directly. The source will be telling maybe its endpoint address or DHT node ID. Both cases may take a different root, not the root through which the query arrives.

So the root mode also now should send the nodes on which it receives the query to that also the keyword and value so that can be cached there. So this is what we call a smearing phenomenon. You smear the route entries stored at a node around to the neighbouring node, typically when trying to access the same key-value pairs.

The more keyword key-value pairs are accessed, the more smear they around their node are responsible for it. So this will avoid the loading of the root node and makes the system far more efficient.

(Refer Slide Time: 18:32)



For mechanism 3, where there is no structured system required because now you are trying to search through the nodes' content at itself, they are not given into a DHT or in a DFS. It is important to note that if a key-value pair exists, you will always find it if it is going to mechanisms 1 and 2. But in mechanism 3, there is no guarantee that you will get the key-value pair if it is existing. If it is with a high probability, you will get it depending on how far your query gets propagated.

Now since it is a structured, unstructured thing, so there is no DHT routing. You do a broadcast routing in that case. You know all the neighbours in your routing table in your neighbour table, which has been created as we have discussed in the earlier videos when routing. The source will simply create a broadcast query. It will put its source node ID, say sequence number.

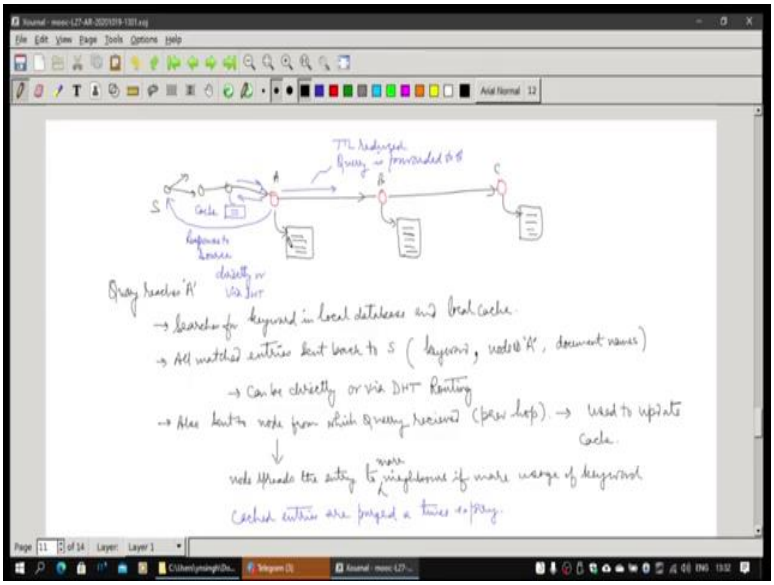
Every sequence number has to be incremented whenever it initiates a query and the age after which the entry can be purged from the broadcast table. This is broadcasted to everybody. A query message will be attached a time to refill is also going to be added. So it goes to all of my neighbours; they will send it to all of their neighbours at any point in time. For example, see here this guy S has sent a query to these neighbours, they also send it to each other because they were connected, but then the broadcast table will say this sequence number from this source. I have already been received, so I am discarding it. So they both will discard.

Everybody will only broadcast only the first copy, not the second, third and other copies. So this will keep on spreading like a tree to in a surrounding of source S, and everybody will receive the broadcast query. Everybody is now maintaining an index of a locally stored document. Hence, they have to essentially receive a query they have to search in the local documents. If they have some documents, they will send the document back. They need not send the document itself.

Later on, we will go through all the key-value pairs and, based on that, may ask you directly to fetch the document. So that usually should be the procedure. Now when you are broadcasting, the age field is essential, so if the age expires of an entry in the broadcast routing table, the entry will get purged. This is typically required so that if S goes off and then comes back again, it will start using a 0 sequence number from 0. It does not remember the earlier sequence numbers.

Usually, its queries will be disregarded or simply will drop because the sequence number which is there with the neighbours is higher. So once they are purged because of this age, this source can start making the queries again. Usually, this happens pretty fast, so it is not an issue.

(Refer Slide Time: 21:39)



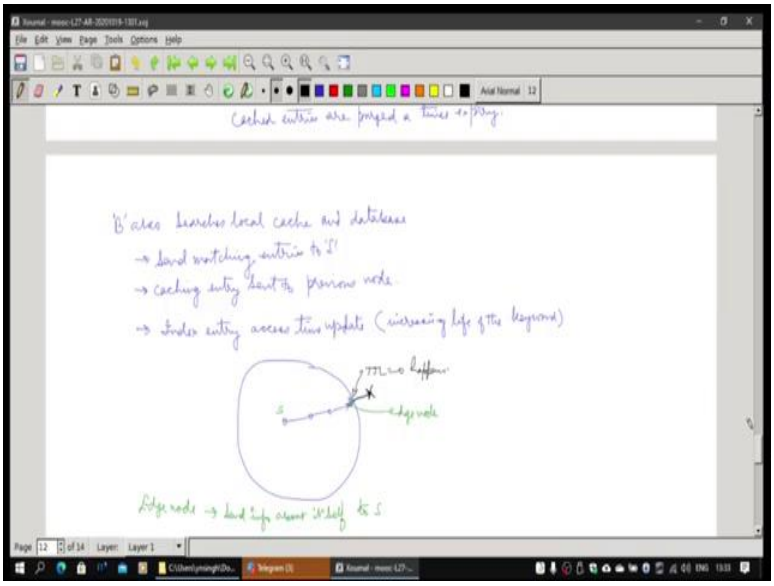
So, now typically, the way it will happen the query comes to node A, node A will search, it will send the response back to S, but it will also send the response back to the node from which it receives the query. So it will be kept in the cache. If you see the way, so the idea is a cache searching somebody in the neighbour of S may also search for it, the query's locality nearly that.

So, in that case, it will be satisfied from here itself and the response back, and then they can talk to A and get the feedback get the document.

And if S gets the document, it can also keep a copy, but A's authenticity cannot be absolute unless its document is digitally signed. And then, of course, A will also send the query forwarded to B and a TTL value time to leave value will also be reduced. So, B can also again search the document in the same fashion and respond. Now this entry you are sending back can go via again DHT rooting or IP routing. That is the way it has been told. If any query is processed much faster, that query will become more smeared, so it becomes much more readily available in case nodes dies off in that case scenario.

Of course, the content will not be there, but content can also be cached actually. We can even have a provision where the same content gets replicated to the cache also. This guy can tell let another content and let the content be digitally signed, so authentication is taken care of by my images. And cached entries need to be purged after timer expiry. You keep the usage of the space optimal.

(Refer Slide Time: 23:23)



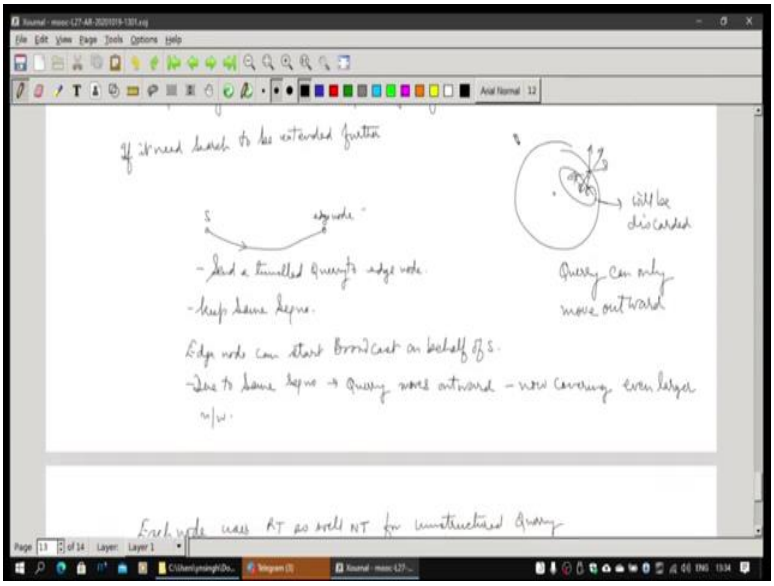
Now whenever off query comes, you will also search the local cache and your local database and send all the matching entries to the S. And if any entry which is being matched, those also be sent to the previous node for caching purpose. So there can be multilevel caching, which can

happen if the entries being requested again and again. And of course, now it can happen is that when such someplace TTL can become 0. If the TTL becomes 0, then the node will no more forward, so this entry will not be going any further. This will simply be dropped here. So as a consequence, the query will only propagate to this particular range.

But edge nodes can do something more here. They can send information that I was the edge node to get TTL to become zero. And this is my node ID or my IP address port number, say direct endpoint address or node ID both can be, either of them can be given. So S will get all the responses from within this zone, decide by the TTL.

Now let us say I am not happy, and of course, now another important thing when a node makes a query will keep getting the answers one after another sometimes. After some time, it will stop getting the answers. So it may decide I am not happy with the search I want to do much more of them.

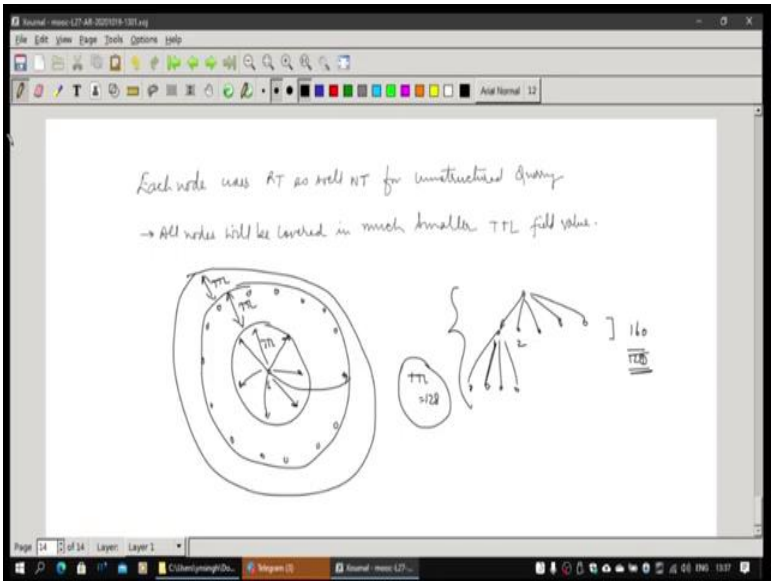
(Refer Slide Time: 24:45)



So it can now communicate to the edge node directly and say kindly you initiate a query on my behalf is tunnel query to the edge node. It can keep the same sequence number done earlier through which it figures out the figured out edge node. So what will happen is the query will go to the edge node it will start the query again, but now TTL will be full actually when a new one is starting.

When the query starts moving backward, they will be discarded; their broadcast routing table already has that sequence number. So, it can only propagate outward, readily outward to the source. So in the first and when the TTL goes 0 again, this will get stopped.

(Refer Slide Time: 25:28)



So you will have a scenario where the source broadcast goes out. There is a node you tell him it starts broadcasting; it can probably tell too many edge nodes, maybe all of them. They will broadcast, so they are now your region of influence will become this you will start getting queries from here.

And next time, the edge nodes from here will be informed to you. The number of edge nodes that you will be getting will become larger and larger. The next time you can tunnel here and this, all these guys will do it, and again, the region of influence will become larger. So this is the same one TTL value; this will be another TTL value I finish you will get roughly this will be another TTL value you will get. So you can keep on increasing your searching space one after another, so that is possible.

But usually, this will not be required because the routing table the nodal degree is pretty high. Usually, look at a node. A node will have many nodes as the children, so that it will be 1, 2 so,

for us, we are using 160 nodes. We are using 120 neighbours who will be there, so the query will essentially propagate very fast within a very few levels, so this tree because of a lot of partition if this is in addition to your neighbour tables.

So, ultimately, I will reach almost all nodes in a tiny number of hops. Only the TTL of even 128 or something will you be able to cover up the whole network, and queries will resolve. So this is how you can build up a peer to peer search engine. And next, we will look into anonymous peer to peer systems. So which basically means a toxchat, and then the blockchain concept or blockchain is actually using the similar peer to peer network, and then we will move on to the Tor Network, which is a complete enormity.

Because toxchat and blockchain do not provide your enmity, you can locate the guy to monitor, finding out the IP address. Even if you do not know their email ID or phone number, you can pinpoint the person or the building to derive information. We can also do traffic profiling to know who is active and who is not active, so sometimes that is also not desirable.

So that is talk when we are going to talk about your tor that is also constitutes something that we call Dark Net. It can provide a mechanism by which devices services can, or the components can talk to each other while remaining anonymous. So, people cannot figure out you cannot do traffic profiling. We will move to that in the next video.