Peer to Peer Networks

Professor Y. N. Singh

Department of Electrical Engineering

Indian Institute of Technology, Kanpur

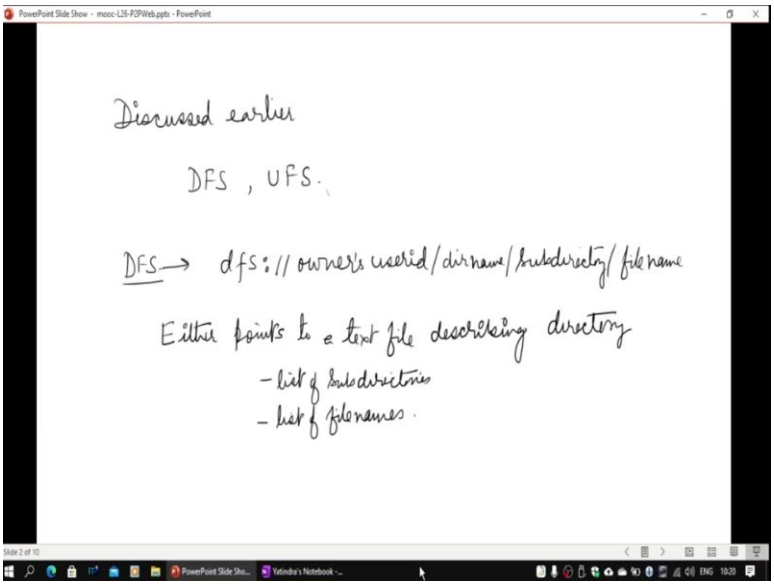
Lecture – 26

P2P Web: A Basic Design

(Refer Slide Time: 0:17)

So, welcome to the lecture video 26. This video will discuss inter-network, basically web, and World Wide Web kind of structure using peer to peer systems.

(Refer Slide Time: 0:35)



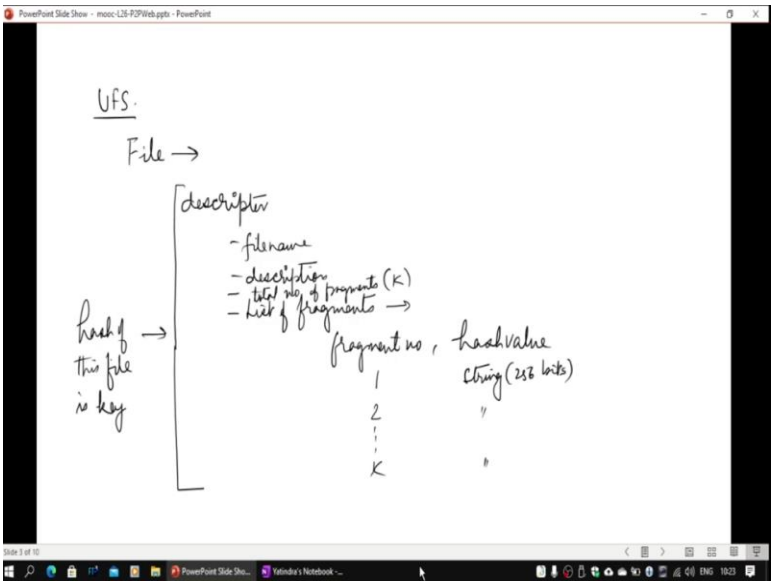
So, we have already discussed the distributed file system and universal file system both. As we have discussed in the distributed file system, we identified a URI by which any file in the owner or the user's namespace can be uniquely identified. We decided that we will be using DFS, which says the distributed file system access we are making.

And then we will put the owner's user-id, basically email id, because that is what we have been using Brihaspati-4. We can then put directory name, subdirectory name and maybe a file name also can be added. This can uniquely identify any file anywhere in the user's namespace in the user's file system.

So, the same file, the same name can be used by some other user also; because in that case, the string will always be unique and different. And of course, this is the key, which will be used to search for the content. This will always be pointing to a text file, which will be described in either a directory.

If it describes a directory, then this text file will contain what all subdirectories are existing or other file names existing. So, we can always pick up those subdirectory names or the file names and append them. And then create a new DFS query string, and then search for that particular subdirectory or file name. So, you can require the whole file system of the user.

(Refer Slide Time: 2:32)



So, we also had discussed UFS. The key I have given here, which we have discussed in the earlier slide, DFS colon slash user id, directory name, file name kind of things, and was in the user's namespace.

But actual files or content need not be stored with those, so we can create a universal file system, where its hash value is indexing content. And chances that two files, two fragments will have the same hash value, is very low. Even if it is there, we will know who has published it; if it is the users, there is some owner. We will typically go to a file whenever it is a file. We are talking about. It will contain or, in fact, even a directory I know file, which will have a list of subdirectories and files and if it is too large, it will go in the same fashion.

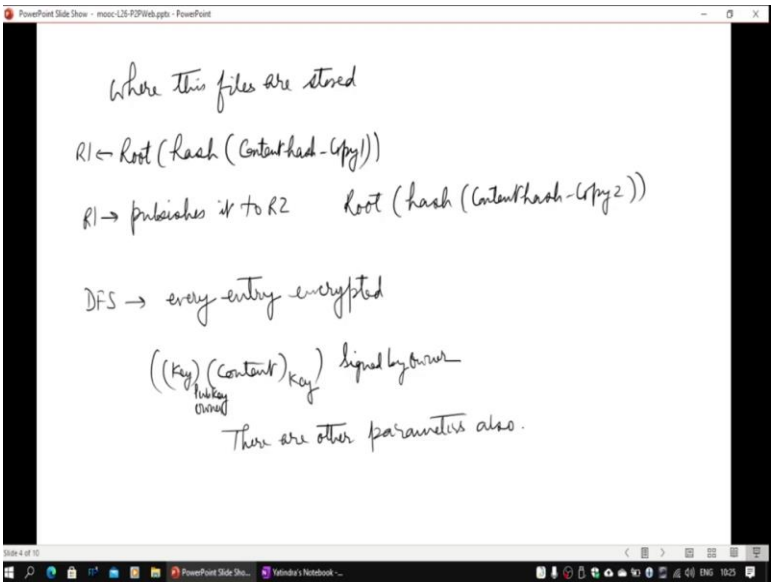
Usually, it will be maintaining that I will go to the descriptor of that particular file whenever I search for that key. I can also search by the hash of this descriptor file, and with that, I can also search; both ways, it should be possible. Whenever I am going to search for that key, the key will give me a hash value. And with that hash value, then I am going to search for the descriptor. Typically, the hash value will be used for files because I will figure out; who, whether this belongs to my namespace or not, has ownership.

Even that small finite probability, which exists for the same hash value being there for two files or two fragments, can need to be avoided. This descriptor will generally consist of the file name, description, and optional text description. And the total number of fragments which are there,

then we will not list of fragments for this particular file. With the K fragments, there will be K separate hash values; and this can be indexed by the hash of this particular file itself.

But, remember, there can be two of them, so this thing needs to be signed by the owner; so, we should know who is the owner in this case. And hash will also and their other options which also need to be added to this. I am not going there because I am just pointing it towards something I had discussed earlier.

(Refer Slide Time: 5:07)

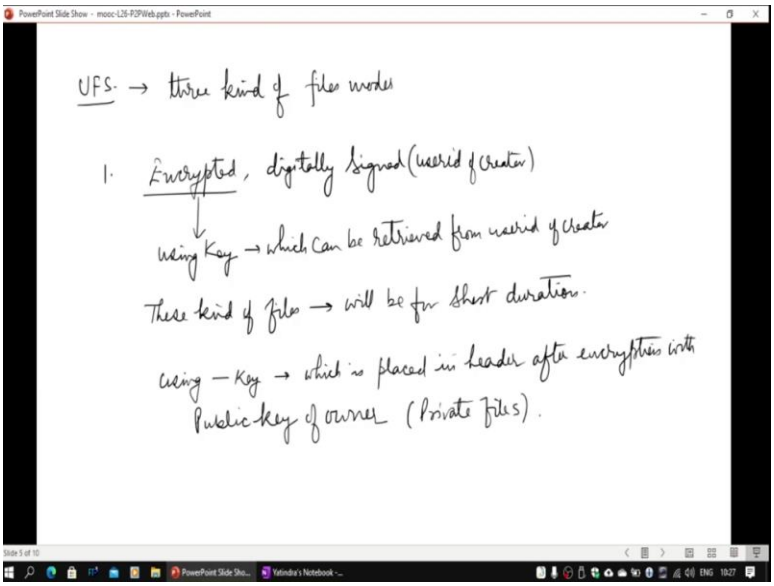


Now, where these files will be stored, that is the question; if I have the hash value. Usually, you will take the content hash, and you will append the string called copy 1. And then you compute the hash of that, and then the root of that will be R1, say, in the number of nodes that will be placed where you will be storing it. Now, since we are actually by design, copy 1 will create copy 2, so R1 will publish it to R2.

And it will do it by appending instead of copy1, replacing it with copy 2 and then computing the hash, and then finding out the root node. Now, in a DFS, when this is done, it is essential that every entry which is going to be stored has to be encrypted. Nobody else except the user because his file system spread around; nobody else should access it. So, usually, the content will be, for example, a key will encrypt this content, and that key itself will also be pre-appended, but the public key of the owner will encrypt this.

The owner will sign this whole thing, so the owner can always be sure. Or, anybody who reads is sure that this is not tampered with by anybody. And then, of course, since the owner can only decrypt this key; only he can read this content; nobody else can. And of course, there are many other parameters, so I am skipping them here; these we have discussed when we look at the distributed file system.

(Refer Slide Time: 6:54)



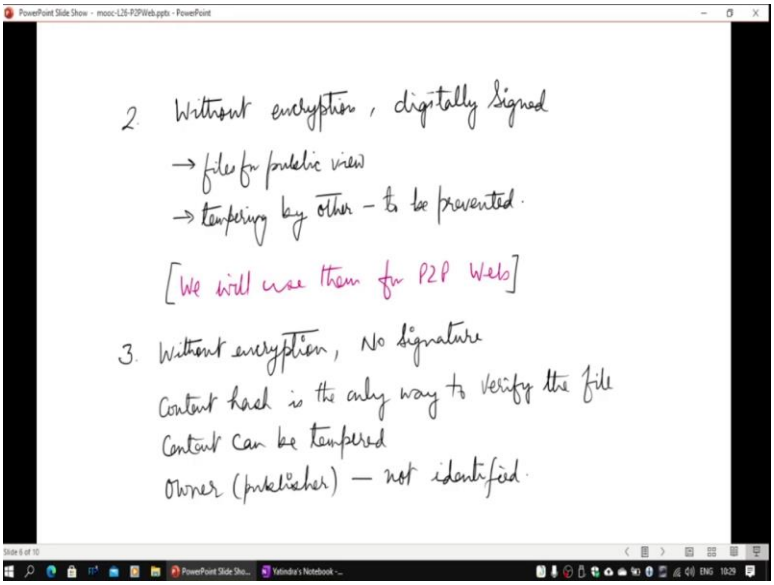
So, in the case of a UFS Universal File System, because that is what is the underlined thing, on top of it, I will create a DFS. Though, of course, we do it the other way around, but to keep uniformity and do as much as possible, we always create UFS. So, content any fragment, any data fragment is always indexed by its hash value; and then these hash values can be further, put inside a descriptor file; which then can be indexed by it is again hash value. But, that hash value is essentially linked with the DFS URI, or these can be directly linked with DFS URI itself; both are, both way it is feasible.

And the system will be flexible, can be made flexible to use both. So, now the three kinds of file nodes can exist in a universal file system; the files and whatever fragments are stored are encrypted. And it has to be digitally signed by the user-id of the creator. Now, usually, this is encrypted. It would help if you had to own decrypted using a key. You can only get it from the creator's user-id, who has created this and digitally signed this stuff.

Either you contact him and get the key for this content, or usually, this kind of file will be for a concise duration. For example, in the case of a list server, we have talked about such a file; or you can this key is also further encrypted and put it in front. And the key is encrypted with the owner's public key so that the only owner can retrieve this particular file.

So, this basically will be used for DFS part or list server or messaging kind of mechanisms. And interestingly, even if whatever be the hash, it is not that the root of the content hash, where it will be stored; it can be restored at some other place. The way it happens in the list server is stored at the list server, which will be the root of the hash of the list name colon user-id. And of course, you are adding some string copy 1 and copy 2.

(Refer Slide Time: 9:15)



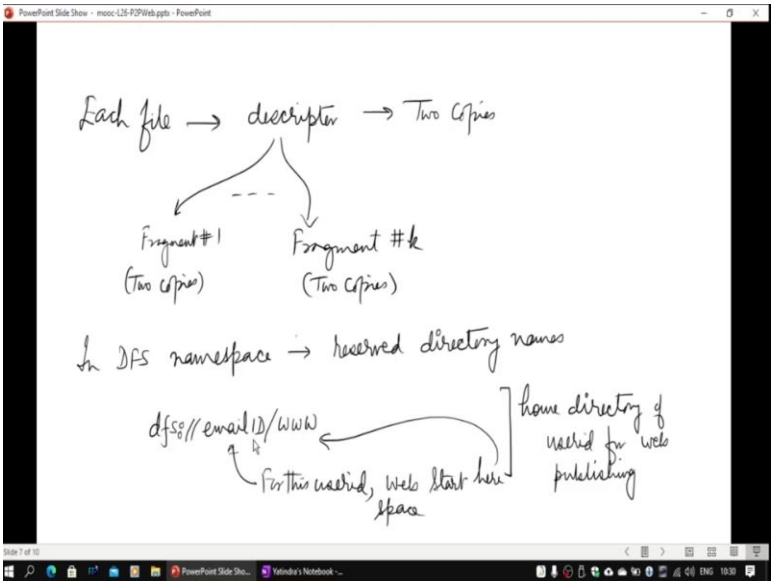
The second kind of files in the UFS system is that they will not be encrypted, but they will be digitally signed. These are essentially the files used when we are thinking of forming a peer to peer web. So, when we are with these files are without encryption, meaning what; anybody can use them, this is are actually for public view. So far, you know the URI; you can always find out the hash value and then from there, you can get this file. And these are digitally signed but not encrypted so that you can view them very clearly. Digital signatures are required so that you ensure that nobody has tampered it is; these are what the way user owner wanted them to be actually.

So, these are basically what we are going to use for peer to peer web; so that is our plan for the Brihaspati-4 system. And there is a third kind of files which we will without encryption, and there will be no signatures. So, like a movie file, nobody wants to own it, but it is dumped, so the content hash is the only way to verify the file. But, whether it is tampered with or not tampered

with, nobody can say anything. You can figure out if you tamper with the file content hash changes, and you can verify with the content hash. So, the publisher of this will not be an identifier, but tamper-proofing cannot be implemented.

People may use it, may not use it, it is only for distributing content without any liability; there can be things sitting inside it.

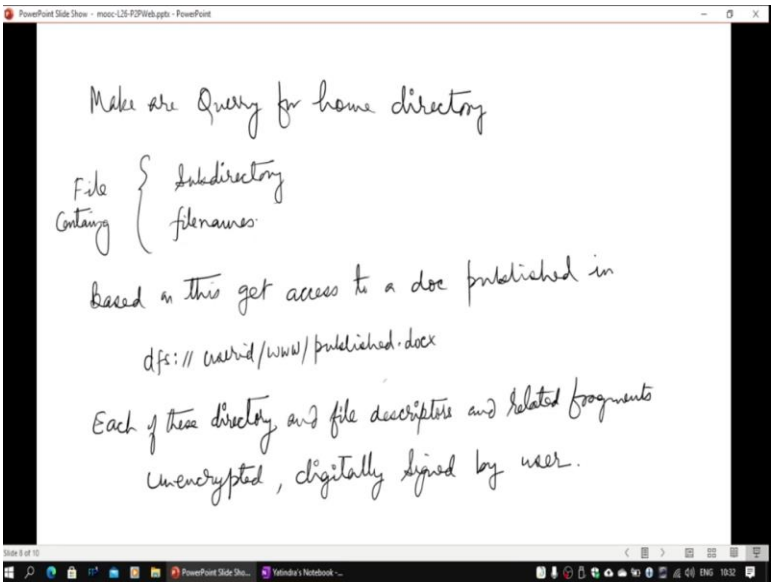
(Refer Slide Time: 11:01)



So, each file descriptor will have two copies; each of the fragment which has been defined says 1 to K, all of them will have two copies. And where these web files will be present? It will be there in the distributed file system. Now, what we do is we have reserved a name www here; so when you search for DFS:/email-id/www. That is the root of the web space of this particular email-id. So, this reserved directory name will be used as the user's home directory, who host whose email id is here to publish his content on the web.

.

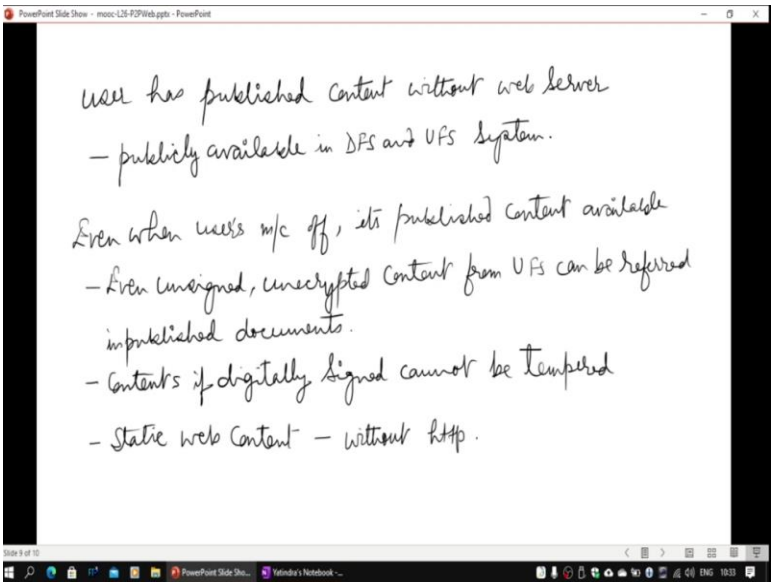
(Refer Slide Time: 11:51)



So, any browser and you can. You can start the full web access for browsing for a user; by querying its home directory. You will get a file, and this file will contain a list of subdirectories and file names, and then you can browse through the whole web space of a user. Remember, it is all publically available; it was intended to be that way, and you can access any published document in this fashion.

And remember each of these descriptors for the directories or file names and the fragments and everything; they will be unencrypted so that anybody can read. But, they all need to be digitally signed to ensure it is tamperproof. See, currently, when we access a web, we are going to a server; somebody owns that server, it has HTTPS. So, the server is the owner or a publisher; because I am always accessing a server. I am verifying the server, so I am assuming it is tamperproof; we usually create an HTTPS connection now, a secured channel on which the content comes back to the browser.

(Refer Slide Time: 13:24)



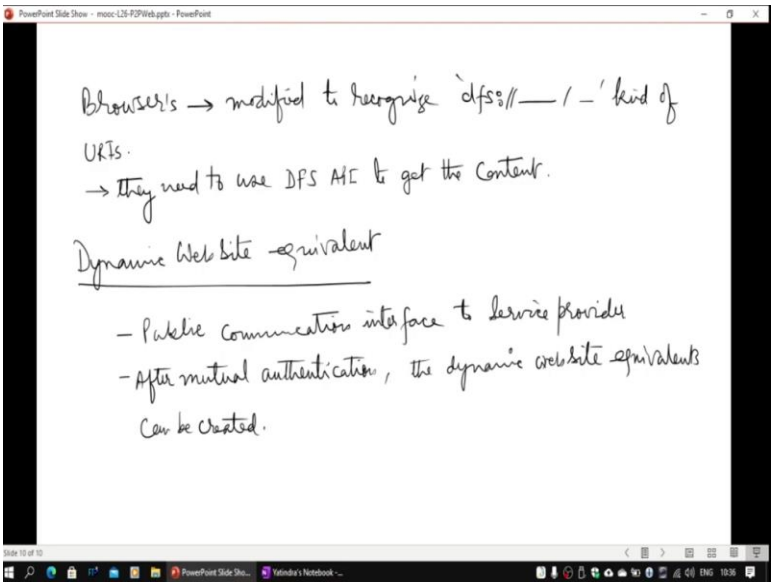
So, user, the important thing there is no Apache webserver which you have put for IIS kind of things where you are going to put up your content, you do not have to go to a server and ask for server space, for your web page. You can now create a web page on your peer to peer in your DFS system, and this is publically available via DFS and UFS combine. And when you are turned off, what happens to it; it does not matter because your DFS content is always there in the peer to peer web. And as machines turn on and off, the content moves around to whatever is a current root node for each fragment or file.

So, whether you are on or off your content, your public web view is always available to anybody. Like even your file system, but your file system is encrypted; only you can decrypt it. You can log in from your other machine and still fetch it, actually, and if you have the same user-id certificate, it is also being used there. You can always retrieve the content, so you also see your DFS view as the same view in all your machines. So, your user-id certificates remain the same, which somehow you have transferred across the machines, and of course, it is a tamperproof system.

But, now the problem arises here this is static web content; it is not dynamic. Usually, we use a lot of JavaScript code and do all kinds of animations, and you might like to build up an

interactive web page. Now, for that actually, we have to go. Further, we have not gone to that part of the design, but that can be incorporated now; that should not be a difficult thing.

(Refer Slide Time: 15:20)



So, but now to implement this, we need to modify the browsers, so browsers currently only understand FTP, SSH or something, so we have also to know this DFS colon slash user id. This protocol also needs to be added; the browser should interpret this kind of URI. So, you probably have to change a few very commonly available browsers; probably, everybody will build it over time.

And once it is done, your browser can browse through your, they will have the DFS API, and they can browse through peer to peer web, which is created on the DFS system. Remember that now web servers now you are working without web servers. They will be slightly slow; I expect it that way; but, it will still function beautifully. Now, how to create a dynamic website? So, you actually can have dynamic content also provision you can run a service.

You can then define an API; this API can be published, so when you give your static page, you can link to a dynamic API. Along with whatever code JavaScript code you have published, which can be executed on the client machine, and that client is now acting like a small application, use your API specifications to talk to your machine, which provides the service. But,

the moment you are turned off, nobody can provide the service; only you can provide to all that thing. So, usually, this will be done by companies or corporations to have multiple servers.

You can then connect to any of your JavaScript application running on your browser, which JavaScript application which you fetch from DFS; can run and can communicate to these small DHT layers been created by the companies. And can fetch the content dynamically and create a dynamic web view for you actually; and of course, mutual authentication, everything will be present in this case. This is how one can create a peer to peer web, which will be an exciting way of changing the internet.