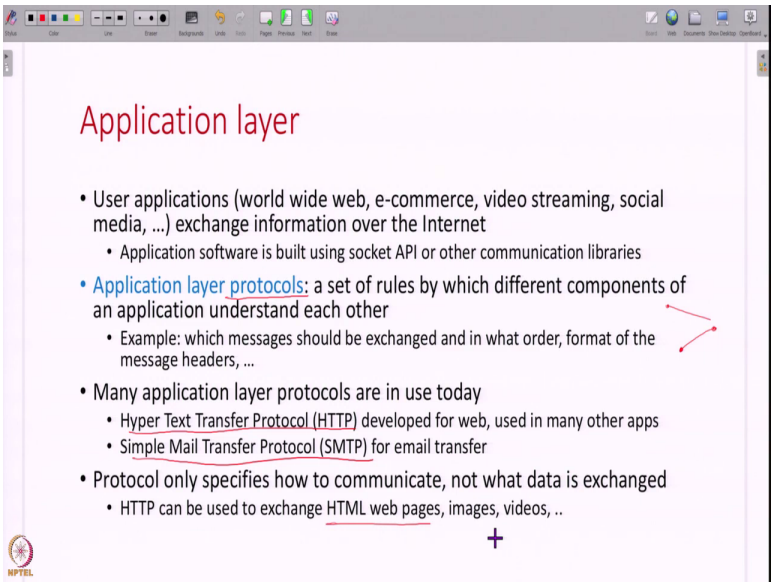
Design Engineering of Computer Systems
Professor. Mythili Vutukuru
Computer Science and Engineering
Indian Institute of Technology, Bombay
Lecture 34
Application Layer Protocols

Hello, everyone. Welcome to the twenty fourth lecture in the course Design and Engineering of Computer Systems. So, this week we have been discussing a few high level details about the various layers in the network stack that comprises the Internet. In this lecture, we are going to study more details about the application layer.

(Refer Slide Time: 00:48)



So, let us get started. So, the application layer is what all of us see and interact with. When we use the Internet, when we use the World Wide Web, browse websites use an e-commerce website or video streaming or social media or anything, all of these are user applications, which are present in the application layer of the Internet. And all these user applications exchange information with each other over the Internet. And this application software we have seen is built using APIs like the socket API, which lets us send and receive messages using the network.

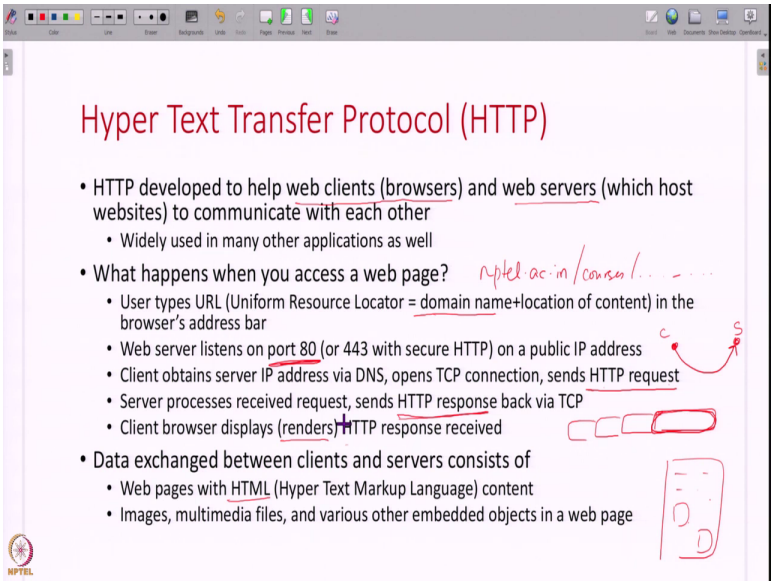
So, the application layer is defined by a set of protocols, which are nothing but rules by which the different components of the application understand each other. For example, if you want to book a ticket, then you will send a message with some information, then that server will send you back a reply with some other information. There has to be a certain format for all of these

messages. And the order in which these messages have to be exchanged, what is, in what order is information written into the packets, all of that has to be decided so that the both sides can understand each other.

So, application layer protocol is like a language that the both sides speak in order to understand each other and you need a common agreement, a common language to understand each other. So that is defined by application layer protocols. And we have many protocols in use today. The most popular one is HTTP or hypertext transfer protocol. This is developed for the web to browse websites, and of course, many other applications today. It was initially developed for the web, but many other applications use it.

Then you have the simple mail transfer protocol, SMTP, for email transfer, and there are many such protocols in use today. So, we will study some high level ideas about these protocols in this lecture. Note that the protocol only specifies how we should communicate. It specifies the language, but actually not what content is exchanged, what data is exchanged in that language. So, with HTTP, you can exchange HTML web pages, you can exchange images, videos, any such things. So, let us not confuse between the protocol itself and the actual data being exchanged by the protocol.

(Refer Slide Time: 03:04)



So, what is HTTP? So, HTTP is a protocol that helps web clients that is typically browsers to communicate with web servers, websites that actually host content, web pages, or videos, or the

NPTEL website that you are accessing, the YouTube website, all of these are served by web servers. And this content is served to web clients using HTTP. So, this is one of the most widely used applications on the Internet.

So, let us understand how HTTP works. Once again, go back to our example of what happens when you access a web page. The user will give a certain URL. This URL is nothing but it has the domain name like nptel dot ac dot in. And inside that domain your URL has other things also nptel dot ac dot in slash some courses, some lectures or something, something. It has the location of the content at that domain name also present in the URL.

So, the user will enter the URL in the browser. Then what will happen. Of course, your server that is listening on a well known port, 80 is the port that is used for HTTP or if it is secured HTTP then 443, whatever it is. On some port at this public IP address, your server is listening to new connections. And then your client will obtain the server's IP address via DNS and it will open a TCP connection to the server.

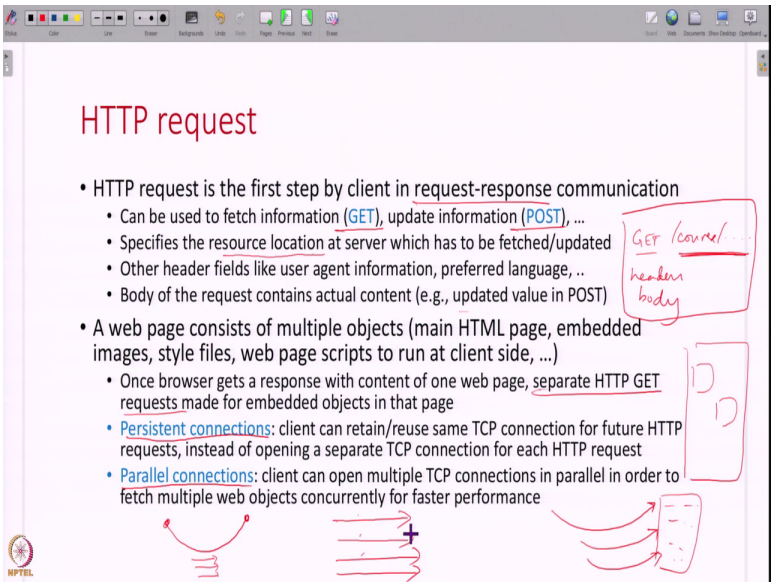
So, your server is there, your client will do the connect system call or whatever and establish a connection to the HTTP server that is listening on port 80 at some IP address. Now, the client will send an HTTP request. It will tell the server please give me some information about this web page or about this course or about this video, whatever. The client will request the server make an HTTP request and the server will send an HTTP response back.

And all of these HTTP requests, HTTP response are just application layer messages that are sent say via TCP sockets and on top of that you have your HTTP request. And then the TCP will add its headers, IP will add its headers, the layer two will add its address, all of that is happening. But the actual message is an HTTP request. The format of this actual message is specified by the HTTP protocol.

And once you get some response back from the server, then your browser will display whatever information is there in that HTTP response, whatever HTML display the page like this, all that information is there in your response, and the browser will render that web page in front of you so that you can see. This is a basic high level view of what happens when you browse a web page.

And this data that is exchanged, in HTTP response whatever the server sends back can be actual web pages, text, like HTML based web pages, it can also be images, multimedia files. On one web page, you can actually have many different things. You can have the HTML webpage, you can have some style files telling how this web page should be displayed. You can have some embedded images, videos, all of that will be sent to you in the HTTP response. So, this is the basics of HTTP.

(Refer Slide Time: 06:34)



Now, let us look at what is the HTTP request that the client sends to the server. The HTTP request is the first step. So, HTTP is a request response communication. The request is the first step. So, the request can be used to get information. The requests are multiple types. It can be a get request. It can be used to update information at the server. For example, if you are booking some train tickets, you can post information to the server saying these are my dates and this is my route and this place I want to go to all of that you can post to the server. So, your HTTP request can be get, post, and there are many such types like this.

And then you can specify the location, the resource location, I want to get information about this course or something or this product on an e-commerce website. So your get request will also contain some extra information about which resource you want to get. And your HTTP request will also have various other HTTP has many header fields. Some extra information it adds to the message. This is the main message get so and so information. But you also add what type is your

browser, what language do you prefer. There are all of these extra things also you will tell the server. And all of this is sent in the application layer message.

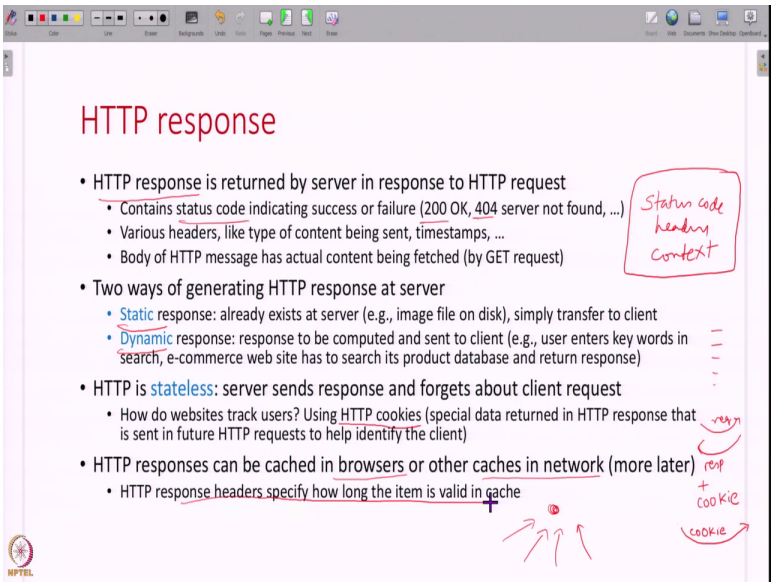
In addition to this, you might also have some, even more content. The body of your request might contain whatever value you want to post. You have get, put, post all of these things, and then you have some headers, and you have some body. All of this is your application layer message, your HTTP request.

And at what granularity do you make HTTP requests. If you have multiple objects in a web page, you have the HTML text, you have some image, you have some video, you have some other script files, various other things in a web page, all of that information cannot be fetched in one HTTP request. Every object on a web page is fetched via a separate HTTP get request. So, your browser will make one get request, get the main page, and inside that page, whatever other things are there, you have to make separate HTTP get requests to get each of these separate items. So, an HTTP get request will only get you one item. An HTTP post request will only update one item for you.

Now, these multiple HTTP get request, what do you have to do? Do you have to open separate sockets. It is not necessary. If you already have one socket open between a client and a server, you can get one object, after that you can get another object, after that you can get another object. So, you can use what are called persistent connections. Over the same TCP connection, you can first get one object, then the other object, then the other object, one after the other. Instead of every time opening a separate TCP connection, doing a TCP handshake, connect, accept, you do not have to do that every time. Once you open a connection, you can fetch multiple HTTP objects using separate HTTP request on the same TCP connection.

The other thing you can do is you can also open parallel connections. To one web server you can open multiple TCP connections. And on this get one object, on this get one object, on this get one object, in parallel, you can do this also for faster performance. If your webpage has 20 objects, you can, in parallel, open four connections, get sort of five objects each on each connection. You can do that also. So, this is about HTTP request.

(Refer Slide Time: 10:20)



And next is HTTP response. Once the server accepts your TCP connection, reads your HTTP request, we have seen a simple server socket program accept connection, then read from the socket, and now the client has sent an HTTP request on the socket, then the server will generate an HTTP response back to the client. This response can, will have various things. It will have a status code indicating could the server successfully handle your request or not. The status code, a common status code is 200, saying, then there is a status code 404 saying you could not contact the server something. There are various status codes that are returned for you in your HTTP response.

Then your response also has various header fields like what is the type of content you are sending, some timestamp, when was the content updated, all of that also you will send and the actual content is also sent. You will send a status code and various headers and then the actual content all of this. So, the HTTP protocol specifies how all of this should be put into a packet and sent out to the client.

And note that there are two ways of generating HTTP response either all of this, whatever content you want to send this file, the user has requested an image file, say, either the content can already exist at the server, in which case it is a static response or the server may have to generate this content for each request. For example, if the user has entered some search keywords on an e-commerce website, then with those keywords you may not have a readymade file that you can

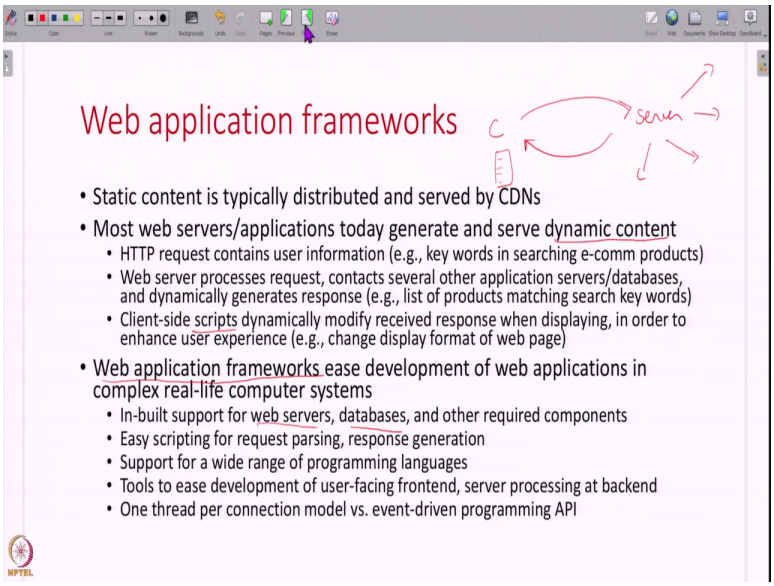
send back to the client. Then the server will look up its database, construct an HTTP response, and then send it back to the client. Those are called dynamic responses.

The one thing to remember about HTTP is that it is stateless. That is request response after that that is it. The server does not remember anything about the client. It would not remember last week you made this request to me, I sent you this response, now I am sending you this respond, does not remember any such thing. HTTP is fully stateless. But then you might be wondering, some websites track you, some websites know what I have done before on that website. How is that accomplished? That is accomplished through what are called HTTP cookies.

So, when you make a request, the server will send a response back. Along with that response, it will also send some special identifier for this user called a cookie. The next time you make a request, if you send this cookie to the server, then the server knows who you are. And it can remember some information about you. If you clear all your cookies, then the server cannot recollect any information about you. So, if you want more privacy, you can always disable cookies on the Internet. And these HTTP responses they can always be cached.

You have some image file you got, then some other user also is going to the same server getting the same HTTP response, then again this idea we have seen again and again in the course you can also cache this HTTP response. Your browsers maintain a cache or there are some special caches in the network, so that if multiple users in a network are accessing the same HTTP content, you can obtain it from the cache instead of going to the server. All of these also can be done. And the HTTP response headers will tell you how long the item is valid in cache and how can you cache it that is specified in the HTTP response headers.

(Refer Slide Time: 14:05)



So, now, most of the web servers today actually generate dynamic content. Whatever static content is there, if you are distributing static content, you can just give it off to CDNs. We have seen CDNs in a previous lecture. But most application servers, web servers today are focused on dynamic content, because that is what is where the most interesting computer systems can be built. For example, how is dynamic content served?

If you want to search some, for some products in an e-commerce website, then you will enter some keywords. Then your HTTP request will convey all of these keywords to the server. Then the server will look at the keywords. Contact other databases or other servers. You made some requests to an e-commerce website. This server will talk to various other components, look up some catalog of products, assemble the response and then send the response back to the client. So, there are many steps here.

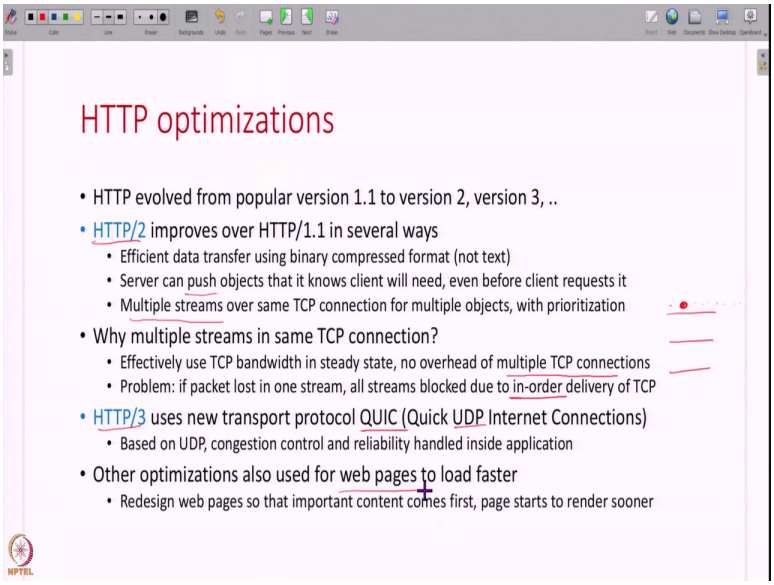
And at the client side also, there could be some scripts running, which basically modify this response when displaying. Suppose the server has sent a list of 100 products, then on your browser, you can have an option to view only the first 10 products that is done by your client side. Some scripts running at your browser that will do all of this, like for example, Java scripts, and all of that. So, in this way, a dynamic content is generated both at the server side and dynamically displayed at the client site.

Now, this is a very complicated thing. It is not like a simple socket program you can write, read something from a socket, and write something back. It is not so easy. You have to talk to multiple components, assemble the response, dynamically construct the response. Therefore, you have various pieces of software today, which are called web application frameworks, that will make it easy for you to build these web servers or application servers that serve dynamic content.

These frameworks will have all the pieces required, the web servers, the databases, and some scripting languages in order to parse the request, construct the response in a wide range of programming languages, some front end tools at the client side, the scripts that you need to display this dynamic web page, the server side tools that are needed to handle all this back end processing.

All of these are packaged together in a web application framework. And you use this web application framework. You do not have to worry about setting up your database, web server or opening a socket. All of that is automatically handled for you. You can just focus on your application logic of how to handle HTTP requests, what things to fetch from the database. You can just focus on your application specific logic. Everything else is taken care of by the web application framework. Even things like should you have one thread per connection, event driven APIs. All of these are also handled by the web application frameworks.

(Refer Slide Time: 17:08)



So, HTTP also has many optimizations today. You have version 1.1 that is evolving to version 2 and version 3 today, which has many features in order to improve performance. For example, HTTP/2 improves performance in several ways over HTTP/1. You have a more efficient way of transferring data. That is HTTP/1.1 send data as text, but HTTP/2 will send it in a much better compressed format.

Then the server can actually push some objects instead of waiting for the client to ask for everything. If the server knows the client needs something it can push. You can also have multiple streams over the same TCP connection so that you can quickly send multiple objects. All of these are optimizations that you have in HTTP/2. If you upgrade your web server to HTTP/2, you will get all of these features for improved performance.

So, you might wonder, why do you need multiple streams in the same TCP connection? That is because once your TCP has done the slow starts, settled into some constant bandwidth into some stable state, you might want to actually send multiple streams of data into that TCP connection, get multiple objects in parallel, instead of opening multiple TCP connections itself. In one connection itself, you might want to get multiple objects. That feature is available in HTTP/2 for you.

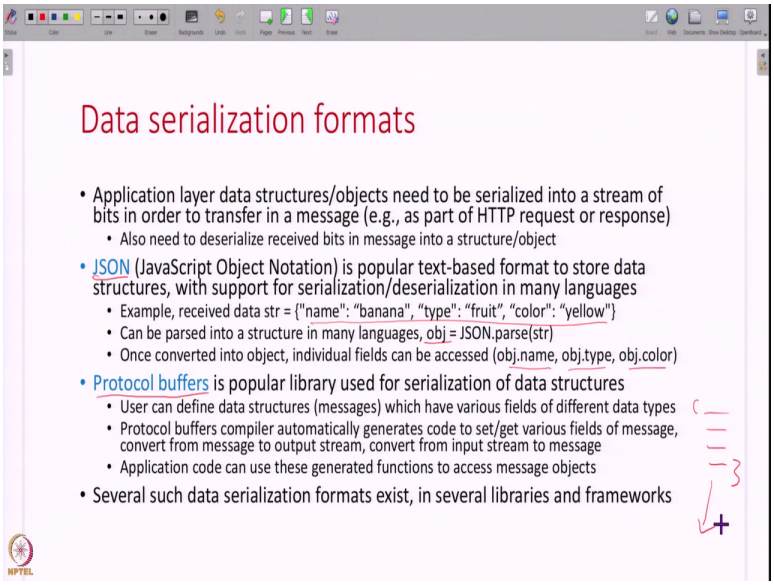
Of course, this has the problem that what if one of the objects is lost, then all the objects in a stream are kind of blocked, because TCP will recover, do retransmission of that object. And even if the other objects have gotten through, they cannot be delivered in TCP because it only does in-order delivery.

Therefore, people are also moving away from TCP towards UDP based transport protocols. Now, HTTP/3 uses a new transport protocol that is called QUIC, which is actually built up on UDP. And you add reliability and congestion control a little bit differently. You are not doing like TCP, but you are doing something different. You are starting from UDP and doing something different. So, HTTP/3 uses QUIC as your transport protocol.

So, in this way, I mean, all of these are sort of advanced topics that we do not have time to discuss in a lot of detail. But one thing to understand is people are constantly working towards how to make HTTP transfers faster, and newer versions of HTTP are evolving to handle all of these issues.

And there are also work being done on how to design your web page itself so that it loads faster, so that the important content comes first, you start to see something on the webpage so that the user is not staring at a blank screen for a long time. All of these are active areas of research on how to improve applications to make them faster for the Internet.

(Refer Slide Time: 20:00)



So, one other thing that you would have come across is, things like JSON and all of that you would have heard. So, I just want to briefly explain what these are. So, whenever you are sending some big data structures or objects on the Internet over protocols like HTTP, you have to serialize them. What does it mean? You have to somehow have a way for deciding how you will break up that object and stream it as a series of bytes. So, there are certain standardized formats in order to serialize objects and deserialize them at the other end.

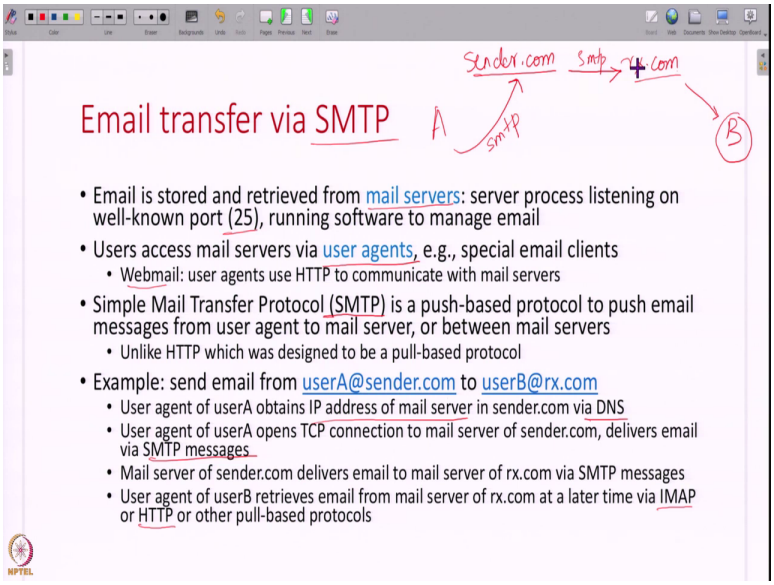
So, one popular format is what is called JSON. So, this is a text based format. So, if you have some data structure like this, for example, a banana is represented as an object. It has a name, a type and a color, and so on. Then JSON will basically serialize that object into the string and send it over the network. And when you receive the string, you can once again parse it and make it into an object. Then you can access the fields of the object like this. So, you can easily go from an object, a data structure to a string, and back, and different programming languages will support this. So, this is a standard way.

Note that you need to agree on the standard format of the string. Otherwise, if different people are serializing and deserializing in different ways, then you cannot recover the object on the other side. The other popular format that is being used today is what is called protocol buffers. This is another library that can be used to serialize.

So, with protocol buffers, you can actually define some message. You can define a structure with all the fields, data types, everything you can define. And then this protocol buffer will automatically generate code that will convert this object into a string, convert a string into the object, modify various fields of the object for all of those. The cord is automatically generated that you can use, that the application can use.

So, you can just create a object like this and give it to this compiled code that will take care of converting this object into an output byte string. So, there are several such serialization formats available so that you as an application developer, you do not have to reinvent the wheel every time and figure out my application has this big data structure, how do I send it to the other side, how do I write it into a socket. All of that is taken care of if you use one of these standard data serialization formats.

(Refer Slide Time: 22:33)



So, the next application that we are going to study is how email works. We have another application layer protocol SMTP that deals with email. So, email is stored in mail servers, like how websites are stored in web servers, email is stored in email servers. And this is again a

server has nothing but separate process that is listening on a special well known port. For email it is 25, whereas for web it is port 80. And this is some software that is listening on this port and can manage all the email that you get. So, these mail servers like companies like Gmail or your own college, every organization can have its own mail servers available to store the email.

Now, when the user accesses the mail servers, the user will use something called a user agent, for example, an email client or the browser. If your browser is your user agent, it is called webmail. Or you can also have other special user agents like Outlook or Thunderbird. These user agents talk to a mail server in order to send and receive emails. And the protocol that is run between these user agents and the mail server, that is called the SMTP protocol. Now, why do you need a separate protocol? For example, why could not we use HTTP?

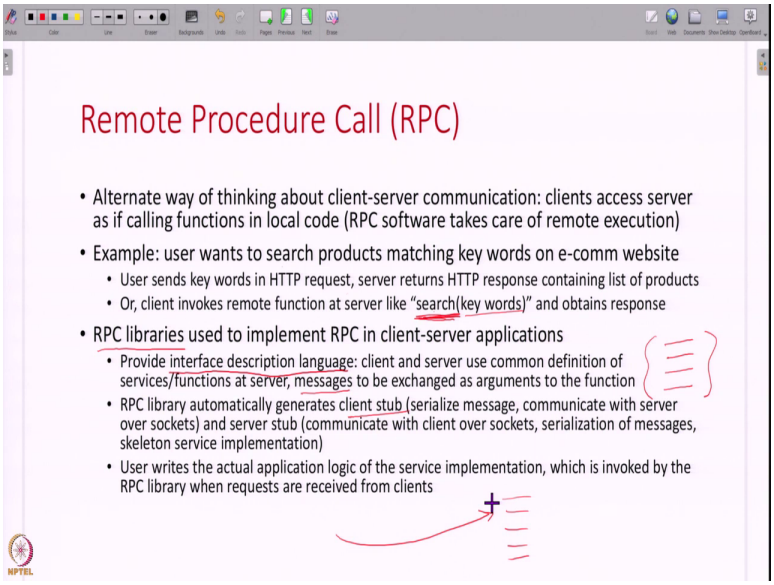
Note that different protocols are designed for different purposes. When you send an email, you have to push an email. You are not fetching anything. HTTP is designed to get information from a website, whereas SMTP is designed to push emails to the other person. Therefore, the semantics, the headers, everything are slightly different. Therefore, you have different application layer protocols for different purposes. So, let us just understand, if you want to send email from this email id to this email id what is happening. You have a user A that is using some email server at sender dot com to send email to somebody at rx dot com to another user B.

So, then the user agent of A which is either a browser or some email client will first obtain the IP address of this mail server at sender dot com. How do you do that? That also you do via DNS. DNS will give you the IP address of the website. It will also give you the IP address of the email server. If you go to gmail dot com, gmail dot com has a website. It also has the mail server that is actually handling all of this email.

So, from DNS you can get the IP address of the mail server also. Then you will open a TCP connection to that mail server. And over that TCP connection, you will send SMTP messages to push your email. You will use SMTP to push your email to the mail server sender dot com. Then sender dot com will open a TCP connection send an SMTP message to push your email to the mail server at the receiver side. Now, then this user at a later point of time this user will fetch the email, will contact the mail server at his the receivers domain and get the email.

Note that on the receiver side, you cannot use SMTP because here you have to pull. So, therefore, there are various protocols available like IMAP, and so on or you can also use HTTP to pull your email. This is like getting information from a website. The concept is similar. So, the user agent of B will pull email. But all of this pushing of email from user agent of A to the mail server here between mail servers, all of this happens via SMTP.

(Refer Slide Time: 26:09)



So, the one final concept I want to discuss in this lecture is that of what is called remote procedure calls or RPC. So, RPC is just a different way of thinking about client server communication, where you actually call the server code as if it is a function in your local code. For example, if you want to search for products on an e-commerce website, let us continue the same example, either you can send your HTTP keywords that you are searching for an HTTP request, then the server returns an HTTP response and so on that is one way of doing it.

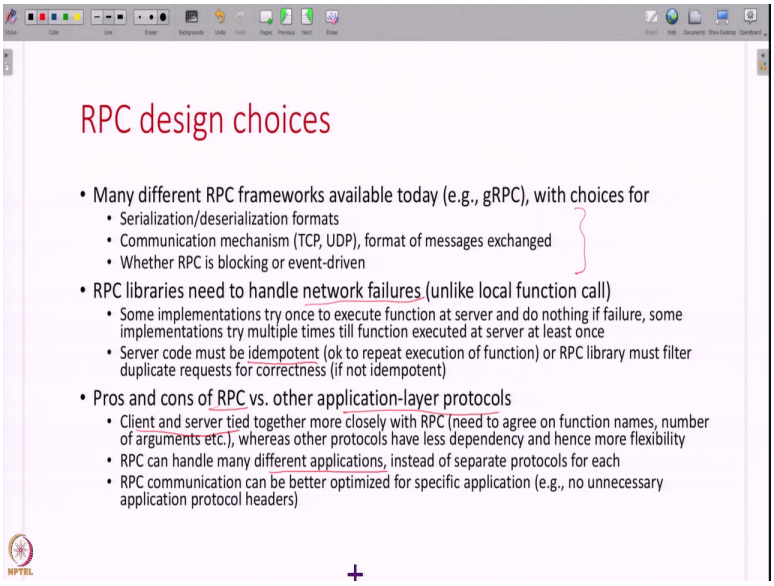
The other way is if the server has some function defined in its code like search, you can just invoke this function at the server. You can invoke a function at a remote server like just like you would call a local function in your code. That is another way of programming applications, which is called remote procedure call or RPC. So, now, you might wonder, in my code, I know what functions are there. How do I know on the server's code what functions are there? That is where you have various RPC libraries or RPC frameworks that will help you implement RPC in client server applications.

For example, one thing that you need is you need a common description of the interface. The client needs to know what are all the functions that the server has, what are the messages it can send, what are the arguments to these functions all of that information it is written in a common interface description language and agreed upon between the client and the server. Then when the client calls this function, like search, the client knows now from this interface description language it knows what are the functions at the server. Then when the client invokes this function like this, this RPC library takes care of everything.

This message that is sent to the server, you will serialize the message. On the client side there is a small stub that will serialize this message, open some socket, talk to the server. On the server side, receiving this message over sockets, converting it into the arguments for the function all of that is done by the RPC library. And you as the user, you just have to write, implement the functions at the server side and invoke those functions from the client side.

So, RPC is a very different way of programming. In some sense, it is more intuitive, where you are able to decide, just write your application in the form of functions and invoke those functions just like how you would do in a regular program with the RPC library, taking care of all this remote execution related complications.

(Refer Slide Time: 28:52)



And there are many RPC frameworks available today, like gRPC is a popular framework. And we are very choices for how do you serialize, deserialize, how do you communicate TCP, UDP,

how to exchange messages, do you do a blocking RPC, like when you invoke a function will the client block, is it event driven API, all of these choices are available to you in different RPC libraries. And the RPC library will also take care of network failures.

Now, when you do a local function call, there is no notion of packets getting lost in all. But when you make a request to a remote server, actually, a request can get lost. The server may not get that request. So, therefore, your RPC client has to repeat, retransmit that request, all of that is also taken care of by RPC libraries. Some libraries will repeatedly execute the function at the server. And your server code, for that to happen, your server code has to have some property called as it has to be idempotent. That is, you have to be able to repeat the function multiple times or the RPC library has have to somehow take care of that it will exactly invoke the function only once, all of these.

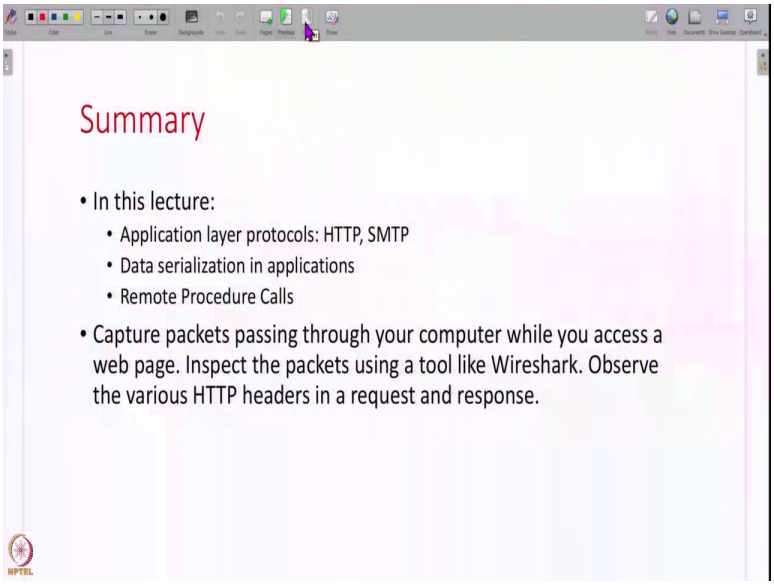
We will study all of this when we study reliability part of our course in more detail. But the thing to note is that your RPC library has to handle many more complications. RPC is not as simple as just running your local function because all of these other complications that are present. And then why would you use RPC versus using an application layer protocol? For example, if you want to talk to get some information, will you use RPC or will you use HTTP?

Well, the choice depends. So, with RPC, the problem is that your client and server are tied together very closely. You cannot just open a connection to any server and request HTTP objects. You have to know what are the various functions being implemented at the server function names, number of arguments, what messages, what arguments, all of that is very closely agreed upon between the client and the server. Therefore, you have lesser flexibility, whereas if you use protocols like HTTP you have more flexibility. And any client can talk to any server. You are not in some sense coupled with the server.

But that said, RPC is very versatile. You can handle many different types of applications. You do not need one protocol for web, one protocol for email. You can do everything with RPC. So, in that sense, it is very powerful. And this RPC communication can also be better optimized. For example, in HTTP, you have so many headers, because it is very general. Anybody can use HTTP for anything. But with RPC you can actually customize what messages are exchanged, what information is exchanged for your specific purpose.

Therefore, it can be optimized better, it is more specialized, but it is also less flexible than application layer protocols. So, we will revisit this discussion on RPC versus HTTP later on in the course also when we are thinking about end to end application design.

(Refer Slide Time: 32:05)



So, I would like to wind up this lecture on the application layer here. In this lecture, we have covered some popular applications like HTTP, SMTP, and so on. We have studied how applications can serialize data to send over the network and we have also studied the concept of remote procedure calls.

So, in order to understand this better to actually see the various headers of these application layer protocols, I request you all to please use Wireshark to capture packets coming in and out of your computer, say when you are browsing the web or sending email and actually see what are the various headers in HTTP, what is the packet format in HTTP, all of that you can inspect in Wireshark. So, that is all I have for this lecture. Thank you all and see you in the next lecture.