CS 240: WebSocket Introduction Transcript

[00:00:00] In this video, we're going to learn about web sockets, which is uh another networking protocol that um clients and servers can use to, to communicate with each other.

Start visual description. The professor demonstrates the basics of WebSocket, explaining how it is a networking protocol that allows clients and servers to communicate with each other. End visual description.

[00:00:10] Up to this point in the class, we've been using the HTTP protocol for network communication between client server and webs. So, it helps solve some uh weaknesses with the http protocol for certain applications.

[00:00:24] And of course, we're going to apply webs sot in the chess project as well.

[00:00:31] So let's start off by reviewing a little bit about HTTP and understanding why WebSocket is necessary.

[00:00:39] Um Th the first thing to emphasize with HTTP is that it is a, a client server protocol.

[00:00:45] It's a request response protocol. So, what that means is that when a client wants to communicate with the server, the client must initiate the communication.

Start visual description. The professor illustrates the request-response nature of HTTP, showing how a client must initiate communication with the server by sending an HTTP request, which the server then processes and responds to. End visual description.

[00:00:56] And when the client wants to send a request to the server, it establishes a connection to the server, sends an HTTP request to the server and then the server processes that request and then responds with an HTTP response.

[00:01:09] So the thing to notice here is that all communication must be initiated by the client.

[00:01:14] There's no way for the server to initiate the communication back to the client.

[00:01:19] And uh in a lot of applications that capability for either side to initiate the communication really is, is necessary and important.

[00:01:28] Another thing about HTTP is that it does have a particular message format that that must be complied with.

[00:01:37] So we've learned about uh the textual messages that, that make up requests and responses.

[00:01:42] In http, the, the first line of a request has the, the um http method or request type, get put post or delete.

[00:01:51] Then after that you have a URL path and so on.

[00:01:55] And then after the, the top line of the request, you have um headers, any number of uh headers.

[00:02:02] And then of course, you can have optionally you can have a request body and of course, uh http response looks very similar to a request.

[00:02:11] And so there's this this envelope, this http envelope that that kind of encapsulates the requests and responses and has to um follow a particular format and that will come into play as well.

[00:02:26] So as an example here, we can see an http client calling trying to call a web API on an HTTP server.

[00:02:33] So um the client's trying to retrieve some scores back from the server.

[00:02:39] And so you can see in this, this diagram, the client would initiate the, the communication by creating connection and it sends a get request to the server with the URL pass slash scores and then the server responds with a response that the body of which contains a JSON string containing the requested scores.

[00:03:00] And so that's a typical client server communication with the http protocol.

Start visual description. The professor shows an example of HTTP client-server communication, where the client retrieves scores from the server by sending a GET request and receiving a JSON response. End visual description.

[00:03:07] So this protocol was originally designed so that people could browse the web.

[00:03:11] So it was really designed for retrieving resources or retrieving documents, other kinds of material from remote servers.

[00:03:21] And so you can imagine a web browser every time you're browsing the web, anytime you follow a web link, it's the browser which is the client makes a request to the a web server and retrieves the document um whose link you clicked on and so on.

[00:03:35] And so http really is good for that kind of uh transactional um request response needed to, to retrieve a resource from a remote server.

[00:03:46] What this protocol wasn't designed for was to do what's called peer to peer communication, which is um two or more programs on running on different computers that need to communicate with each other. But none of them is the server and none of them is the client.

[00:04:03] They all kinds of uh operate in both modes.

[00:04:07] Um Any, any one of those peers could initiate a request and send it to the other peers. And of course, any peer could also behave like a server in the sense that it receives requests.

[00:04:19] And so in a peer-to-peer situation, it's not really one side is the client, one side's the server.

[00:04:26] It's just a bunch of different nodes on a network, bunch of different peers on the network trying to communicate with each other in more of an ad hoc fashion.

[00:04:35] So HTTP is not really built for that.

[00:04:38] Yeah.

[00:04:40] So for example, if you wanted to build a chat application, so imagine people can enter chat rooms and message each other in a, in a chat room.

[00:04:51] Of course, you'd have a server in that case, which is kind of the center point of the whole application.

[00:04:56] And then you'd you could have clients that, that connect to the server.

[00:05:00] And then when one client wants to send a message to the group, the client would send a request to the server that contains the message that they want to share with everyone.

Start visual description. The professor explains the limitations of HTTP for peer-to-peer communication, using a chat application as an example to show how HTTP is inefficient for broadcasting messages from the server to multiple clients. End visual description.

[00:05:10] And then the server would need to then broadcast that message back out to all the other clients that are parti participating in the chat.

[00:05:17] And the problem we run into here is that while it, it's, it's easy for a client to send a message to the server using HTTP, there's really no way with, with regular HTTP for the server to then broadcast that message out to everyone who needs to receive it.

[00:05:33] Because in in an HTTP uh scenario, the clients have to initiate the, the communication.

[00:05:40] And so in this scenario, the server needs to initiate communication so that it can send the message to all the other clients.

[00:05:47] And so doesn't really work out very well. That way.

[00:05:52] Now, in the past, people have been creative and come up with solutions so that you could build a system like a chat system with regular HTTP.

[00:06:02] But it was uh none of those solutions were efficient or, or really elegant. Uh Very much so historically, if you needed to do something like a chat, what, what you would do is all the clients that are participating in the communication would pull the server every few seconds or so and ask the server if it had any messages for them.

[00:06:29] And so in this slide here, you can see that the client is, is periodically pulling the server sending it or a message asking, do you have anything for me? And then the server typically responds with no, I've got nothing for you.

[00:06:44] And then the client would continually pull the server in that way.

[00:06:47] And then every once in a great while the server would say yes, I actually have something for you.

[00:06:51] Somebody sent a message that you need to receive and then the server would return the message back to the client.

[00:06:58] Now, this, this would work, but it's not very efficient because the clients have to sit there and and pull the server uh every few seconds or even more.

[00:07:07] And it really puts a lot of load on the server to, to uh receive all those pulls from the clients.

[00:07:12] And so really didn't, you know, it, it works.

[00:07:15] But it, it, it's a very heavyweight solution, especially on the server.

[00:07:20] And so uh that while people do this, it uh it's not ideal.

[00:07:26] Another thing people tried with HTTP is they would actually um send an HTTP request to the server.

[00:07:36] Of course, that involves a connection being established between the client and the server.

[00:07:40] Now, typically after the client and the server exchange a request and response, the client drops the connection and closes it.

[00:07:48] But what people uh tried in this case was they would, they would create a connection to the server, they would send it an HTTP request, but then the client would hold the connection open indefinitely.

[00:07:59] It wouldn't close the connection.

[00:08:01] So it would just kind of hold that, that connection to the server open.

[00:08:04] And then at some point in the future, when the server needed to send a message back to the client, it could do so over the connection that was held open previously.

[00:08:16] Now this this can be made to work as well.

[00:08:18] But again, it's not really what HTTP was designed to do.

[00:08:21] It's, it's not efficient, it's kind of a heavyweight solution.

[00:08:24] And, and so um that's not ideal either. And so, after people tried these kinds of solutions with HTTP, um back in 2011, uh people designed a new protocol called web socket.

[00:08:39] That was, that was designed specifically for doing peer to peer communication.

Start visual description. The professor introduces the WebSocket protocol, designed specifically for peer-to-peer communication, and explains how it allows for asynchronous communication where either side can initiate the connection. End visual description.

[00:08:44] So a lot of systems use http in addition to web socket uh for different communication tasks.

[00:08:51] So HTTP is often used to call web API S and receive results from those.

[00:08:57] And then the web socket protocol is, is used to do peer to peer communication, that's more asynchronous and in which either side of the, the connection might need to initiate the communication.

[00:09:10] Now, in this slide here, it shows that there's only one Http client, but in general, there could be a whole bunch of clients uh needing to interact with each other.

[00:09:19] Like we talked about the chat application has many clients typically or in in the chess project.

[00:09:25] Um You could, you have at least uh you have two people playing a game, so there's at least two clients and then you may have any number of other people watching the game. And so, um just keep in mind that uh there can be any

number of clients, even a large number of clients that need to be communicated with here.

[00:09:43] Ok.

[00:09:44] So HTTP was, was inefficient for the kinds of things people were trying to do with it.

[00:09:49] There's a lot of overhead in HTTP actually because the HTTP message format requires certain things to be included.

[00:09:57] And it's not necessarily the fastest format to parse and things like that. So, there's a certain amount of overhead in the HTTP protocol that makes it less, less than ideal efficiency wise in some cases.

[00:10:10] And that's usually not a problem unless you really need um super-efficient communication.

[00:10:15] Uh For example, maybe in a video game or something when uh multiple players are playing against each other, there's a lot of communication that needs to happen between those different players in order to keep their, each copy of the game in sync with each other so that they feel like they're, they're playing in the same world.

[00:10:32] And as we've, we've pointed out, um the client always has to initiate communication and, and that's not, um that doesn't fit well with, with a lot of things we want to do these days.