# Name attributes of http protocol that makes it difficult to use for real time systems

Large request headers.

Response could be “old” when sent back to the client.

Half duplex.

# Explain polling and long-polling strategies, their pros and cons

With *polling*, the browser sends HTTP requests at regular intervals and immediately receives a response. This technique was the first attempt for the browser to deliver real-time information. Obviously, this is a good solution if the exact interval of message delivery is known, because you can synchronize the client request to occur only when information is available on the server. However, real-time data is often not that predictable, making unnecessary requests inevitable and as a result, many connections are opened and closed needlessly in low-message-rate situations.

With *long-polling*, the browser sends a request to the server and the server keeps the request open for a set period. If a notification is received within that period, a response containing the message is sent to the client. If a notification is not received within the set time period, the server sends a response to terminate the open request. It is important to understand, however, that when you have a high message volume, long-polling does not provide any substantial performance improvements over traditional polling. In fact, it could be worse, because the long-polling might spin out of control into an unthrottled, continuous loop of immediate polls.

# What is WebSocket protocol, how is it different from HTTP communication, what advantages it has over HTTP?

Full-duplex.

Small Headers.

Can communicate in either direction at the same time.

# Explain what the WebSocket Protocol brings to the Web-world.

A full-duplex, bidirectional communications channel that operates through a single socket over the Web.

Very small headers.

Once established, WebSocket data frames can be sent back and forth between the client and the server in full-duplex mode. Both text and binary frames can be sent full-duplex, in either direction at the same time. The data is minimally framed with just two bytes.

# Explain and demonstrate, using a package like Socket.io (on the Server), the process of WebSocket communication - From connecting client to server, through sending messages, to closing connection:

# On the client

# On the server

*GET /chat HTTP/1.1*

*Host: example.com:8000*

*Upgrade: websocket*

*Connection: Upgrade*

*Sec-WebSocket-Key: dGhlIHNhbXBsZSBub25jZQ==*

*Sec-WebSocket-Version: 13*

Upgrade er den del i request header som fortæller serveren at der skal laves en websocket connection. Når serveren sender et response som ser sådan her ud:

*HTTP/1.1 101 Switching Protocols*

*Upgrade: websocket*

*Connection: Upgrade*

*Sec-WebSocket-Accept: s3pPLMBiTxaQ9kYGzzhZRbK+xOo=*

At that point, both client and server keep that socket from the original HTTP request open and both switch to the webSocket protocol.

## Sending messages:

With WebSockets you can transfer as much data as you like without incurring the overhead associated with traditional HTTP requests. Data is transferred through a WebSocket as messages, each of which consists of one or more frames containing the data you are sending (the payload). In order to ensure the message can be properly reconstructed when it reaches the client each frame is prefixed with 4-12 bytes of data about the payload. Using this frame-based messaging system helps to reduce the amount of non-payload data that is transferred, leading to significant reductions in latency.

To send a message through the WebSocket connection you call the send() method on your WebSocket instance; passing in the data you want to transfer.

socket.send(data);

You can send both text and binary data through a WebSocket.

## Closing Connection:

Once you’re done with your WebSocket you can terminate the connection using the close() method.

socket.close();

After the connection has been closed the browser will fire a close event. Attaching an event listener to the close event allows you to perform any clean up that you might need to do.

# What's the advantage of using libraries like Socket.IO, Sock.JS, WS, over pure WebSocket libraries in the backend and standard APIs on frontend? Which problems do they solve?

#### But wait, can't I just use WebSockets directly?

Yes, you could. Pretty much all modern browsers, Android and iOS versions offer native Websocket support. But no, you shouldn't! Websocket libraries do a few things that you will definitely want in production. They give you:

#### heartbeats / keep alive calls

these are tiny messages that are send on pre-defined intervals. They make sure that the other side is still responsive and prevent certain network constructs like proxies or firewalls from cutting your connection due to inactivity.

#### reconnecting

if your connection drops, you want it to be re-established. Some libraries also queue messages while the connection is down and resend them once it becomes available again.

#### http fallback

if WebSockets are not available, some libraries fall back to http long polling and other techniques to mimic bi-directional communication (collectively known as "comet"). Depending on your target audience this might not be a necessity, but is still a good choice if your app needs to be available within large corporate infrastructures.

#### [engine.io](https://github.com/socketio/engine.io)

Engine.io is the transport layer that powers socket.io, but can also be used as a standalone library. It incorporates a few unusual design choices, such as starting with http long-polling to ensure immediate connectivity and only upgrade the connection to websocket a bit later.

#### [WS](https://github.com/websockets/ws)

A solid, barebone WebSocket server for node. Used by engine.io

#### [SockJS](https://github.com/sockjs/sockjs-client)

A fast JavaScript / Node.js abstraction layer for websockets, supporting a number of fallback techniques

#### [Primus](https://github.com/primus/primus)

If commitment is not for you, Primus is. Its not a connectivity library in itself, but an abstraction layer that allows you to switch your connectivity library once the initial romance has worn off.

#### [Tornado](http://www.tornadoweb.org/)

Tornado is a general purpose networking library for Python, offering Websocket abstractions and fallbacks.

#### [web-socket-js](https://github.com/gimite/web-socket-js)

Flash based websocket implementation

#### [libwebsocket](https://libwebsockets.org/)

Really fast websocket implementation in low level C.

#### [Fleck](https://github.com/statianzo/Fleck)

Fleck is a WebSocket implementation in C#

#### [Atmosphere](https://github.com/Atmosphere/atmosphere)

Atmosphere is a general websocket abstraction layer for JVM compatible languages. It's focused to run on Java and J2EE application servers and comes with an ecosystem of extension points to connect it to caches like Redis or Hazelcast.

#### [Java Web Socket](http://java-websocket.org/)

Exactly what the name suggests

#### [Mojolicious](http://mojolicious.org/)

A websocket implementation with fallbacks for Perl

#### [SignalR](http://signalr.net/)

WebSocket abstraction for .NET

#### [SuperWebSocket](http://superwebsocket.codeplex.com/)

TCP/Websocket library for .NET

#### [Plezy](http://www.plezi.io/)

WebSocket abstraction for Ruby

# GOD INFO:

http://blog.teamtreehouse.com/an-introduction-to-websockets