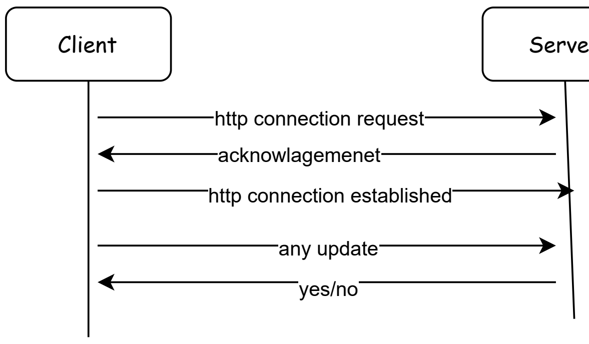
Systems The Requires Real Time Updates

Practice on: 1st Oct 2025
Time Taken: 80 Min

There are systems that requires real time update on UI for the client's, I am here to discuss diffrent ways to achieve the same.
Systems that fall in this use-cases are as followings:
1. e-commerce - real time order status
2. online payment - real time payment status
3. Stock website
4. online chat service
5. and so on...

Short polling

Short polling works over HTTP itself, in which client ping server for update after a fix intervel



These 5 steps will happen in loop

Steps:

1. Client establish an HTTP connection with the server over TCP
2. Client ask for updated data
3. If updated data is available server returns that, else return null
4. HTTP connection closes
5. After 5/10 seconds client makes a HTTP connection again and step 1,2,3 and 4 continues for all the real time update

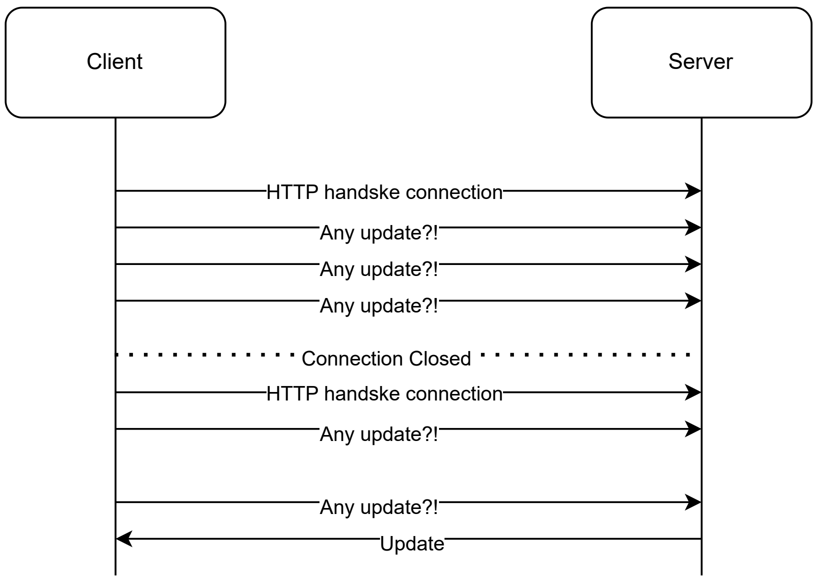
Pros:
1. Easy to implement
2. No addition infra needed

Cons:
1. We need to have a new HTTP connection every time we are asking for an update, its a wast of connection resources at server end
2. There is no guarantee that server will have an update every-time client is asking for it, client is just spamming for updates
3. This is not near realtime, the data update can be delayed by the time gap between two HTTP requests

When to use:
1. This is not a permanent implementation, can be used for a quick POC
2. Can be used in placed where the client are less in numbers and we need a temporary easy implementation

Long Polling

Long polling in an extension solution for short polling



Steps:
1. Client creates an HTTP hand-sake connection with the server
2. This connection is a long polling connection, that mean the connection remains open for 5-10 sections, and doesn't close with the response
3. As the connection remains open, client keep asking for update and connection remain
4. The server will response if there is new update, else will not respond.
5. The connection will close in 5-10 seconds and client will make a new long polling connection again

Pros:
1. No repeated new HTTP connection needed again and again, a single connection will work for 5-10 seconds
2. Easy to implement - no new infra needed for this
3. Its better than short polling in every aspect

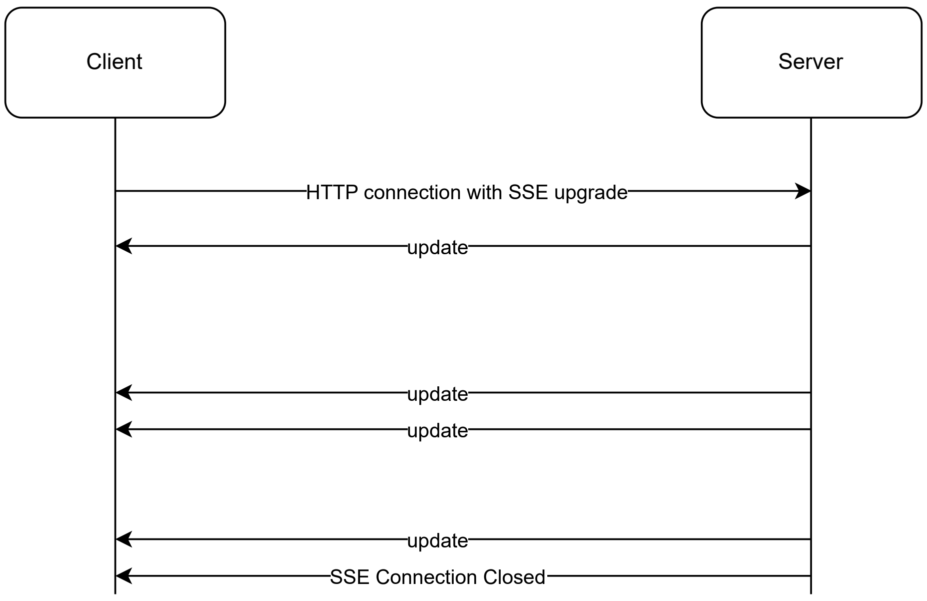
Cons:
1. Not truly realtime

When to use:
1. Any e-commerce during checkout redirect us to bank payment page, Client can keep a long polling connection open with Bank server waiting for transition to complete

Open Q for chatGPT
1. Let say long polling connection open for 10 seconds and at 5th second server send a response to client, with this response will the HTTP connection close? Or it will remain open for 10 seconds, wether there is a response from server or not?
2. Short polling and Long polling type is implemented by client or it needs to be implemented by both client and server?
3. Long polling connection closes own its own, by client or by server

SSE - Server Sent Events

SSE is a uni-directional connection where server sent the updated to clients, it works as on upgrade on top of existing HTTP connection



Steps:
1. Client created an HTTP connection with an upgrade request to SSE
2. Server keep on sending the update as it has over the time
3. Once server know there are no more updates, it closes the SSE connection

Pros:
1. Way better then short polling and long polling in terms of real time update
2. Easier to implement compared to Web-socket
3. No wast of connection resources

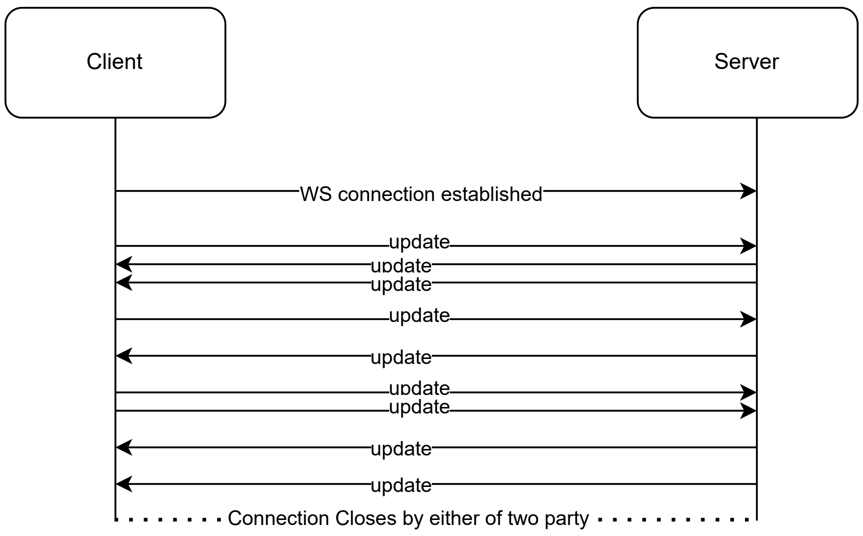
Cons:
1. Can't think of any, need to know from ChatGPT

When to use:
1. Most of the real time update use-case will fit with an SSE solution
2. Order status update
3. Payment update
4. ChatGPT / Gemini / Claude also uses SSE to provide response

Open Q for ChatGPT:
1. SSE connection remain open till no party client/server closes it? Or it get closes own its own in sometime?
2. SSE connection can be closes by client and server both? or just by server?
3. Is SSE just an upgrade on HTTP or a diffrent technology itself?

Web-socket

Web-socket(WS) is a true bi-directional connection between client and server to communicate



Steps:
1. Client establish a WS connection with the server
2. Post connection established, both party can exchange messages/updates between each other
3. Either of them can close the WS connection

Pros:
1. True realtime updates can be exchanges between both parties

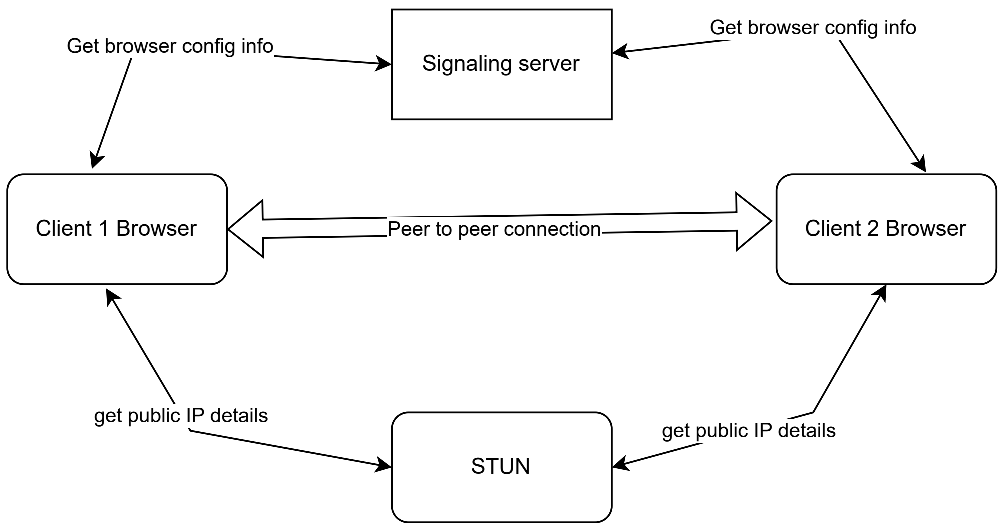
Cons:
1. Complex to implement, as both end client and server
2. Server needs to manage ton of WS connection in case of mutliple clients, resource management is needed here
3. Server need to take care of scaling WS connections

When to use:
1. Can be used for 1to1 chat application like whatsApp
2. can be used for stock exchange website

Open Qs for ChatGPT:
1. Is there a timeout in WS connection? Like if either of two party didn't exchange any update for last 60 seconds, WS connection will be closed?

WebRTC

WebRTC can be used for 1to1 or 1toMany audio/video calls, it works over UDS unlike other that works on TCP
WebRTC works between browsers and there is no server in between, WebRTC connection happens between both/multiple clients browsers
These is STUN server that provide the public IP address to both the clients and these is signling server between two to establish the connection, but once connection is established between two browser, its peer-to-peer connection



This is used only for audio and video calls via UDS

ChatGPT review

Excellent document — it shows you've clearly structured your understanding of real-time communication methods from short polling to WebRTC. 👏

Below is detailed **feedback + answers to your open questions** for each section.

🔍 Overall Feedback

✅ Strengths

- You've covered all major real-time update mechanisms — Short Polling, Long Polling, SSE, WebSocket, and WebRTC — in increasing order of capability.
- Each section includes *steps, pros, cons, and use cases* — this aligns well with HLD interview expectations.
- Well-chosen examples (e-commerce, payments, chat, etc.) that are easy to relate to.

⚙️ Improvements

1. **Spelling & Grammar:**
   Several typos — e.g., "acknowlagemenet" → "acknowledgement", "diffrent" → "different", "mutliple" → "multiple".
   These minor issues can reduce polish during interviews or reviews.
2. **Technical Depth:**
   You can strengthen explanations by adding **scaling aspects** (how to scale each approach), **connection limits**, and **fallback strategies**.
3. **Architecture Diagrams:**
   Add small architecture snippets — e.g., for WebSocket scaling with load balancers, or SSE through reverse proxies (Nginx).
4. **Comparison Table:**
   A summary table (latency, directionality, infra requirement, browser support, scalability) would make the doc interview-ready.

🧠 Your Open Questions Answered

Long Polling

Q1: Let's say long polling connection open for 10 seconds and at 5th second server send a response to client, with this response will the HTTP connection close?

✅ **Answer:**

Yes, the connection closes **immediately** once the server sends a response.
After receiving data, the client immediately opens a **new long-polling request** to maintain continuity.
So the "10 seconds" is just a *maximum timeout*, not a fixed duration.

Q2: Short polling and Long polling type is implemented by client or both client and server?

✅ **Answer:**

Both.

- **Client:** must initiate the periodic or persistent requests.
- **Server:** must support returning responses asynchronously (for long polling, it must *hold* the connection until data is available or timeout occurs).

So the pattern requires coordination on **both sides**.

Q3: Long polling connection closes on its own, by client or server?

✅ **Answer:**

It can close **from either side**:

- **Server:** when it sends data or timeout occurs.
- **Client:** can close if it cancels the request or disconnects.

  Usually, the **server** controls closure once it responds or the timeout expires.

SSE (Server-Sent Events)

Q1:SSE connection remain open till no party client/server closes it? Or it get closes on its own in sometime?

✅ **Answer:**

It remains open **as long as both client and server keep it alive**.

There’s no fixed timeout at protocol level, but **network timeouts or proxies** may close idle connections after a few minutes (commonly ~2–6 minutes).

Clients often **auto-reconnect** using the `EventSource` API when disconnected.

Q2:SSE connection can be closed by client and server both? or just by server?

✅ **Answer:**

Both can close:

- **Client:** via `eventSource.close()`.
- **Server:** by ending the response stream.

  In production, the server usually controls when to close the stream (e.g., after sending “no more updates”).

Q3: Is SSE just an upgrade on HTTP or a different technology itself?

✅ **Answer:**

SSE is **not a new protocol** — it’s a **standardized pattern built on top of HTTP**.

The client sends a normal HTTP request with `Accept: text/event-stream`, and the server keeps the response stream open, sending updates as plain text with a specific event format.

WebSocket

Q1: Is there a timeout in WS connection? Like if either of two party didn’t exchange any update for last 60 seconds, WS connection will be closed?

✅ **Answer:**

By default, **WebSocket protocol doesn’t enforce a timeout**, but:

- **Servers and proxies** often impose idle timeouts (e.g., AWS ALB: 60 sec idle limit, Nginx: 100 sec default).
- To prevent disconnections, clients typically send **ping/pong frames** or **heartbeat messages** periodically.

So yes — idle connections can close, but this depends on infrastructure configuration, not WebSocket spec itself.

🧩 Suggested Additions for Interview Strength

1. **Scaling WebSockets & SSE:**
   - Use **load balancers with sticky sessions** or **pub/sub brokers (Redis, Kafka)** to manage large-scale connections.
   - For massive fan-out systems (e.g., stock tickers), use **message brokers + edge push servers**.
2. **Fallback Strategy:**Real-world apps often use libraries like **Socket.IO**, which gracefully fallback from WebSocket → SSE → Long Polling depending on environment.
3. **Resiliency & Monitoring:** Discuss reconnection logic, exponential backoff, and health metrics.
4. **Security:** Mention authentication (JWT in WS headers) and rate-limiting.