→ Real-time communication and data exchange?

① Short Polling
② Long Polling
③ WebSockets
④ Server Sent Events

→ Storing IP addresses of all clients on server?
→ Privacy Concern → can reveal location
→ Scalability → no. of clients can be massive
→ Dynamic IP addr → IP can change → outdated / incorrect info
→ Network Address Translation (NAT) - home / corporate n/ws share a single public IP, can't differentiate individual clients

Short Polling

→ client sends req to server at regular intervals using HTTP GET reqs.

→ server immediate response with "no new data" resp

→ not efficient use of resources

→ freq & unnecessary reqs

→ latency since client has to wait for server response even if no new data

→ high nlw traffic

→ increased server load

→ not fully real-time system - depends on req. intervals

```
function fetchData():
    response = makeHTTPRequest(Method, URL)
    if response.status == 200:
        processData(response.body)
    else:
        handleRequestError(response.status)

function initiateShortPolling():
    loop forever:
        fetchData()
        wait(5 seconds)

initiateShortPolling()
```

Long Polling

→ reduces latency & server load
→ server doesn't respond immediately
→ server keeps req open until new data becomes
avail or a timeout occurs.
→ once new data is avail, server responds with updated
info & client immediately sends another req to maintain
connect
→ near real-time communicat
→ still requires client to initiate new req every time

```
function initiateLongPolling():
    response: makeHTTPRequest(Method, URL)
    if response.status == 200:
        processData(response.body)
        initiateLongPolling()
    else if response.status == 204:
        initiateLongPolling()   // no new data available
    else
        handleRequestError(response.status)

initiateLongPolling()
```

WebSockets

→ full duplex communication channels over single TCP connection

→ persistent connection

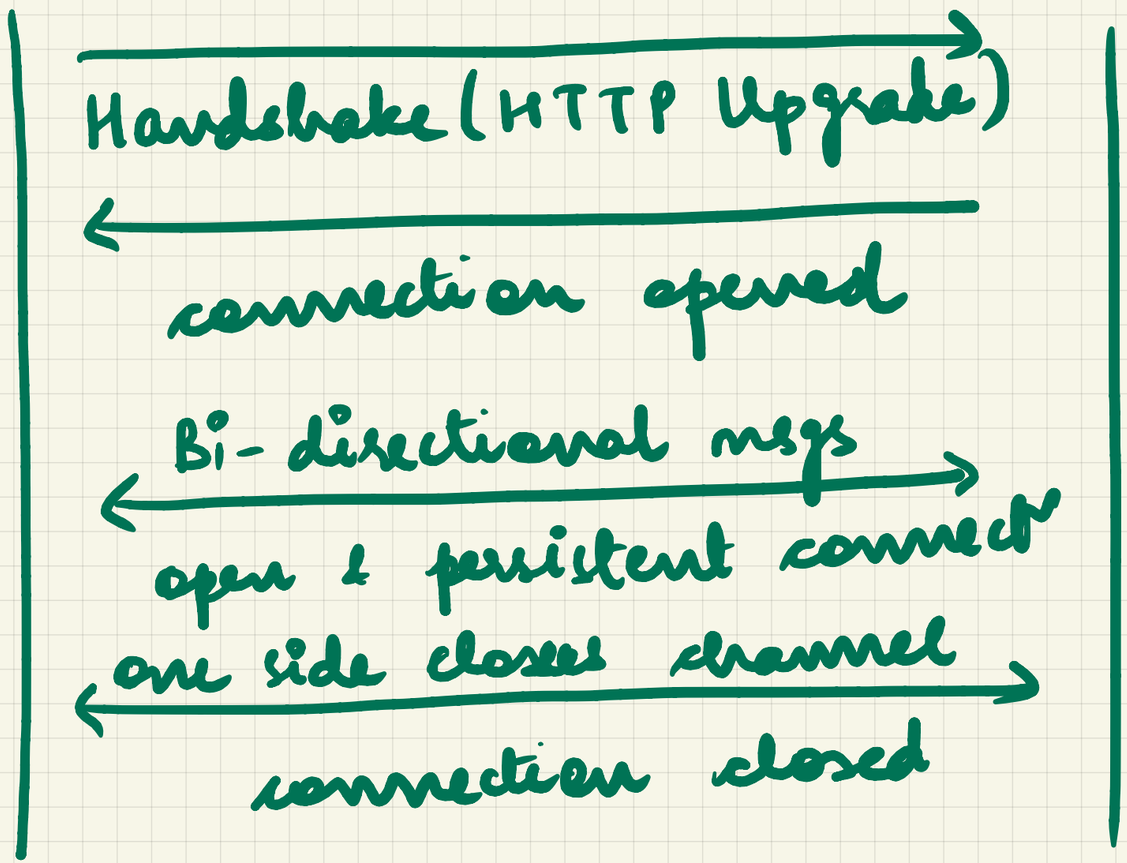
→ allows both parties to send data to each other freely.

→ bidirectional communication

→ eliminates overhead of creating new connections.

→ lower latency, reduced nlw traffic, improved scalability

→ eg- chat apps, collaborative editing tools & real-time dashboards



① Handshake - client sends HTTP req to server, requesting to upgrade connection to WS.

Headers:
Upgrade : websocket
Connection: upgrade

② Server Response - HTTP 101 (switching protocols)
same headers included with web-Socket-Accept header

③ connection upgrade

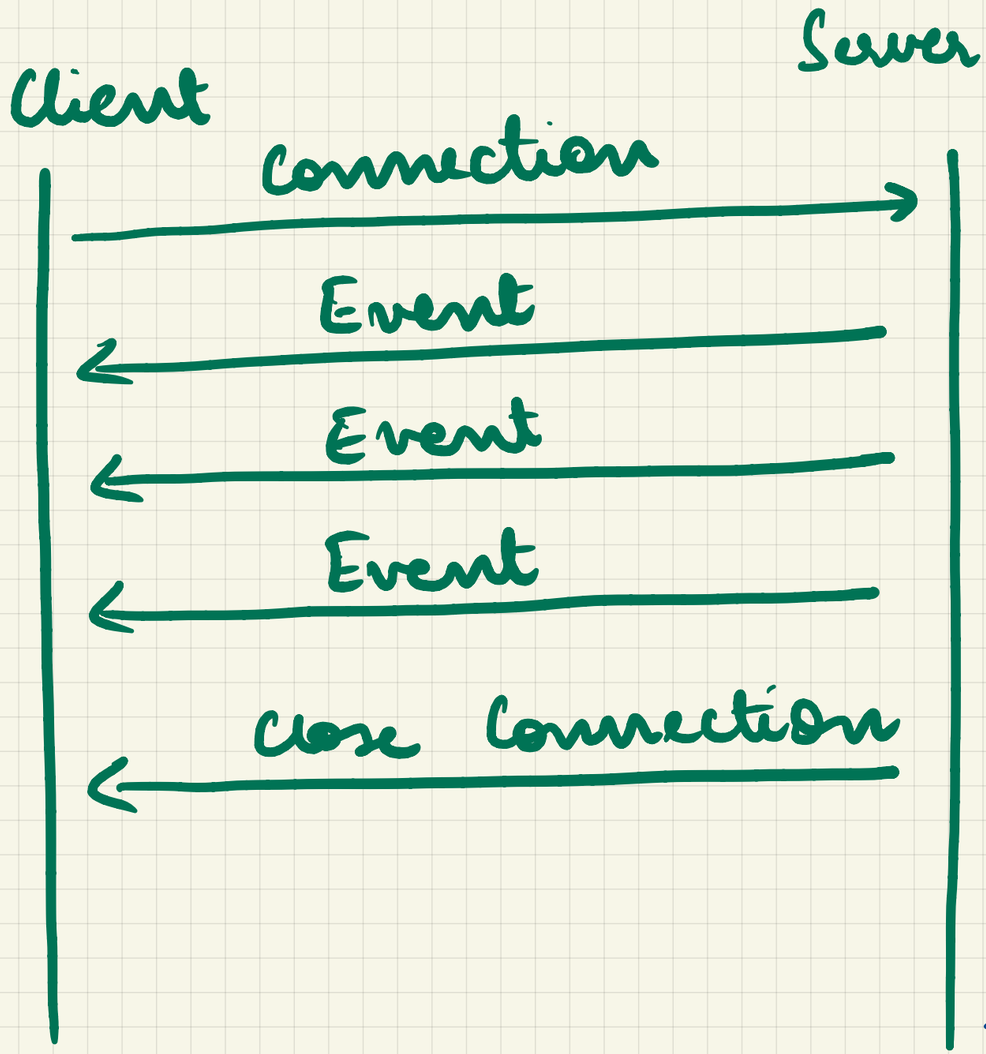
④ Data Transmission - data sent in frames - header, payload
Msgs can be in text / binary format, can be of any length

⑤ Heartbeat & Keep-Alive - to ensure connection remains active, prevent it from being closed due to inactivity

(6) Connection closure - either client or server can initiate by sending close frame. Other party ack & closes on their end as well.

Server Sent Events

→ unidirectional communicat^n protocol
→ allows server to push data to client over single HTTP connect^n
→ focuses on server to client data streaming
→ server sends data as stream of events
→ server continuously pushes new events to client, client can handle these events as they arrive



① Connection Establishment

→ Client initiates HTTP GET req.

→ header → "Accept: text/event-stream"

→ Server receives & prepares to establish SSE connection

② SSE Stream format

→ Server starts sending data as SSE stream

→ SSE stream - series of individual events sent as plain text using specific format

③ Event format:
event field, data (payload), id - identifier of event

④ Sending Events : Server constructs event & sends as part of SSE stream

⑤ Client side - listen for specific event types & process received data accordingly.

⑥ Connection Termination : SSE can be kept open indefinitely. either client or server can close. client can call 'close' method or server can send a specific instruction to close the connection with SSE stream.