Long Polling vs WebSocket

Long Polling

Long polling is the technique that mimics real-time behavior by keeping HTTP requests open until the server has data.

How does it work?

1. Client sends a request to the server, expecting new data.
2. Server holds the request open until it has an update or a timeout is reached.
    - If there's new data, the server immediately responds.
    - If there's no new data and the timeout reached, the server responds with an empty or minimal message.
3. Once the client receives a response — new data or a timeout — it immediately sends a new request to the server to keep the connection loop going.

Pros

- Simple to implement
- Supported universally since it uses standard HTTP, and it works reliably through firewalls and proxies
  Cons
- Higher latency after each update
- Resources-heavy on servers

Use cases

- Simple chat or comment where real-time but slightly delayed updates are acceptable
- Notification systems for less frequent updates
- Legacy systems where WebSockets aren't feasible

WebSockets

Websockets provide a full-duplex, persistent connection between the client and the server.

How does it work

- **Handshake**: Client send HTTP request with Upgrade: websocket.
- **Connection**: If supported, the server upgrades the connection to WebSocket(switching from https:// to ws://). After the handshake, client and

server keep TCP socket open for communication.
- **Full-Duplex Communication**: Once upgraded, data can be exchanged bidirectional in real time until either side closes the connection.

Pros

- Ultra-low latency
- Lower overhead since there's only one persistent connection rather than repeated HTTP requests.
- Scalable for real-time applications that need to support large number of concurrent users.

Cons

- More complex setup
- Some proxies and firewalls may not allow WebSocket connections
- Complexity of implementation and error handling
- Server resources usage might grow if you have a large number of concurrent users.

Use Cases

- Live chat or collaboration tools
- Multiplayer online games with real-time state synchronization
- Live sport/financial dashboards that need to push frequent updates

Choosing the Right Solution

1. **Complexity and Support**
   - **Long Polling** is easier to implement using standard libraries. Any environment that supports HTTP can handle it, often without extra packages.
   - **WebSocket** require a bit more setup and a capable proxy environment. However, many frameworks simplify the process significantly.
2. **Scalability and Performance**
   - **Long Polling** can become resource-intensive with a large number of simultaneous clients, due to multiple open connections waiting on the server side.
   - **WebSocket** offer a more efficient, persistent connection and scale better for heavy, frequent data streams.
3. **Type of interaction**
   - **Long Polling** fits scenarios where data update's are not super frequent. If new data arrives every few seconds on minutes long polling might be enough.

- **WebSockets** are better for high-frequency updates or two-way communications.

4. **Network Constraints**
    - **Long Polling** typically works even in older networks or those with strict firewalls.
    - **WebSocket** might face issues in certain corporate or older mobile environments, though this is less of a problem as the standard becomes more widespread.

Alternative Solutions Worth Considering

1. **Server-Sent Events (SSE)
    - Allows the server to push messages to the client over HTTP.
    - It's simpler than WebSockets for one-way communication, but not full-duplex.
    - Best suited for use cases like news feeds, real-time notifications, and status updates.
2. **MQTT**
    - Commonly used in IoT for lightweight publish-subscribe messaging.