Real-Time Communication in Modern Web Applications: Techniques and Comparisons

Peter Bobka, Ján Janech, Patrik Hrkút

***Abstract* —** The evolution of web applications from static pages to dynamic, interactive platforms has make necessary the development of real-time communication technologies. This paper explores the various methods enabling real-time communication between clients and servers, focusing on Short Polling, Long Polling, Server-Sent Events (SSE), and WebSockets. Each technology is analyzed in terms of its functionality, advantages, limitations, and appropriate use cases. Special attention is given to the role of these technologies in delivering a seamless user experience in applications such as online gaming, live broadcasting, and collaborative tools. The study also compares these methods, highlighting factors like latency, efficiency, and ease of implementation to aid developers in selecting the most suitable technology for specific application requirements. By examining current trends and challenges in web communication, this paper provides insights into the architectural decisions crucial for building responsive and efficient web applications.

***Keywords* —** Real-time communication, Short Polling, Long Polling, Server-Sent Events (SSE), WebSockets, real-time updates, interactive web technologies, modern web development.

I. Introduction

While in the past, websites provided users with mostly static content, today the displayed content changes right before the user's eyes without them having to do anything. Static pages have been replaced by web applications, whose content users no longer just read, but create.

One of the important features of modern web applications is real-time communication. Its goal is to ensure that the content displayed to the user is continuously updated as needed, or that new data is delivered to the user regularly. This includes, for example, online games where multiple players need to synchronize data, chat applications where users can communicate with each other, or possible live audio/video broadcasting.

Real-time communication technologies such as *WebSocket* protocol or *Server-sent events (SSE)* offer many advantages in the web environment. For example, more efficient communication between the server and the client, or improved synchronization between users of a web application. Despite the existence of these technologies, there are still applications that do not use these technologies and instead use older techniques such as *Short polling* or *Long polling*.

The choice of suitable technology to execute the finished product is a component of web application development. Their choice depends on the specific situation and needs of the given project, while over time it is necessary to update, change, and add new technologies to ensure continuity in development and, finally, to bring the best possible experience to users when using applications on the Internet.

II. Traditional Model of Communication

A. Client-server

The basic concept of this communication is sending data between devices that are connected through a network based on *client-server model* [1]. This fundamental interaction between a

Peter Bobka, University of Zilina, Zilina, Slovakia (e-mail: peter.sevcik@fri.uniza.sk)
Ján Janech, University of Zilina, Zilina, Slovakia (e-mail: jan.janech@fri.uniza.sk)
Patrik Hrkút, University of Zilina, Zilina, Slovakia (e-mail: patrik.hrkut@fri.uniza.sk)

client and a server forms the basis for delivering content and services over the internet. Regular client-server communication employs traditional models that enable data exchange using HTTP protocols.

Communication between the client and server is facilitated by the *Hypertext Transfer Protocol* (HTTP), a stateless protocol that defines the format and transmission of messages. HTTPS, the secure variant of HTTP, incorporates encryption through Transport Layer Security (TLS) to ensure data confidentiality and integrity during transmission. Data exchanged between clients and servers is typically structured using formats like HTML, JSON, or XML. JSON is widely used in modern web applications due to its lightweight and easily parsable structure. The process of communication involves the client sending an HTTP request to the server, specifying the required resource using methods such as GET, POST, PUT, or DELETE. The server processes the request, performing necessary computations or retrieving data from databases, and then sends an HTTP response containing the requested resource or an appropriate status code indicating the outcome of the request.

The client represents the user's device, typically running a web browser or application. It serves as the interface through which users interact with the server, generating requests based on user actions such as clicking links or submitting forms. The server is a powerful computing entity that processes client requests. Servers host resources such as web pages, APIs, and multimedia content.

HTTP's stateless nature means that each request from the client is independent and unrelated to previous requests. While this simplifies server design, it necessitates additional mechanisms such as cookies or sessions to maintain state information when required. Regular client-server communication often follows a synchronous pattern, where the client waits for the server's response before proceeding. This ensures that the client has the most up-to-date information but can lead to latency issues if the server response is delayed. Reliability in communication is ensured through protocols like Transmission Control Protocol (TCP), which underpins HTTP. TCP guarantees that all packets of data are delivered in order and without loss, making it suitable for applications where data integrity is critical.

This method of communication is not suitable for creating real-time applications, because every time the content changes, even the smallest one, it requires sending a request to the server and the server returns the entire newly generated web page. Such communication is inappropriate in real-time applications and, moreover, communication can only be initiated by the client, which does not meet the requirements of real-time applications at all, because the changed data from other clients does not reach the client immediately.

III. Real-time communication

In the case of web applications in which multiple users collaborate, it is necessary to synchronize data or displayed content between them. A typical example would be a chat, through which people on the Internet can communicate with each other. After adding a message, it is necessary to save it in the database and deliver it to the recipient. In principle, it is necessary to ensure that after changes are made on the server side, the data displayed to the user on the client side is updated.

A. AJAX

According [2] AJAX is defined as "Asynchronous JavaScript and XML (AJAX) is a web application development technology in which an application retrieves data from a server by sending asynchronous HTTP requests and uses the new data to update relevant parts of the page without having to reload the entire page."

In [3], the author describes AJAX technology in the context of the era known as "Web 2.0", in which web applications are beginning to function more like desktop applications.

The loading of data that occurs asynchronously in the background of the displayed page in the web browser during the use of the application has prompted the emergence of frameworks for creating so-called Single-page applications (SPA). Since updating the content displayed to the user is no longer conditional on loading the entire page, the total amount of data that needs to be loaded from the server is also reduced. Static websites are being replaced by modern, dynamic, interactive web applications.

AJAX is more suitable for creating real-time applications because it allows only the transfer of data that has changed, and in a structured form. Unfortunately, like the client-server communication model, communication can only be initiated by the client, so immediate data exchange is also not possible, and the client is dependent on synchronously querying the server for changes.

B. Short polling and long techniques

Displaying current data can also be achieved in the usual way, which consists of sending HTTP requests and responses between the client and the server. In combination with AJAX technology, data can be loaded from the server while the user is working with the application, and the displayed data can be updated as needed.

The first described method is a technique called *Short polling*, which is based on regularly retrieving data from the server. The client regularly sends a request for new data to the server. If the server contains new data, it sends it back to the client in a response. Otherwise, it sends an empty response. When the client receives new data, it displays it to the user. The problem is that it is not possible to clearly determine how often the data should be retrieved from the server. Frequent retrieval of data ensures that the user sees the latest data. However, this may mean that the server responds to many requests with an empty message. Their processing can be seen as redundant work for the server. Short polling is therefore not a very efficient solution.

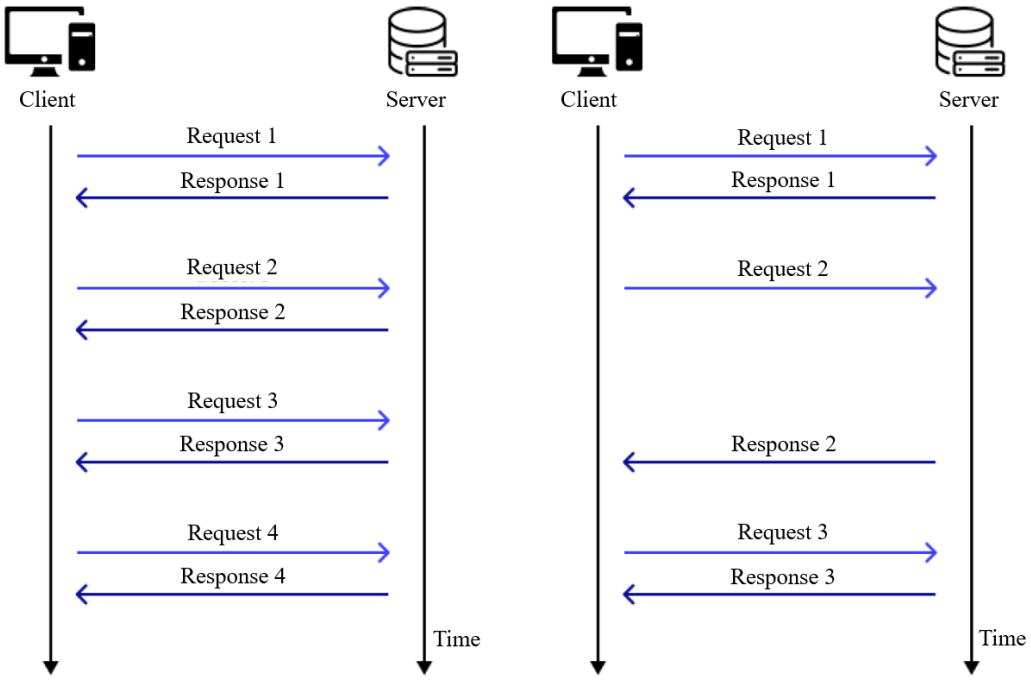

Fig. 1 Scheme of Short and long polling communication

The second technique, called *long polling*, differs in how the server handles requests for new data sent by the client. If no new data has been received since the server last processed the request, the server temporarily stores the request. Any requests for new data that are stored in this way are only responded to by the server when new data is available. [4]

C. Server-Sent Events (SSE)

SSE is a technology that allows data to be sent from the server when the server deems it necessary. A stable connection is established between the client and the server, within which all communication takes place. The connection is active for the entire time the server is sending data. The main advantage is low response time with the possibility of scaling with increasing demands on the number of active connections. [5]

Server-Sent Events (SSE) are particularly good for scenarios requiring real-time one-way communication from the server to the client. In the context of content-heavy applications like newsfeeds or social media platforms, SSE ensures that users receive the latest data efficiently without requiring constant page refreshes or manual interactions. By reducing overhead compared to traditional polling, SSE enhances user experience while maintaining server performance.

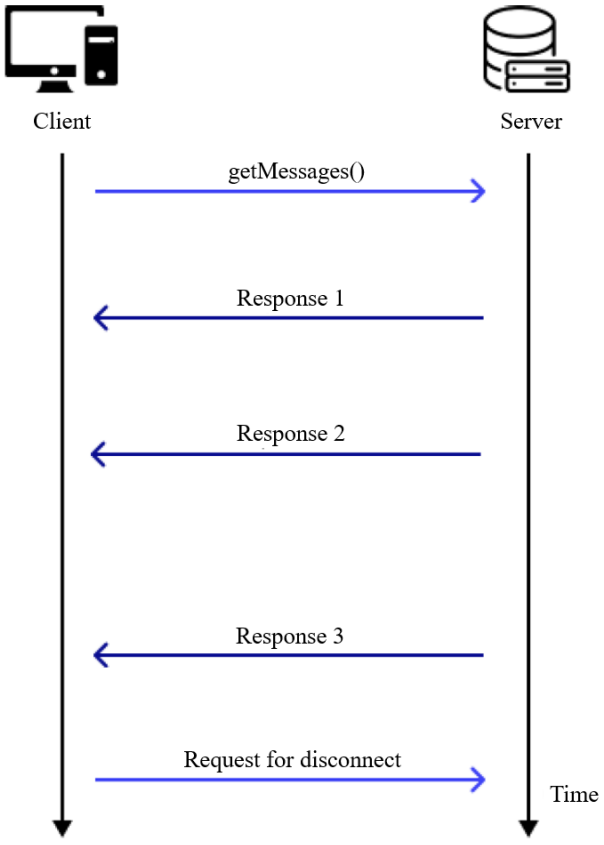

Fig. 2 Scheme of Server-sent events communication

D. WebSocket protocol

The *WebSocket* protocol, like SSE technology, is used in applications that support real-time communication. In both cases, a stable connection is established between the client and the server during communication. The WebSocket protocol differs in principle in that communication within the established connection can be *bidirectional* – data can be sent by both the client and the server. The same functionality can be achieved in the case of SSE by

combining it with AJAX technology. SSE is a suitable alternative to the WebSocket protocol. The disadvantage is the limited number of active connections within the browser and support for sending messages only from the server. On the other hand, the WebSocket protocol creates a two-way communication channel between the client and the server, supports many connections within the browser, and supports sending binary data. [6]

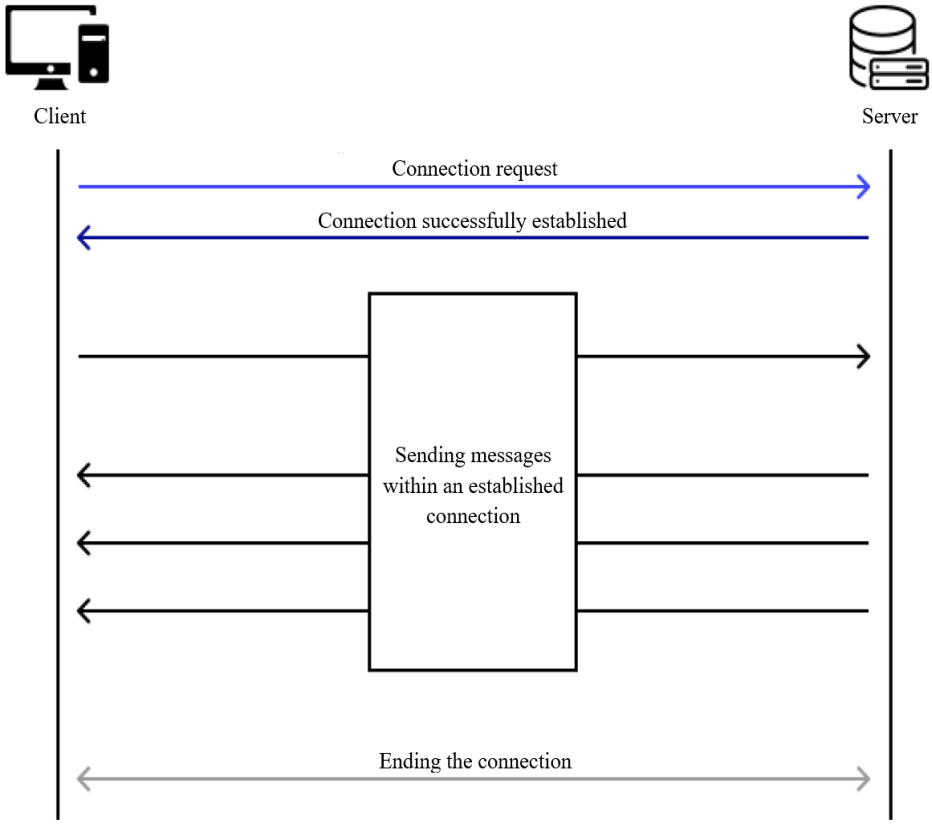

Fig. 3 Scheme of WebSocket communication

Lifecycle of WebSocket communication

The life cycle of a connection to a WebSocket server begins when the client initiates the connection via an HTTP request and ends when the client disconnects. Within a given connection, it is possible to connect to communication channels through which data is sent in the form of events. On the client side, functionality is defined is executed after a specific event is captured. Because the connection between the client and the server is two-way, events can be sent from the server to the client side and vice versa via channels.

In addition to basic functionality, the library provides the ability to monitor everything that happens within the communication via the WebSocket server, either through a graphical interface or in the console. [5]

Event

An event represents some information sent as part of a communication that can be received by the user and then acted upon. It has a name, a communication channel through which the event is sent, and can contain additional data. Unless the developer explicitly sets the data structure, the event is sent along with a list of class attributes marked as public.

Channels

As mentioned, events are broadcast over a specific communication channel. Each of them has its own name, which is used when defining events and connecting on the client side. WebSocket allows you to use 3 types of communication channels:

- Public channel
- Private channel
- Presence channel

The first type of communication channel is the *Public channel*. Any user of the application who has successfully established a connection to the WebSocket server can connect to this type of channel and respond to events sent through it. No authentication or authorization process is required.

Another type is the *Private channel*. The process of connecting to this type of communication channel is carried out via a classic HTTP request sent by the client. It requires that the user be authenticated. In addition, the server checks whether the user is authorized to connect to the given channel and either allows or denies the connection accordingly.

The last type of communication channel is the *presence channel*. In principle, it works similarly to the private channel. It provides the ability to send events through it and perform the authorization process for users who connect to it. In addition, it offers information about the users connected to this channel.

If the authorization check is successful and the client can successfully connect to the channel, it is provided with a list of currently connected users. In addition, the client has the option to respond to connecting with a new client or disconnecting a connected client.

IV. Comparison of Short/Long Polling, Server-Sent Events, and WebSockets

Each method has distinct advantages, limitations, and suitable use cases. Below is a comprehensive comparison of these technologies.

Short Polling is one of the simplest methods for fetching updates from a server. The client periodically sends HTTP requests to the server at fixed intervals, querying for new data. If there is no new data, the server responds with an empty response. This method is easy to implement and widely supported, working with any HTTP-compliant server or client. However, it is inefficient due to frequent requests, even when no new data is available, which increases server load and network traffic, especially with many clients. Short Polling is best suited for applications with infrequent updates or where simplicity is prioritized over performance.

Long Polling improves upon Short Polling by keeping the connection open until new data is available. The client sends an HTTP request, and the server holds the request open until it has new data to send. This approach reduces redundant requests compared to Short Polling and provides near real-time updates. However, it requires maintaining open connections, which increases server memory usage, and connection timeouts may occur, requiring re-establishment of connections. Long Polling is commonly used in chat applications or notification systems where updates are frequent but not constant.

Server-sent Events allow the server to push updates to the client over a single, long-lived HTTP connection. Data is sent as events, making it suitable for applications requiring one-way communication. SSE is efficient for one-way data flow from server to client, with built-in reconnection mechanisms and a simplified implementation compared to WebSockets for one-way updates. However, it is limited to text-based data, requiring encoding for binary data, and lacks built-in support for bi-directional communication. SSE is ideal for real-time dashboards, live feeds, or notification systems where updates flow from server to client.

The *WebSocket* protocol enables full-duplex communication between the client and server over a single connection. It establishes a persistent connection, allowing data to flow in both directions with minimal latency. WebSockets are ideal for bi-directional communication, offering low latency and efficient use of network resources, with support for both text and binary data. However, they are more complex to implement and manage compared to SSE or polling, and they require modern browsers and WebSocket-compatible servers. WebSockets are particularly suitable for real-time multiplayer games, collaborative tools, or any application requiring interactive, two-way communication.

TABLE I
COMPARISON OF ALL COMMUNICATION TYPES

Feature	*Short Polling*	*Long Polling*	*Server-Sent Events (SSE)*	*WebSockets*
Communication	One-way (client->server)	One-way (client->server)	One-way (server->client)	Bi-directional
Connection Persistence	No	Partial	Yes	Yes
Latency	High	Medium	Low	Very Low
Efficiency	Low	Medium	High	High
Binary Data Support	No	No	No (requires encoding)	Yes
Ease of Implementation	High	Medium	Medium	Low

E. Choosing the Right Technology

The choice of technology depends on the application's requirements. Short Polling is suitable for simple applications with low-frequency updates. Long Polling is a better choice when updates are more frequent, but simplicity remains a priority. SSE is an excellent option for real-time one-way updates, especially when text-based data is sufficient. WebSockets are ideal for interactive, bi-directional communication where performance is critical. Each of these technologies has a unique role in real-time web communication, and understanding their strengths and limitations helps in making suitable architectural decisions.

V. CONCLUSION

Real-time communication has become an integral part of modern web applications, enabling dynamic and interactive user experiences. This paper examined various technologies, including Short Polling, Long Polling, Server-Sent Events (SSE), and WebSockets, each offering unique capabilities tailored to specific application needs. While Short and Long Polling provide simpler implementations, their inefficiencies limit their suitability for real-time demands. SSE and WebSockets, in contrast, offer low-latency, scalable solutions for one-way and bi-directional communication, respectively.

The choice of technology depends on the application's requirements, such as the need for real-time updates, communication directionality, and performance considerations. WebSockets are ideal for interactive and resource-intensive scenarios like multiplayer games and collaborative tools, while SSE remains a robust choice for one-way updates in dashboards and notification systems.

As web applications continue to evolve, developers must carefully evaluate the trade-offs of each technology to deliver efficient, secure, and reliable solutions. The integration of real-time communication technologies will remain essential to meet user expectations in an increasingly connected digital environment.

REFERENCES

[1] Client-Server Overview, Available online, Accessed 05/07/2024, https://developer.mozilla.org/en-US/docs/Learn_web_development/Extensions/Server-side/First_steps/Client-Server_overview
[2] Ajax - MDN Web Docs Glossary: Definitions of Web-related terms. Available online, Accessed 05/06/2024 https://developer.mozilla.org/en-US/docs/Glossary/AJAX.
[3] Anthony T. Holdener. *Ajax: The Definitive Guide.* O'Reilly Media, Inc., 2008. ISBN: 978-0-596-52838-6.
[4] HTTP Long Polling - What it is and when to use it. Available online, Accessed 08/06/202, https://ably.com/topic/long-polling.
[5] An introduction to Server-Sent Events: A WebSockets alternative ready for another look. Available online, Accessed 05/06/2024. https://ably.com/topic/server-sent-events.
[6] WebSockets vs Server-Sent Events: Key differences and which to use in 2024. Available online, Accessed 05/06/2024, https://ably.com/blog/websockets-vs-sse.