Comparative Analysis of Instant Messaging Protocols and Technologies for Effective Communication in Computer-Mediated Environments

Nataliia Sharonova [1], Iryna Kyrychenko[2] and Daria Shapovalova[2]

[1] *National Technical University "KhPI", Kyrpychova str. 2, Kharkiv, 61002, Ukraine*
[2] *Kharkiv National University of Radio Electronics, Nauky Ave. 14, Kharkiv, 61166, Ukraine*

Abstract

The Internet is based on communication and information exchange between users. One of the most important functions that this network provides us with is the ability to communicate with other people in real time, regardless of the location of the interlocutor. This paper focuses on existing approaches to implementing instant messaging in web applications. The goal of this research is to analyze existing protocols and technologies for instant messaging, describe their advantages and principles of work, select approaches for comparison and measurement criteria. This paper documents the process of comparison, as well as provides justification of the evaluation results and gives recommendation on their basis.

Keywords

Computer-mediated communication, Web applications, instant messaging, WebSocket, server-sent events, long polling, short polling, client pull, server push

1. Introduction

Today, the Internet has become an integral part of most people's everyday lives. The technologies that use this network and the opportunities that they provide to users are rapidly developing and changing every day. The basis of this network is to ensure communication and exchange of information between users. One of the most important functions that the Internet provides is the ability to communicate with other people in real time, regardless of the location of the interlocutor.

There are several different approaches to ensuring online communication between people. One of the earliest and most widespread is email. Email addresses are required to register for many services on the Internet, and it is generally assumed that every user on the Internet has at least one email address. When using email, user can write a letter and send it to the recipient's email address. The letter will be sent, and the user will receive a reply after a certain time. There are different advantages and disadvantages of using email addresses for communication. Among its advantages, we can highlight the following:

1. Ubiquity: email is ubiquitous and is used by individuals and businesses around the world. Most people have an email address, making it a convenient and efficient way to communicate with a wide range of people;
2. Convenience: email can be accessed from anywhere with an internet connection, allowing users to send and receive messages at their convenience. It also allows users to send attachments such as documents, images, and videos;
3. Record keeping: email messages can be saved and archived, making it easy to keep track of communication and reference old messages if needed;

COLINS-2023: 7th International Conference on Computational Linguistics and Intelligent Systems, April 20–21, 2023, Kharkiv, Ukraine
EMAIL: nvsharonova@ukr.net (N. Sharonova); iryna.kyrychenko@nure.ua (I. Kyrychenko); daria.shapovalova@nure.ua (D. Shapovalova)
ORCID: 0000-0002-8161-552X (N. Sharonova); 0000-0002-7686-6439 (I I. Kyrychenko); 0009-0007-0580-7902 (D. Shapovalova)

4. Formality: email provides a formal mode of communication that is appropriate for professional and business settings;
5. Reduced cost: email is typically less expensive than traditional mail and phone calls, making it a cost-effective means of communication.

However, email communication also has its disadvantages:

1. Spam: email is often plagued by spam messages, which can clog up inboxes and make it difficult to manage communication;
2. Security: email messages can be intercepted and hacked, making it essential to use secure channels and encryption to protect sensitive information;
3. Misinterpretation: email messages can be misinterpreted, leading to confusion, misunderstandings, and potential conflicts;
4. Overload: email overload can be a problem, with many people receiving too many messages and struggling to keep up with their inbox;
5. Lack of immediate feedback: email messages are asynchronous, meaning that there can be delays in responses, making it difficult to have immediate feedback or conversations.

Email is a useful tool for communication, but it has some disadvantages that can make it challenging to use effectively. This method is still working, but over time, other technologies have emerged that are more convenient to use.

Another way to communicate online is through forums. A forum is a type of website where members can post questions, start discussions, or participate in various discussions. Each individual question or discussion is called a topic. A forum is usually updated and monitored by an administrator or moderator. Online forums can help build a sense of community among users who share common interests or goals. Forums are an excellent source of information on a variety of topics. Users can ask questions, share their knowledge and expertise, and receive feedback and advice from other members of the community. Online forums can facilitate collaboration among members of a community, allowing users to work together on projects, share ideas, and provide feedback to each other.

One of the most common ways to communicate today is through Instant Messaging (IM). It is a form of text-based communication in which two or more people engage in a single conversation via their computers or mobile devices in an online chat room. It differs from the commonly accepted definition of chat, in which user participates in a more public, real-time conversation in a chat room where everyone on the channel can see everything that all other users are saying.

In its simplest form, instant messaging seeks to accomplish two goals: availability monitoring to send availability notifications to chat users and messaging. The software relies on a central server or servers to monitor presence. When a user logs in to an instant messaging system, their login is recognized and other online users who have that address listed as a "buddy" or friend receive notifications of the user's presence. The software establishes a direct connection between users so that they can communicate with each other synchronously in real time.

There are several different approaches to providing instant messaging. All of them fulfill their purpose and provide real-time communication between users: messages in chats appear as soon as the other person sends them, and the conversation can take place without delay. But each of the existing approaches implements this communication in a different way.

The main types of instant messaging implementation are client pull and server push. Each of these approaches has several different implementations, which will be discussed in this paper.

The purpose of this research paper is to compare existing protocols and approaches for implementing instant messaging. Since instant messaging is a popular and widespread technology, the study of ways to implement it in order to identify the most optimal one can be considered relevant in modern realities.

This paper introduces four distinct methods for putting instant chat web systems into practice. It explains what instant messaging is and how it differs from other online communication methods in terms of benefits. This is the first time the problem has been posed in this formulation. This paper introduces four distinct methods for putting instant chat web systems into practice. It explains what instant messaging is and how it differs from other online communication methods in terms of benefits.

2. Related Works

Instant messaging tools and chat applications has been around us for some time, so some research has already been conducted.

One of the most recent papers is a research conducted by J. Botha, C. Van 't Wout, and L. Leenen titled "A Comparison of Chat Applications in Terms of Security and Privacy" [1]. This research paper focuses on comparison of the most popular chat applications in terms of their security and privacy features, ways to store and access users' data, and range of functionality available in paid and free applications. The study evaluated the applications based on criteria such as encryption, data storage, metadata collection, user control, and open-source code availability. This study is important in the context of researching instant messaging systems as it provides an analysis and comparison of popular chat applications, focusing on their security and privacy features, as well as highlights the importance of data protection, encryption, metadata collection, and third-party access in ensuring user privacy and security. The findings provide valuable information for users to make informed decisions when choosing a chat app that aligns with their security and privacy preferences. The research is significant in promoting greater awareness and understanding of the importance of security and privacy in online communication.

Another similar work was published in "IOP Conference Series: Materials Science and Engineering" in 2020 by R. M. Ali and S. N. Alsaad [2]. The paper is titled "Instant messaging security and privacy secure instant messenger design" and it focuses on the threats on the privacy of social networks and the trends of its solution, depicts the criteria of security in IMs and proposes design of secure IM application. This work references creates and describes criteria to identify if the messaging application can be considered secure.

The article "Securing Instant Messages With Hardware-Based Cryptography and Authentication in Browser Extension" [3] describes a method of integrating hardware-based public key cryptography into Converse.js, an open-source instant messaging client for web browsers that uses the Extensible Messaging and Presence Protocol (XMPP). This is achieved by creating a plugin for Converse.js, which replaces some of the client's original functions, and a browser extension that works in conjunction with the plugin to handle encryption and decryption services for each message sent and received. By combining these components, the researchers were able to experimentally verify their proposal, which provided digital certificates to verify the identity of IM users and protected their messages with hardware-based cryptography.

A paper titled "A Model for Social Communication Network in Mobile Instant Messaging Systems" [4] presents a novel concept called the social communication network (SCN), which takes into account the unique structural characteristics of communication in mobile instant messaging (MIM) systems. The authors propose a model that can represent and generate the SCN in MIM systems, encompassing all social interactions among users, groups, and channels and accurately reflecting the statistical patterns observed in real-world data. The study also redefines some existing properties and introduces new ones that prior models of complex networks fail to capture. To assess the efficacy of the proposed model, the researchers conduct several simulation experiments and compare the results against a real-world graph derived from Telegram.

In the paper "Communication-Efficient and Fine-Grained Forward-Secure Asynchronous Messaging" [5] authors explore the development of a new technique called forward-secure puncturable encryption (FSPE) to achieve practical forward-security for asynchronous messaging systems. FSPE enables fine-grained forward security, allowing users to maintain the decryption capacity of messages that have not yet been received while ensuring the security of already-received messages even if the secret key is compromised. The researchers propose an efficient FSPE scheme for achieving fine-grained forward-secure asynchronous messaging and extend it to support outsourced decryption to improve efficiency. They also implement the proposed scheme and evaluate a proof-of-concept of the main algorithms to enhance confidence in its correctness and feasibility.

The study "iOS Digital Evidence Comparison of Instant Messaging Apps" [6] involved analyzing digital evidence and comparing findings from WhatsApp, Telegram, and Messenger instant messaging (IM) applications on iOS version 13.3, under two conditions: jailed and jailbreak. The NIST 800-101 Revision 1 method was utilized to guide the forensic process, while Cellebrite UFED 4PC tool was

used for data acquisition, and Oxygen Forensic Detective, FTK Imager, and Autopsy for data examination and analysis. The study results were then compared to the iPhone's condition and executed scenarios.

The purpose of the paper “The Impact of Instant Messaging on the Energy Consumption of Android Devices” [7] is to evaluate the impact of the number and distribution of received instant messages on the energy consumption of Android devices. The study confirms that the energy consumption of the Android device is directly proportional to the number of received messages for both apps. Overall, this study provides evidence that even a relatively low number of received messages (10 per minute) can substantially reduce the battery life of an Android device, and sending bursts of messages does not have a significant impact on energy consumption.

The paper titled “OSC-MC: Online Secure Communication Model for Cloud Environment” [8] addresses the problem of exploiting outsourced data involved in online communication and proposes an Online Secure Communication Model for Cloud (OSC-MC) by identifying and terminating malicious VMs and inter-VM links prior to occurrence of security threats.

In a recent research “Online Deep Neural Network for Optimization in Wireless Communications” [9], the authors present a novel online deep neural network (DNN) approach in correspondence to address general optimization issues in wireless communications. A separate DNN is trained for each data sample, with the optimization variables treated as network parameters and the objective function as the loss function. Because of the online optimization approach, this proposal has excellent generalization ability and interpretability.

The paper called “Research of Ways to Increase the Efficiency of Functioning Between Firewalls in the Protection of Information Web-Portals in Telecommunications Networks” [10] provides the results of a full-factor experiment and analysis of experimental data to increase the performance between network screens in information security systems of telecommunications networks. Several factors that impact the interaction of firewalls are examined, including packet fragmentation, network structure, auxiliary functions, and security policy. Empirical relationships that reflect the impact of the security policy on the performance between network screens are derived. It is demonstrated that redistributing security rules can enhance the network's level of information security. A methodology for analyzing firewall performance is presented, which entails initial processing of experimental data and secondary processing to ensure satisfactory precision of outcomes. The study outcomes can be employed in telecommunications network security systems to enhance the security of web portals.

The research “Formal representation of knowledge for info communication computerized training systems” [11] conciders various questions of creation of integrated development environment for info communication training systems and describes ways to improve the efficiency and quality of learning process with computer training systems for distance education are pointed out.

The paper titled “Metrics applicable for evaluating software at the design stage” [12] reviewed and analyzed the existing metrics used in the software evaluation process at the design stage. Discussed issues related to the selection and accounting of indicators affecting the evaluation of software at the design stage in the management of software development. The study of the metrics used to perform the evaluation of software at the design stage. Found conclusions about the use of the metrics examined in management practice. Proved the need to study the metrics described in the Rada standards governing the process of developing software.

The aim of the article “Beyond First Impressions: Estimating Quality of Experience for Interactive Web Applications” [13] is to examine how waiting time for a user's interactions during a web browsing session can be measured and to establish the correlation between waiting time and user-reported perceived quality, with web maps as a use case. It proposes two new measures: interactive Load Time and Total Completed interactive Load to establish the waiting time associated with a web application user’s interaction.

Another paper called “Holistic Web Application Security Visualization for Multi-Project and Multi-Phase Dynamic Application Security Test Results” [14] forms a generic data structure using data sources that are commonly accessible to most web applications. The main contribution of the study is a new dashboard tool that can visually represent the results of dynamic application security tests. The paper also introduces the metrics and measures provided by the tool. Furthermore, a validation study is conducted in which participants are asked to answer quiz questions after using the tool prototype.

The article “Using Server-Sent Events for Event-Based Control in Networked Control Systems” [15] discusses the use of Server-Sent Events (SSEs), a standard HTTP technology that allows communication between a server and a client based on messages sent by the server. SSEs are commonly used in web applications and can be defined with different event triggers that clients can subscribe to. The article proposes the use of SSEs in event-based control strategies, with the plant acting as the server and the controller as the client. The article compares SSEs to other possible solutions and presents a communication workflow between the server and client to define event triggers and subscribe to event streams.

In [16] the authors discuss the difficulties associated with the authentication process and the efforts being made to enhance its usability. Ultimately, recommendations are provided for service providers, along with suggestions for future research.

In the research [17] the author discusses the increased use of technology in education and how instant messaging apps like WhatsApp and WeChat are being used to support language learning. The article focuses on a pilot study that used WeChat as the messaging app and analyzed posts made by students to understand the perceived purposes of using the app for collaboration, peer-support, and knowledge sharing. The findings suggest that WeChat served as a vital link between students, their classmates, and teachers, and was used as a social platform to support the key educational purposes of the program.

Another paper called “Optimization and improvement of bidirectional connections with Web Socket on Synapse framework using Web Workers technology” [18] uses WebSocket connection functionalities to improve connection between devices used for research and highlights performance optimization in comparison to implementation without web workers.

The paper “Gradational Correction Models Efficiency Analysis of Low-Light Digital Image” [19] describes approaches to efficiency comparison and analysis using models of gradation correction when solving actual applied problems of improving the quality of darkened digital images.

In the work [20] the authors point out that end-user devices with multiple access networks can achieve better application performance by distributing traffic across them, but matching application traffic to the most suitable network is challenging due to varying needs and network characteristics. The proposed solution is an application-informed approach for access network selection (IANS), which selects the better access network based on Web resource size and latency and available downstream capacity. IANS was implemented in the Socket Intents prototype and evaluated for Web page loads under different network conditions and for various Web pages. IANS provides the highest speedups in scenarios with asymmetric network conditions and low downstream capacity.

Many of the abovementioned papers make security of communication their main focus. The goal of the current paper is to take security into account as one of the main factors when comparing different instant messaging techniques. Aside from security, we can also define the following criteria to compare by:

- Speed. Speed is an important aspect of instant messaging chats because it enables real-time communication and collaboration. Unlike email or other forms of communication, IM allows users to exchange messages quickly and efficiently, allowing for instant feedback and responses. This can be particularly important in business and professional settings where timely communication can make a significant difference in decision-making and productivity. In addition, speed can also enhance social connections and relationships, allowing users to maintain more frequent and meaningful interactions with friends, family, and colleagues. Speed is a key factor in the effectiveness of IM chats and contributes to the convenience, efficiency, and real-time nature of this form of communication.
- Difficulty of development. Difficulty of development is an important aspect of instant messaging chats because it affects the resources and expertise required to create and maintain the platform. Developing systems that use instant messaging requires specialized knowledge and skills in areas such as software engineering, networking, and security. The more difficult the development process, the more resources, time, and expertise are required, which can impact the cost and quality of the final product. Additionally, difficulty of development can also affect the speed of updates and improvements to the platform, as well as its ability to adapt to changing technologies and user needs. Therefore, the difficulty of development can impact the competitiveness and sustainability of IM platforms in the market. However, it is worth noting that while difficulty of development is important, it is not the only factor that determines the success and adoption of IM.

3. Instant messaging implementations

There are several different approaches to providing instant messaging. All of them fulfill their purpose and provide real-time communication between users: messages in chats appear as soon as the other person sends them, and the conversation can take place without delay. But each of the existing approaches implements this communication in a different way. The main types of instant messaging implementations are client pull and server push. Each of these approaches has several different implementations, which will be discussed in this paper.

The types of implementation this paper is going to focus on are:

- WebSocket-based messaging;
- Polling (short and long);
- Server-sent events.

Both short and long polling are methods that implement client pull approach. Client pull means that client initiates the update process and asks server for updates at certain regular intervals of time. Web-socket messaging and server-sent events, however, are implementations of a different approach called server push. It means, unlike client pull, that the updates are initiated by the server side, where server proactively sends them to the client.

Short polling is a type of communication between a client and a server in which the client repeatedly sends requests to the server to check for updates or new information. In short polling, the client sends a request to the server at regular intervals, asking for any updates or changes since the last request. The server responds immediately with any new information, and the client then processes that information and sends another request after a short interval.

Schematically short polling is presented on Figure 1.

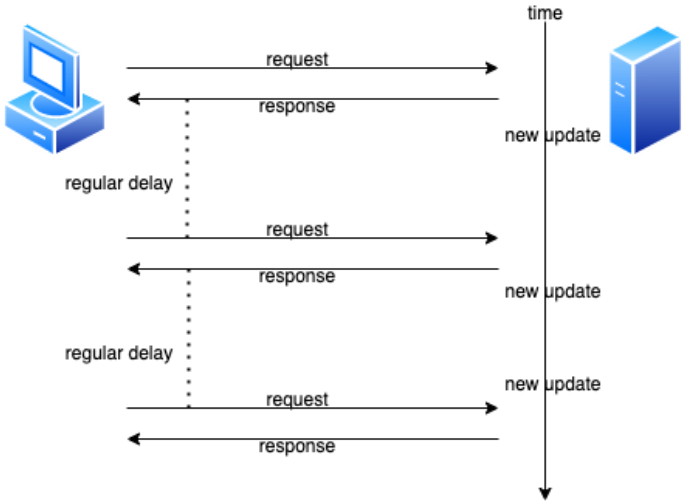

Figure 1: Short polling scheme

As shown on the figure 1, requests are sent after regular delay and not when new updates appear on the server. Short polling is implemented using AJAX-based timer. The frequency of sending requests for new data depends on the latency that the client can afford in retrieving changed information from the server. Server can either send an empty response or data object in its body. One of the drawbacks of this approach is that sending repeated requests to the server wastes resources as each new incoming connection must be established, the HTTP headers must be passed, a query for new data must be performed, and a response (that often doesn't contain new data) must be generated and delivered. The connection must be closed and any resources cleaned up [21].

Long polling is another implementation of client requesting updates from the server. It aims to somewhat fix the issue with wasting too many resources that short polling has. Unlike the previous approach, long polling send a request to the server and waits until the server has any updates to send back. This allows us to avoid cases in which multiple requests to the server don't return any changes.

Long polling attempts to minimize both the latency in server-client message delivery and the use of processing/network resources. The server achieves this by responding to a request only when a particular event that signals an update available, status, or timeout has occurred. Once the server sends a long poll response, typically the client immediately sends a new long poll request. This means that at any given time the server will be holding open a long poll request, to which it replies when new information is available for the client. As a result, the server is able to asynchronously "initiate" communication [22].

Schematically long polling is presented on Figure 2.

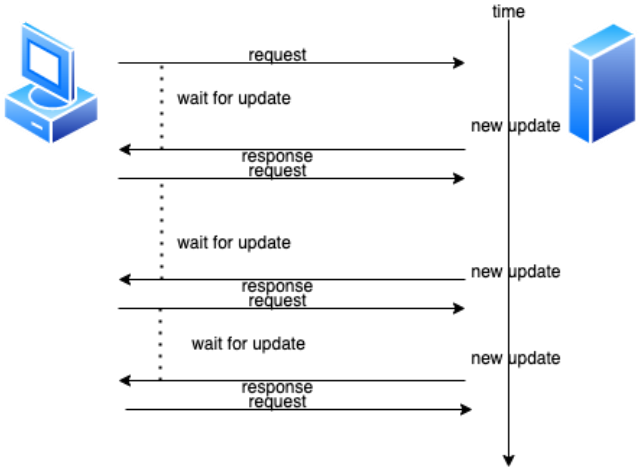

Figure 2: Long polling scheme

Long polling attempts to minimize both the latency in server-client message delivery and the use of processing/network resources. The server achieves this by responding to a request only when a particular event that signals an update available, status, or timeout has occurred. Once the server sends a long poll response, typically the client immediately sends a new long poll request. This means that at any given time the server will be holding open a long poll request, to which it replies when new information is available for the client. As a result, the server is able to asynchronously "initiate" communication.

Now let's look at the server push implementations. WebSocket is a technology that allows to open a two-way interactive communication session between a client and a server [23]. The protocol consists of an opening handshake followed by basic message framing, layered over TCP. The goal of this technology is to provide a mechanism for browser-based applications that need two-way communication with servers that does not rely on opening multiple HTTP connections [24]. WebSocket schema is presented on Figure 3.

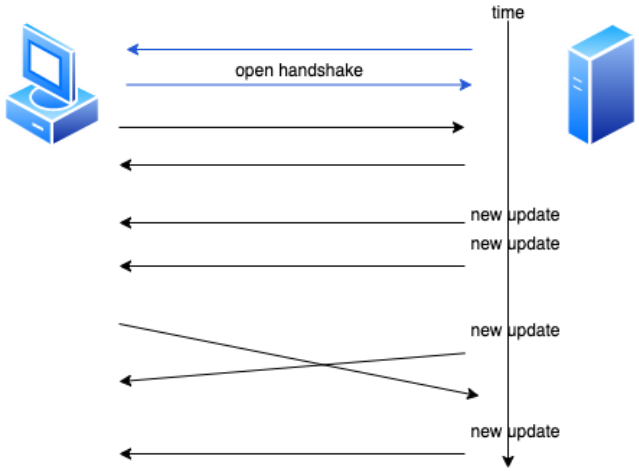

Figure 3: WebSocket scheme

Server sends updates to the client as soon as they appear, and in the meantime, client doesn't wait for a response and can send a new one. WebSockets provide a two-way ongoing conversation up until the connection is closed. They do not, however, operate on top of HTTP protocol – instead, they need their own TCP connection to work.

Another implementation of server push approach is called server-sent events. The goal of server-sent events is to give the browser a convenient way to receive data from the server without having to ask for it. It's a specification that defines how servers start sending data to clients after establishing a client connection. This method allows the server to push data in asynchronous way back to the client once the connection has been established. This can be described as a one way "publish-subscribe" model, where client "subscribes" to a specific event and watches its updates once the server sends them [25].

Schematically server-sent events are displayed on Figure 4.

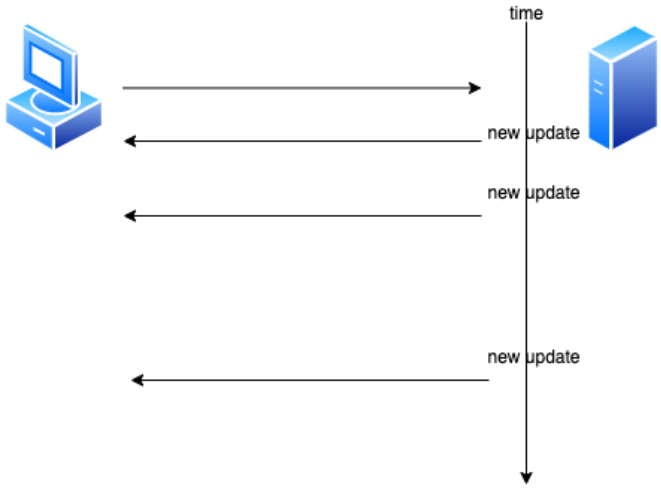

Figure 4: Server sent events scheme

Unlike WebSockets, server-sent events operate on top of HTTP protocol. Once the connection is established, the client registers the event source (a way of "subscribing" to updates). If connection with event source is lost, it can be automatically re-established.

The main difference between WebSockets and server-sent events is that WebSockets allows to have a two-directional communication between client and server meanwhile server-sent events are mono-directional [26].

Overall, the described methods can be summarized using a few key points of their implementations. Check Table 1 to see how they compare with each other.

Table 1

Key points of compared implementations

Method	Who retrieves data	Protocol	Principle of work
Short polling	Client pull	HTTP	Regular delay using AJAX timer
Long polling	Client pull	HTTP	Wait for the response
WebSocket	Server push	HTTP (initial connection) + WebSocket	Two-way communication
Server-sent events	Server push	HTTP	One-way communication

Now that the main approaches have been selected and described, we can define the methods which are going to be used for comparison and what criteria will we base the comparison on.

4. Methods and criteria for comparison

As previously mentioned, we can define three major points that can to be compared: security, speed,s and difficulty of development.

To compare security aspects, we need to analyze and specify main vulnerabilities and issues a web application can face. There is no straightforward approach to measure security of application, but we can use a risk assessment methodology that takes into account the severity of security issues and their likelihood of occurrence.

First, let's specify what security issues we will consider. For this purpose, we can use a standardized framework called OWASP. The OWASP Top 10 is a standard awareness document for developers and web application security. It represents a broad consensus about the most critical security risks to web applications [27]. For each security issue, we would need to consider the following:

- Issue impact – numeric value that represents the severity or priority of issue;
- Probability of occurrence – numeric value that represents the likelihood of appearance of an issue. It can be calculated considering spread of this issue in web environment, as well as if the current implementation of instant messaging has a vulnerability for this issue.

There are many things we need to take into account while calculating these points, but a high-level formula to calculate security risk score can be written as following:

$$Sec = \sum Imp(i) * Prb(i), \tag{1}$$

where *Sec* – security risk score,
Imp – issue impact,
Prb – probability of occurrence,
i – security issue.

When talking about instant messaging in real-time web applications, speed is crucial. It directly affects users' experience and satisfaction, as well as defines messaging latency.

We can distinguish the following factors when considering speed of an instant messaging application:

- Message round trip time: this refers to the time it takes for a message to be sent from one user to another and for the acknowledgement to be received. This includes the time it takes for the message to travel over the network and any delays introduced by the client and server. The lower the message round trip time, the faster the application.
- Server response time: This refers to the time it takes for the server to receive and process incoming messages and send responses back to the clients. The lower the server response time, the faster the application.
- Client processing time: This refers to the time it takes for the client to receive incoming messages and process them before displaying them to the user. This includes any time spent decoding messages, rendering them in the user interface, and updating the message history. The lower the client processing time, the faster the application.

We will be considering all three of these factors to have an equal priority. Based on that, we can formulate a high-level formula to calculate the speed score for an application:

$$Spd(app) = \frac{Mrt(app) + Srt(app) + Cpt(app)}{3}, \tag{2}$$

where *Spd* – application speed score,
Mrt – message round trip time,
Srt – server response time,
Cpt – client processing time,
app – application.

Lastly, to measure difficulty of development for an application we would also need to specify influential factors (such as availability of frameworks for development, browser support, popular

programming languages support, number of available libraries etc.) and use them to calculate a score for each of the implementations we want to compare.

For the purpose of this research, we are going to focus on measuring speed score for each of the implementations mentioned in section 3.

5. Experiment

While conducting the experiment, we are going to focus on measuring speed criteria of the following instant messaging implementations: short polling, long polling, WebSocket, server-sent events.

Measuring the speed of an instant messaging web application can help identify potential performance issues that may affect the user experience. The purpose of measuring speed is to determine how quickly messages are delivered between users and the server, and how quickly the application responds to user actions such as sending or receiving messages, displaying notifications, and updating the user interface.

The goal of measuring the speed of an instant messaging web application in a real-world environment is to ensure that users have a seamless and fast messaging experience, which can lead to increased user satisfaction and engagement.

First, we need to prepare our testing environments. For each of the abovementioned approaches, we are going to create a small web application that supports minimal functionality of sending and receiving messages.

We take into account factors, mentioned in section 4: message round trip time, server response time, and client processing time.

In order to measure the effectiveness of each of the indicators, it is necessary to conduct initial measurements and long-term measurements. Initial measurements are those obtained when the system is first launched. Long-term measurements are the indicators measured during further interaction with the system.

We are going to use two values for each of the factors: the initial measurement taken during the first launch, and the long-term measurement which is going to be calculated as an average between values we got during the next three runs.

To implement testing environments, we are going to use python library called Flask. Flask is a web framework that provides libraries to build lightweight web applications in python. It is based on WSGI toolkit and jinja2 template engine. Flask is considered as a micro framework [28]. We also need to identify additional libraries we are going to use to ensure the implementation of each approach.

We are going to create a client and a server for each of the applications and ensure their support of used implementation.

To create client that supports short polling:

- Use fetch API – a JavaScript interface for accessing and manipulating parts of the protocol, such as requests and responses. It also provides a global fetch() method that provides an easy, logical way to fetch resources asynchronously across the network [29];
- Implement a getLatestMessages() function that is going to send a GET request to the server to retrieve chat messages;
- Implement a sendMessage() function that is going to send a POST request to the server to preserve a new message;
- Use setInterval() to call getLatestMessages() with a fixed delay for updates.

To create a server side for short polling:

- Implement api endpoint “/api/get_latest_messages” that supports GET requests and returns a list of available chat messages;
- Implement api endpoint “/api/send_message” that supports POST requests with message data and preserves new messages.

In order to create a web application that uses long polling, we need to create a new client that also supports fetch API and provides getLatestMessages() and sendMessage() functionality, but instead of using timer to send s new GET request, we send a new GET request on success of the previous one.

To implement a WebSocket instant messaging app, we are going to use SocketIO library. Socket.IO is a library that enables low-latency, bidirectional and event-based communication between a client and

a server. It is built on top of the WebSocket protocol and provides additional guarantees like fallback to HTTP long polling or automatic reconnection [30].

To write a client that supports WebSocket implementation, we also need:

- Implement event listeners “status” and “message”;
- Display new messages once the server sends an update.

On the server side, we need to use an additional flask library that supports SocketIO. We need to add the following logic:

- Implement event listeners “text” when client sends a new text;
- Emit updated list of messages for client to process.

To create an application that uses server sent events, we need to create a stream of data on the server side and assign client to listen to it. On each new message we should send a request that would trigger a server-sent event and update list of all chat messages.

We are also going to define a custom MessageAnnouncer class that implements the following methods:

- listen() – register listeners;
- announce – send the incoming message to queue.

Once the testing environments are ready, we need a way to measure time for previously specified criteria. We need to identify a representative use case to base measurements on. Since the applications don’t have a lot of functionality, we are going to use a basic scenario “User opens the chat and sends a message”.

The following rules should be applied during measurements:

- Message round trip time – the time between the first click of a “Send message button” until the new message is displayed in the dialog window;
- Server response time – the time between when server accepts a new message until an acknowledgement is sent to the client;
- Client processing time – the time between when a new message arrives to the client until it’s displayed in the chatroom;
- Application speed score – value calculated using formula (2).

For measuring time we are going to use built-in libraries and their methods: datetime.time() and performance.now(). The performance.now() method is a JavaScript API that provides high-resolution timestamps for measuring performance [31]. To measure the speed of an application using performance.now(), we can follow these steps:

- Identify the code to be measured: Identify the specific code that we want to measure for performance. This could be a function, a block of code, or an entire script;
- Insert performance.now() calls: Insert performance.now() calls at the start and end of the code to be measured. This will capture the timestamp at the beginning and end of the code block.
- Calculate the elapsed time: Subtract the start timestamp from the end timestamp to calculate the elapsed time. This will give us the time taken for the code to execute.
- Repeat the measurement: To get a more accurate measurement, repeat the measurement several times and calculate the average elapsed time.

Once we have completed to plan the experiment, we can move on to performing the measurements.

6. Results

Once the testing applications have been created, we can start measuring speed performance for each of selected implementation methods.

After we run the application that uses short polling method, we document the results we get on the first run and on the next three consecutive requests. These measurements are presented in Table 2.

Using these values, we are going to calculate a long-term measurement value by taking the results of consecutive runs and diving them by number of runs:

- avg(message round trip time) = 37,466 ms
- avg(server response time) = 3 ms
- avg(client processing time) = 40,846 ms

These values are added to the Table 2 as well.

Table 2
Speed results for short polling

Number of run	Message round trip time, ms	Server response time, ms	Client processing time, ms
Initial	940,299	4	945,300
2nd	16,400	2	19
3rd	51,700	3	55,100
4th	44,299	4	48,440
Long-term	37,466	3	40,846

Using results of measurement for short polling, we can calculate application speed score with formula (2):

- Spd(initial) = 629,866 ms
- Spd(long-term) = 81,312 ms

Next, we are going to measure speed of application implemented using long polling technique. The measurements we got after running the chat application and sending messages are presented in Table 3.

Table 3
Speed results for long polling

Number of run	Message round trip time, ms	Server response time, ms	Client processing time, ms
Initial	36,8	5	43,2
2nd	7,5	4	11,8
3rd	10,3	3	13,9
4th	1,09	4	6,02
Long-term	6,29	3,6	10,57

In the same way we are going to calculate a long-term measurement value by taking the results of consecutive runs and diving them by number of runs:

- avg(message round trip time) = 6,29 ms
- avg(server response time) = 3,6 ms
- avg(client processing time) = 10,57 ms

Application speed score for short polling is as follows:

- Spd(initial) = 21,09 ms
- Spd(long-term) = 6,82 ms

Next, let's measure speed of an application that uses WebSocket implementation of a chat application.

The results of measurement are presented in Table 4.

Table 4
Speed results for WebSocket

Number of run	Message round trip time, ms	Server response time, ms	Client processing time, ms
Initial	7,199	0,0002	7,2
2nd	4,69	0,00009	4,7
3rd	4,4	0,00001	4,4
4th	4,59	0,00001	4,6
Long-term	4,56	0,0003	4,56

Let's calculate long-term measurement in the same way as previous ones:

- avg(message round trip time) = 4,56 ms
- avg(server response time) = 0,0003 ms
- avg(client processing time) = 4,56 ms

Application speed score for short polling is as follows:

- Spd(initial) = 4,799 ms
- Spd(long-term) = 3,041 ms

Finally, we are going to measure speed of server-sent events implementation. The resulting numbers are presented in Table 5.

Table 5
Speed results for server sent events

Number of run	Message round trip time, ms	Server response time, ms	Client processing time, ms
Initial	10,218	0,40	10,7
2nd	7,553	0,08	7,7
3rd	6,901	0,12	7,01
4th	7,315	0,18	7,430
Long-term	7,256	0.13	7,38

Calculated long-term measurements:

- avg(message round trip time) = 7,256 ms
- avg(server response time) = 0,13 ms
- avg(client processing time) = 7,38 ms

Application speed score for short polling is as follows:

- Spd(initial) = 7,1 ms
- Spd(long-term) = 4,922 ms.

After all of the measurements were taken, we can present final data in resulting Table 6.

Table 6
Resulting table

Implementation	Initial speed score, ms	Long-term speed score, ms
Short polling	629,866	81,312
Long polling	21,09	6,82
WebSocket	4,799	3,041
Server-sent events	7.1	4,922

Once we finished experiment and got results on different speed scores for each of the instant messaging approaches, we can proceed and analyze what they mean.

7. Discussions

While conducting the experiment, we compared four different ways to implement instant messaging technique in terms of their speed score, both on initial run and the consecutive (long-term) runs. The results were summarized and presented in Table 6.

Overall across different implementations we can see that server response time in general tends to be lesser than client processing, regardless of which side initiated the data update (client pull or server push). We can also see that numbers on the initial run tent to be bigger than those on long-term run. This is explained by the fact that we take additional time to establish the connection initially. There can be multiple contributing factors that make the initial run slower than the rest, among them the following:

- Server startup time: when a web application is first started, the server may need to perform initialization tasks, such as connecting to a database or loading configuration settings. These tasks may take some time to complete, which can result in a longer initial load time;
- Network latency: the first time a user accesses a web application, their browser needs to establish a connection with the server. This process can take some time, especially if there is high network latency or if the server is located far away from the user;
- Browser optimization: ss users interact with a web application, their browser can optimize performance by preloading resources, prioritizing critical rendering path, or running faster interpretation of JavaScript, which can make subsequent runs of the application faster.

In terms of speed, we can see that implementations that use server push approach tend to perform better than those using client pull. Among all implementations, WebSocket based messaging showed the best results. On the other hand, short polling shows the worst results by far – it can be influenced by the fact that this approach uses a fixed delay in between requests.

Good speed in real-world instant messaging apps provides several advantages, including:

- Quick communication: a fast instant messaging app allows users to send and receive messages quickly, which can be beneficial in time-sensitive situations where quick responses are necessary;
- Increased productivity: faster messaging speeds can increase productivity as users can communicate more efficiently and get more work done in a shorter amount of time;
- Better user experience: a speedy app can provide a better user experience as it reduces waiting times and ensures that messages are delivered and received without delay;
- Improved engagement: fast messaging can encourage users to engage more frequently with the app as they are more likely to respond to messages quickly and keep the conversation going;
- Reduced frustration: slow messaging speeds can be frustrating for users, leading to a negative experience with the app. A fast messaging app reduces the likelihood of frustration and improves overall satisfaction with the app.

Overall, a fast messaging app provides several benefits that can enhance user experience, improve productivity, and increase engagement with the app.

Aside from speed, there is another factor we didn't consider during the experiment run that makes client pull implementations lose in comparison to server push ones. Both short and long polling send a huge number of requests per minute, which in long term affects applications' performance, overuses resources and puts additional load on server.

Based on the results of the experiment, we can conclude that the optimal instant messaging implementation in terms of speed is WebSocket. However, one should keep in mind other comparison criteria that was mentioned in this paper but wasn't included in the experiment, such as security and difficulty of development. Those can be additionally measured and investigated further for WebSocket approach, as well as its runner up server-sent events, to verify if the WebSocket based messaging is indeed the best option to go with.

8. Conclusions

In conclusion, this paper provides readers with an introduction of four different approaches to implementing instant messaging web systems. It gives us a description of what instant messaging is and what advantages it has in comparison with other forms of online communication.

The study presents main approaches to implementing chat systems, describes the principals on which they work, points out similarities and differences between them. Aside from that, the paper selects measurable criteria based on which these methods can be compared and provides justifications for using each of these parameters. The key factors selected for comparison were:

- Speed;
- Security;
- Difficulty of development.

This paper proposes approaches to measure each of these factors, suggests criteria and high-level formulas that can be used for calculating a numeric value of each factor.

Afterwards, an experiment was deducted to measure speed score of each implementation in different testing environments using the same use case. As a result, the WebSocket approach was named an overall best solution speed-wise. However, it is recommended to conduct additional research outside of this paper using the remaining criteria to verify the results.

The key takeaways are:

- Server push implementations tend to have higher speed than client pull ones;
- Client pull implementations might potentially overload the system by sending too many requests for updates;
- WebSocket approach tends to perform slightly better than server-sent events, however the difference in numbers is not significant, so additional investigation might be conducted.

The results of this research can be used when deciding on which approach to use when developing a new instant messaging application, as well as get a general introduction in existing implementations and their main differences.

Furthermore, the research described in this paper can be continued using suggested approaches to specify new key factors and conduct comparison using them. The proposed formulas can be supplemented with additional criteria and weight factors. The results might be used in web application development cycle, as well as expanded to other types of applications, i.e. mobile or desktop ones.

9. References

[1] J. Botha, C. van ’t Wout, L. Leenen, A Comparison of Chat Applications in Terms of Security and Privacy, University of Coimbra, Portugal, 2019.

[2] R. M. Ali, S. N. Alsaad, Instant messaging security and privacy secure instant messenger design, Vol. 881: IOP Conf. Series: Materials Science and Engineering, 2020.

[3] G. A. Pimenta Rodrigues et al., "Securing Instant Messages With Hardware-Based Cryptography and Authentication in Browser Extension," in IEEE Access, vol. 8, pp. 95137-95152, 2020, doi: 10.1109/ACCESS.2020.2993774.

[4] E. Sahafizadeh and B. Tork Ladani, "A Model for Social Communication Network in Mobile Instant Messaging Systems," in IEEE Transactions on Computational Social Systems, vol. 7, no. 1, pp. 68-83, Feb. 2020, doi: 10.1109/TCSS.2019.2958968.

[5] J. Wei, X. Chen, J. Ma, X. Hu and K. Ren, "Communication-Efficient and Fine-Grained Forward-Secure Asynchronous Messaging," in IEEE/ACM Transactions on Networking, vol. 29, no. 5, pp. 2242-2253, Oct. 2021, doi: 10.1109/TNET.2021.3084692.

[6] M. S. Al-Faaruuq and D. F. Priambodo, "iOS Digital Evidence Comparison of Instant Messaging Apps," 2022 International Conference of Science and Information Technology in Smart Administration (ICSINTESA), Denpasar, Bali, Indonesia, 2022, pp. 83-88, doi: 10.1109/ICSINTESA56431.2022.10041620.

[7] S. Rammos, M. Mundra, G. Xu, C. Tong, W. Ziolkowski and I. Malavolta, "The Impact of Instant Messaging on the Energy Consumption of Android Devices," 2021 IEEE/ACM 8th International Conference on Mobile Software Engineering and Systems (MobileSoft), Madrid, Spain, 2021, pp. 1-11, doi: 10.1109/MobileSoft52590.2021.00007.

[8] D. Saxena and A. K. Singh, "OSC-MC: Online Secure Communication Model for Cloud Environment," in IEEE Communications Letters, vol. 25, no. 9, pp. 2844-2848, Sept. 2021, doi: 10.1109/LCOMM.2021.3086986.

[9] J. Gao, C. Zhong, G. Y. Li and Z. Zhang, "Online Deep Neural Network for Optimization in Wireless Communications," in IEEE Wireless Communications Letters, vol. 11, no. 5, pp. 933–937, May 2022, doi: 10.1109/LWC.2022.3149863.

[10] K. Smelyakov, A. Datsenko, V. Skrypka and A. Akhundov, "The Efficiency of Images Reduction Algorithms with Small-Sized and Linear Details," 2019 IEEE International Scientific-Practical Conference Problems of Infocommunications, Science and Technology (PIC S&T), 2019, pp. 745-750, doi: 10.1109/PICST47496.2019.9061250.

[11] Shubin, I., Kyrychenko, I., Goncharov, P., Snisar, S., "Formal representation of knowledge for infocommunication computerized training systems," 2017 IEEE 4th International Scientific-

Practical Conference Problems of Infocommunications, Science and Technology (PIC S&T), 2017, pp. 287–291, doi: 10.1109/INFOCOMMST.2017.8246399.
[12] Gruzdo, I., Kyrychenko, I., Tereshchenko, G., Shanidze, N., "Metrics applicable for evaluating software at the design stage," 2021 5th International Conference on Computational Linguistics and Intelligent Systems (COLINS-2021), 2021. – CEUR-WS, 2021, ISSN 16130073. – Volume 2870, PP. 916–936.
[13] H. Z. Jahromi, D. T. Delaney and A. Hines, "Beyond First Impressions: Estimating Quality of Experience for Interactive Web Applications," in IEEE Access, vol. 8, pp. 47741–47755, 2020, doi: 10.1109/ACCESS.2020.2979385.
[14] F. Ö. Sönmez and B. G. Kiliç, "Holistic Web Application Security Visualization for Multi-Project and Multi-Phase Dynamic Application Security Test Results," in IEEE Access, vol. 9, pp. 25858-25884, 2021, doi: 10.1109/ACCESS.2021.3057044.
[15] L. de la Torre, J. Chacon, D. Chaos, S. Dormido, “Using Server-Sent Events for Event-Based Control in Networked Control Systems”, IFAC-PapersOnLine. vol. 9, pp. 260-265, 2019, doi: 10.1016/j.ifacol.2019.08.218.
[16] A. Herzberg, H. Leibowitz, K. Seamons, E. Vaziripour, J. Wu and D. Zappala, "Secure Messaging Authentication Ceremonies Are Broken," in IEEE Security & Privacy, vol. 19, no. 2, pp. 29–37, March–April 2021, doi: 10.1109/MSEC.2020.3039727.
[17] Fester, A. & Horvath, G. (2022). Instant messaging and the facilitation of collaborative, student-led learning and teacher-support: the NZCEL EAP scenario. TESOL Journal, 13, e691. https://doi.org/10.1002/tesj.691
[18] E. Arias, M. Arellano and M. Cuadros, "Optimization and improvement of bidirectional connections with Web Socket on Synapse framework using Web Workers technology," 2021 IEEE Sciences and Humanities International Research Conference (SHIRCON), Lima, Peru, 2021, pp. 1–4, doi: 10.1109/SHIRCON53068.2021.9652363.
[19] K. Smelyakov, A. Chupryna, M. Hvozdiev and D. Sandrkin, "Gradational Correction Models Efficiency Analysis of Low-Light Digital Image," 2019 Open Conference of Electrical, Electronic and Information Sciences (eStream), 2019, pp. 1-6, doi: 10.1109/eStream.2019.8732174.
[20] T. Enghardt, P. S. Tiesel, T. Zinner and A. Feldmann, "Informed Access Network Selection: The Benefits of Socket Intents for Web Performance," 2019 15th International Conference on Network and Service Management (CNSM), Halifax, NS, Canada, 2019, pp. 1–9, doi: 10.23919/CNSM46954.2019.9012714.
[21] A. Aggarwal. Short Polling vs Long Polling vs Web Sockets, 2021. URL: https://anuradha.hashnode.dev/short-polling-vs-long-polling-vs-web-sockets/.
[22] Loreto, S., Saint-Andre, P., Salsano, S., and G. Wilkins, "Known Issues and Best Practices for the Use of Long Polling and Streaming in Bidirectional HTTP", RFC 6202, DOI 10.17487/RFC6202, 2011. URL: https://www.rfc-editor.org/info/rfc6202.
[23] MDN Web Docs. The WebSocket API (WebSockets), 2023. URL: https://developer.mozilla.org/en-US/docs/Web/API/WebSockets_API.
[24] Fette, I. and A. Melnikov, "The WebSocket Protocol", RFC 6455, DOI 10.17487/RFC6455, 2011. URL: https://www.rfc-editor.org/info/rfc6455.
[25] B. Ganesan. Polling vs SSE vs WebSocket– How to choose the right one, 2018. URL: https://codeburst.io/polling-vs-sse-vs-websocket-how-to-choose-the-right-one-1859e4e13bd9.
[26] P. L. Gunawardhana. WebSockets vs. Server-Sent Events, 2022. URL: https://blog.bitsrc.io/websockets-vs-server-sent-events-968659ab0870.
[27] OWASP. OWASP Top Ten, 2021. URL: https://owasp.org/www-project-top-ten/.
[28] JavaTPoint. Python Flask Tutorial. URL: https://www.javatpoint.com/flask-tutorial.
[29] MDN Web Docs. Using the Fetch API, 2023. URL: https://developer.mozilla.org/en-US/docs/Web/API/Fetch_API/Using_Fetch.
[30] SocketIO. Introduction. What SocketIO Is, 2023. URL: https://socket.io/docs/v4/.
[31] MDN Web Docs. https://developer.mozilla.org/en-US/docs/Web/API/Performance/now, 2023. URL: https://developer.mozilla.org/en-US/docs/Web/API/Performance/now.