Journal of Advanced Zoology

ISSN: 0253-7214

Volume 44 Issue S-7 Year 2023 Page 322:330

Performance Evaluation of WebRTC-Based Video Conferencing: A Comprehensive Analysis

Shyam Sunder Saini[1*], Lalit Sen Sharma[2]

[1*, 2]*Department of Computer Science and IT, University of Jammu, Baba Saheb Ambedkar Road, Jammu, 180006, India, sundershyam51@gmail.com*

**Corresponding Author Email: sundershyam51@gmail.com*

Article History

Received: 23 June 2023
Revised: 17 Sept 2023
Accepted: 13 Dec 2023

CC License
CC-BY-NC-SA 4.0

Abstract

In an ever-evolving technological landscape, addressing the performance challenges of real-time communication protocols is crucial. Real-time communication, facilitated by streaming media protocols, utilizes peer-to-peer or client-server models to enhance Quality of Service (QoS). WebRTC (Web Real-Time Communication) stands as a widely adopted, browser-based, open-source, peer-to-peer protocol, offering real-time media transmission through JavaScript APIs without third-party plugins. This paper presents an in-depth performance evaluation of a WebRTC-based video conferencing system using Socket.io services on a Node.js server. Our research expands on recent studies by introducing a comprehensive set of performance parameters, including Processing delay, CPU Utilization, Latency, Jitter, and Packet Loss, and packet delay. Our findings indicate that WebRTC performs exceptionally well within specific latency thresholds. However, scalability concerns emerge when a large number of clients are introduced, especially in bandwidth-constrained environments.

Keywords: Real-Time Communication; Socket.io; Peer-to-Peer; Scalability

1 Introduction

In an age marked by rapid technological progress, the domain of real-time communication stands as a linchpin of modern connectivity. The ability to communicate seamlessly and instantly across geographical divides has reshaped the way we engage, share information [1], and conduct business. At the heart of this transformation lie streaming media protocols, pivotal tools that facilitate the efficient transmission of audio, video, and other multimedia data across networks. This paper embarks on a comprehensive exploration of the Performance Evaluation of Video Conferencing using WebRTC (Web Real-Time Communication) Streaming Media Protocol—a crucial area of study in today's landscape [2].

The significance of real-time communication in our interconnected world cannot be overstated. It underlies numerous facets of our daily lives, from virtual meetings and online gaming to telemedicine and remote collaboration. The capacity to exchange information instantly, mirroring face-to-face interactions, has become indispensable for both personal and professional communication. Real-time communication transcends mere convenience; it has become a necessity [3].

Real-time communication has witnessed substantial evolution in the twenty-first century, with a notable surge during the COVID-19 pandemic. Video conferencing emerged as a predominant mode of communication, adopting various networking models, notably the peer-to-peer model and the Client-server model [4]. These models represent innovative approaches to address the evolving demands of real-time communication in a rapidly changing world. WebRTC, discussed later, employs the peer-to-peer model for real-time communication [5, 6].

Central to the efficacy of real-time communication are streaming media protocols, which facilitate the transmission of multimedia content, including audio and video, across the internet and other networks [7]. These protocols govern the mechanisms for data delivery, encoding, and decoding, ensuring that users can

experience audiovisual content seamlessly and without disruptions. Without efficient streaming media protocols, the quality of real-time communication experiences would deteriorate, leading to issues such as buffering, lag, and subpar audiovisual quality.

WebRTC, as an open-source framework, directly integrates with web browsers, eliminating the need for third-party software or plugins. In scenarios where network congestion blocks UDP ports, WebRTC employs a custom-designed congestion control algorithm to manage such situations adeptly. During connection establishment, all involved users exchange crucial Session Description Protocol (SDP) information through a centralized signaling server, primarily for sharing user-specific details [8]. This signaling server plays a pivotal role in exchanging vital information, including IP addresses, codes, bandwidth details, media types, and other essential data among users, ensuring uninterrupted communication.

The overarching objective of this research is to conduct a comprehensive evaluation of the performance of video conferencing systems reliant on the WebRTC streaming media protocol. WebRTC, a browser-based, open-source technology, has garnered widespread adoption due to its capacity to enable real-time communication without necessitating third-party plugins or extensions. It operates seamlessly within web browsers, harnessing the capabilities of JavaScript APIs (Application Programming Interfaces) to facilitate direct peer-to-peer communication [9].

However, despite the growing prevalence of WebRTC, critical performance considerations necessitate investigation. The primary research problem centers on comprehending the performance of WebRTC video conferencing systems under diverse conditions and loads. This entails evaluating factors like Total Assembly Time per frame, Processing delay, CPU Utilization, Latency, Jitter, packet loss, and Total decoding time [10]. Additionally, the study aims to illuminate the scalability of WebRTC-based systems. As the number of users participating in real-time communication sessions escalates, potential challenges associated with bandwidth utilization and system resource allocation may surface. Thus, it becomes imperative to scrutinize how effectively WebRTC systems address scalability concerns.

This paper is structured into several sections to facilitate a comprehensive exploration of the Performance Evaluation of Video Conferencing using WebRTC Streaming Media Protocol. The paper's organization is as follows: Section II provides a foundation in theoretical background. Section III conducts an extensive review of existing literature concerning real-time communication, streaming media protocols, and WebRTC. Section IV, The Methodology section, elucidates the research approach and methods employed in this study. Section V presents the results of the performance evaluation, and finally, Section VI interprets these results within the context of the research objectives.

Theoretical Background

In this comprehensive exploration of the theoretical underpinnings of WebRTC (Web Real-Time Communication), we delve into the fundamental concepts and protocols that constitute the bedrock of real-time communication systems. These insights lay the essential groundwork for comprehending the subsequent sections of this research.

WebSocket Protocol

WebSocket is a full-duplex communication protocol that operates atop the TCP (Transmission Control Protocol) and is standardized by the Internet Engineering Task Force (IETF) under RFC 6455. It facilitates full-duplex communication, allowing simultaneous data exchange between client and server. This is a departure from the conventional HTTP protocol, which serves client requests in a half-duplex manner [11]. Notably, WebSocket enjoys support from major web browsers, including Chrome, Firefox, Safari, and more, making it a cornerstone of real-time web communication [12].

Signalling Server

The Signalling Server assumes a critical role in the realm of WebRTC applications, serving as the linchpin for establishing connections between users. Its significance lies in its resource-efficient approach, employing the TCP protocol to share essential information, as opposed to UDP, which is reserved for media data transmission. Notably, this service is not explicitly mentioned within the WebRTC API, as WebRTC primarily focuses on the media transfer aspect. Operating in a centralized manner, the Signalling Server orchestrates crucial functions, including tracking the online user count, managing the transmission of error and control messages, and overseeing the initiation, reception, and termination of calling services.

Session Description Protocol (SDP)
Within the intricate WebRTC ecosystem, the Session Description Protocol (SDP) emerges as a cornerstone element for conveying vital parameters during connection establishment. These parameters encompass critical information, such as media codec specifications (e.g., Opus/VP8), network connectivity details, and codec specifics. SDP utilised for exchanging media information which is to be negotiated in streaming (See Fig. 1).

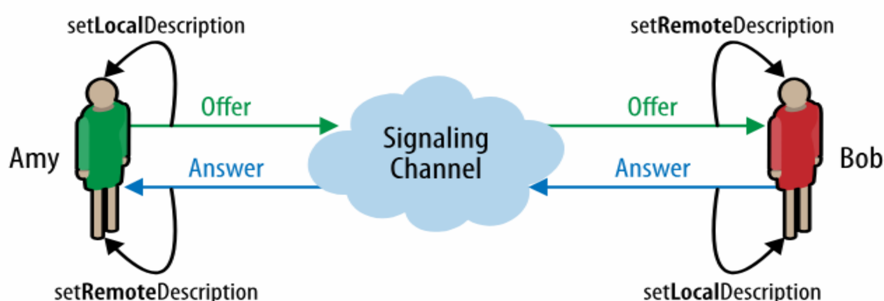

Fig. 1. Offering and answering of SDP Information among peers

WebRTC Protocols
WebRTC relies upon a suite of protocols to ensure the reliability, security, and efficient flow control of media data (See Fig. 2). At the core of WebRTC's data transmission lies the User Datagram Protocol (UDP) [13]. However, the complexity of real-time communication necessitates additional protocols to bolster security and maintain data integrity. During data transmission, the Datagram Transport Layer Security (DTLS) protocol is employed to safeguard the data. DTLS enhances UDP security by implementing Transport Layer Security (TLS) at the transport layer, effectively thwarting eavesdropping and data tampering. Secure Real-Time Transport Protocol (SRTP) comes into play for key generation and media stream encryption, ensuring secure data transfer via UDP.

To maintain the proper order of packets during transmission, timestamps and sequence numbers are incorporated for the receiver's benefit, facilitating accurate sequence reconstruction. Error control is effectively managed by the Secure Real-Time Transport Control Protocol (SRTCP), which operates in a reverse direction—flowing from the receiver back to the sender. SRTCP keeps meticulous records of the sequence numbers of the last packet received, instances of packet loss, jitter, and latency during the transmission of media data. In addition to these vital protocols, WebRTC also harnesses the Stream Control Transmission Protocol (SCTP). SCTP integrates TCP-like features such as Flow Control, multiplexing, and congestion control with UDP to ensure the ordered delivery of packets.

With these formal introductions and a clear presentation of each element, we have established a comprehensive theoretical foundation that sets the stage for the subsequent sections of this research.

Literature Review
In the realm of WebRTC, a peer-to-peer protocol for real-time streaming media, extensive research has been conducted, exploring various techniques and algorithms. The findings of different researchers can be summarized as follows:

Prior to the advent of WebRTC, communication between systems posed a significant challenge. Historically, systems communicated through public switched telephone networks (PSTN), requiring unique identity registration. This approach necessitated third-party applications, often raising security concerns. WebRTC eliminates the need for third-party service registration, enhancing security. Research has demonstrated that WebRTC ensures security through DTLS (Datagram Transport Layer Security) and SRTP (Secure Real-Time Protocol), safeguarding communication from eavesdropping during media packet streaming.

One of the early studies [14] provided a comprehensive evaluation of the initial version of WebRTC. Experimental results revealed that latency beyond a certain threshold led to packet drops, and uneven bandwidth utilization resulted in network congestion during connection setup. Another group of researchers [15] implemented a decentralized system to enhance performance compared to server-based solutions, offering a cost-effective approach for video streaming. Research has also explored WebRTC compatibility with various

browsers, proposing architectures to address compatibility issues with legacy browsers. However, while the architectural concepts were well-explained, implementation details were lacking.

A.Heikkinen et.al [16] implemented a complete decentralized system for WebRTC and aimed to reduce packet redundancy among clients in conferencing scenarios. They also integrated protocols like SIP, XMPP, and XHR with WebSockets to improve compatibility. Nevertheless, communication via WebSockets in full-duplex mode consumed substantial resources as connections remained open until manually closed.

D. Mauro and M. Longo [17] performed an experiment to evaluated WebRTC protocol performance on mobile devices, considering parameters like battery life, hardware utilization, and bandwidth consumption. Results indicated that WebRTC consumed fewer hardware resources and extended battery life. Another experiment was conducted by K.Ng et.al [18] on mobile devices using a 3G network in the NS-3 simulation environment involving four users and a NodeJS server. Scalability for a larger number of users was left for future exploration.

N.Edan et.al [19] proposed a novel approach for initiating connections with various network topologies, including star and mesh for one-to-many and many-to-many connections. They developed the WebNSM protocol and conducted experiments on LAN and WAN for performance evaluation. Results showed promise for one-to-one connections but revealed resource-intensive bandwidth usage and network congestion in many-to-many scenarios.

G.Suciu et.al [20] assessed noise caused by congestion and compared SIP and WebRTC-based video and audio communication. SIP was developed using C#, and WebRTC was implemented using JavaScript deployed on a NodeJS server. Results demonstrated that WebRTC had a lower PSNR (Peak Signal-to-Noise Ratio) compared to SIP. Future experiments, focused on protocol capabilities, were conducted by E.Emmanuel et.al [6]. These experiments, conducted on a NodeJS server with higher-speed communication media channels, revealed that WebRTC proved to be a scalable protocol for browser-based streaming. Additionally, thermal cameras were explored for measuring body temperature to identify COVID-19 symptoms using WebRTC data streaming.

2 Materials and Methods

In the WebRTC framework, prior to establishing connections among users for video conferencing, a virtual room must be created. This room serves as a unique primary space where users can gather to engage in video conferencing sessions. The generation of a unique room ID is facilitated through the utilization of the 'Peer-Id' Library, which is available through the Node Package Manager (NPM). This room ID is initially generated for the first user, and subsequently, other users can join the room to participate in video conferencing sessions.
Once the virtual room is created, the next step involves the sharing of user information among participants. This is achieved through signaling mechanisms utilizing various WebRTC components, including Media Stream, RTCPeerConnection, RTCDataChannel, and the getStats API. These components collectively enable the exchange of essential data among users, paving the way for seamless communication.

Connecting users directly to one another is a pivotal aspect of WebRTC, achieved through Interactive Connectivity Establishment (ICE) candidates. However, this process is not without challenges, particularly when users are located in different networks. The traversal through intermediate routers can be a cumbersome task, primarily due to the imposition of firewall rules on networking devices.

To overcome these challenges, Network Address Translation (NAT) mechanisms come into play. NAT is employed to connect two different networks, and it becomes imperative to integrate STUN (Session Traversal Utilities for NAT) and TURN (Traversal Using Relays around NAT) servers within WebRTC. These servers are essential in addressing the problem of blocking through intermediate routers when communicating with other clients in video conferencing scenarios.

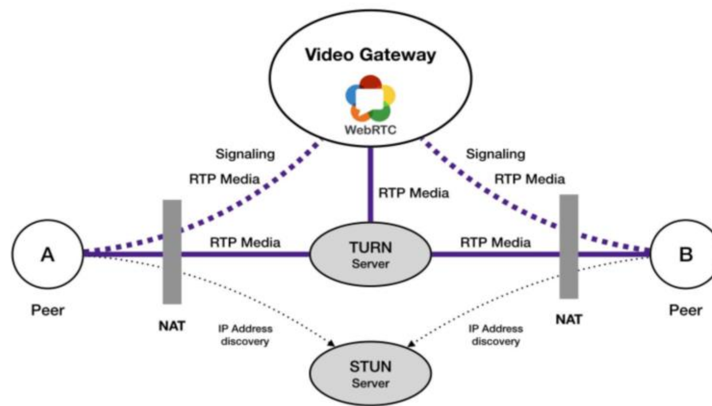

Fig.2. Working of Signalling Server for NAT traversal

In essence, NAT traversal in WebRTC is accomplished through the use of STUN and TURN servers. STUN (Session Traversal Utilities for NAT) operates at the browser level and connects with another user's STUN server. In rare cases where direct communication via the STUN server is not feasible, a centralized TURN (Traversal Using Relays around NAT) server is employed to share ICE candidate data. This ensures that every user intending to join the room possesses the necessary information regarding other users' ICE candidates, thus enabling the formation of direct end-to-end connections.

Once this groundwork is laid, the users are equipped to share media streams. Media negotiation includes checking compatibility between users also comes into play, facilitating the exchange of crucial information such as media codecs, compression and decompression formats, and more. Each user is required to generate an offer and respond with an answer to establish compatibility.

Once the connection is established, users can seamlessly share media packets through their browsers in a peer-to-peer (P2P) manner, obviating the need for dedicated servers. The proposed methodology is visualized in Fig. 3. Algorithm 1 describes pseudocode of WebRtcpEval.

Response after that direct peer to peer connection is established and media stream is shared using the MediaStream library. The connection is closed and all RTCPeerConnection objects collected and destroyed during the connection closing phase.

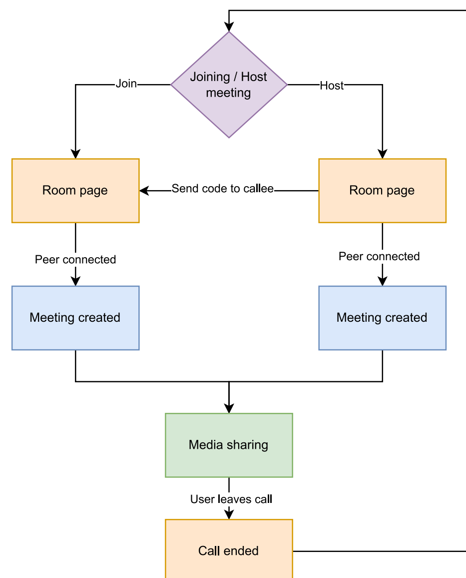

Fig. 3. Flowchart for WebRTC Room Creation and Joining

Algorithm 1. Procedure WebRtcpEval (Connection Http, media_stream M, Users U, Encoding e1, Bitrate Br1)

Initialization	
1.	User_Media = Media_Stream_Generation(media)
2.	peer_connection1 = RTCPeerConnection()
3.	peer_connection2 = RTCPeerConnection()
4.	user1_sdp = None

5.	user2_sdp = None
6.	USR = 2
7.	current_media_packet = None
Step 1. Setting up a connection between two peers	
8.	peer_connection1 = RTCPeerConnection()
9.	peer_connection1.create_local_media_stream(audio=True, video=True)
10.	peer_connection2 = RTCPeerConnection()
Step 2. Session Description Exchange	
11.	user1_sdp = Generate()
12.	peer_connection1.set_local_description(user1_sdp)
13.	peer_connection1.send(user1_sdp, user2)
14.	user1_sdp = peer_connection2.receive()
15.	user2_sdp = peer_connection2.create_answer_sdp()
16.	peer_connection2.set_local_description(user2_sdp)
17.	peer_connection2.send_response(True)
18.	peer_connection2.send(user2_sdp, user1)
19.	user2_sdp = peer_connection1.receive()
20.	peer_connection1.set_remote_description(user2_sdp)
Step 3. Exchange ICE Candidates	
21.	while not done:
22.	candidate1 = peer_connection1.waiting_ice()
23.	candidate2 = peer_connection2.waiting_ice()
24.	if candidate1 is not None:
25.	peer_connection1.send_to_remote(candidate1)
26.	if candidate2 is not None:
27.	peer_connection2.send_to_remote(candidate2)
Step 4. Establish Direct Media Communication	
28.	while current_media_packet is not None:
29.	User_Media.append(current_media_packet)
Step 5. Close the Connection	
30.	peer_connection1.close()
31.	peer_connection2.close()
Step 6. Cleanup	
32.	close(RTCPeerConnection)

In this study, various key parameters have been formulated to quantitatively assess the performance of the communication system. These parameters are defined as follows, conforming to established scientific conventions:

Latency (L): Latency is defined as the sum of various components, including Propagation Time (T_p), Transmission Time (T_t), Queuing Delay, and Processing Delay. It can be mathematically expressed in Eq. (1):

$$L = T_p + T_t + \text{Queuing Delay} + \text{Processing Delay} \quad (1)$$

Propagation Time (T_p): Propagation Time represents the time it takes for a signal to travel from the sender to the receiver over a certain distance (d) at a given transmission speed (v). It can be calculated using the Eq. (2):

$$T_p = \frac{d}{v} \quad (2)$$

Transmission Time (T_t): Transmission Time is determined by the packet size (l) and the data transmission rate (b). It can be calculated as Eq. (3):

$$T_t = \frac{l}{b} \quad (3)$$

Throughput: Throughput measures the rate at which data packets are successfully delivered. It is influenced by Round-Trip Time (RTT) and packet loss (P). Throughput can be defined as Eq. (4):

$$\text{Throughput} = \frac{MPS}{RTT} \times (1 + \sqrt{P}) \qquad (4)$$

3 Experimental Setup

The experimental environment was meticulously configured to ensure reliable and robust testing. The setup featured a system equipped with an Intel Core i7 8th generation CPU boasting a clock speed of 3.20GHz, accompanied by a substantial 16 GB DDR3 RAM operating at 3200MHz clock speed. To enhance graphical processing capabilities, an NVIDIA P620 GPU (Graphical Processing Unit) was integrated into the system.
The experimental phase commenced with the development of the WebRTC protocol and an MEAN (MongoDB, Express.js, AngularJS, Node.js) architecture-based application. This development process was facilitated using the Visual Studio Code IDE (Integrated Development Environment), incorporating a NodeJS compiler plugin for streamlined coding and debugging.

Subsequently, the project was deployed onto a NodeJS server, providing a suitable platform for the evaluation of performance. To monitor and measure performance accurately, WebRTC-internals, an API integrated by most modern browsers, was employed. This API allows for the meticulous tracking of inbound and outbound streams of packets, providing invaluable insights for performance assessment.
The careful setup and integration of these components laid the foundation for a rigorous and comprehensive evaluation of the WebRTC protocol and its associated application, ensuring the reliability of experimental results.

4 Experiment Results and Analysis

In this section, we delve into the outcomes of our conducted experiments and provide a comprehensive analysis of the observed results.

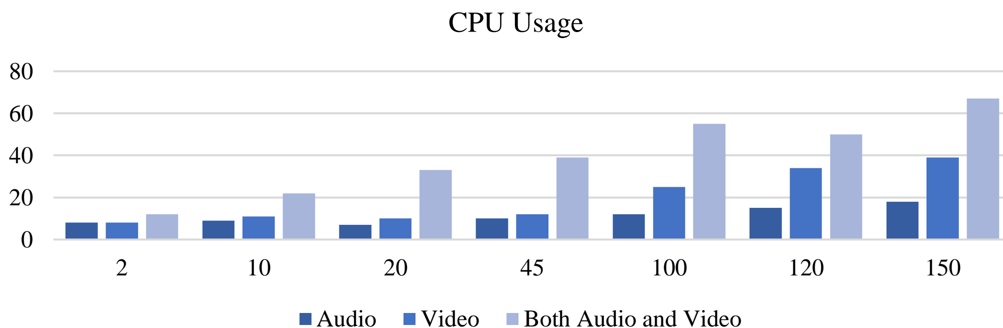

Fig. 4. CPU utilization in WebRTC

Our experiments were conducted with varying numbers of users, as outlined in the accompanying table. The results revealed a noteworthy observation regarding CPU utilization. Specifically, in the context of audio streaming, we observed an optimal level of CPU utilization. However, as the number of users increased in video streaming scenarios, there was a noticeable surge in CPU resource utilization as shown in Fig. 4. This phenomenon not only escalated streaming costs but also led to a degradation in system performance, primarily due to the influx of redundant packets on the client side.

Table 1. Statical Analysis of performance of media data

Media type	No. of users	Jitter (ms)	Latency (ms)	Packet loss (%)	Packet delay (ms)
Audio	2	10	10	0.0	1.0
	10	14	11	0.0	4.0
	20	13	10	0.0	3.2
	45	19	12	0.1	3.7
	100	24	15	0.2	4.8
Video	2	11	30	0.0	1.2
	10	13	30	0.0	3.5
	20	16	13	0.1	2.9
	45	16	32	0.4	4.9
	100	24	53	0.9	7.3

Average	2	10.5	20	0	1.10
	10	13.5	20	0	3.75
	20	14.5	11.5	0.05	3.05
	45	17.5	22	0.25	4.00
	100	24	34	0.55	6.05

Furthermore, the processing delay of packets exhibited a direct correlation with the performance of the processing unit. Over-utilization of system resources during high user loads contributed to delays in packet processing. However, upon surpassing this threshold, the results did not align with expectations due to the prevalence of redundant frames.

Packet drop rates remained minimal, but latency and jitter exhibited more pronounced changes in video streaming scenarios involving a larger number of users. These observations are detailed in Table 1. It's worth noting that WebRTC's utilization of VP9, an optimized open-source encoding scheme, played a pivotal role in reducing total encoding time, contributing to the overall efficiency of the system.

Candidate (pair) id	State / Candidate type	Network type / address	Port	Protocol / candidate type	(Pair) Priority	Bytes sent / received	STUN requests sent / responses received	STUN requests received / responses sent	RTT	Last data sent / received	Last update
CPZrEPvI5K_nBWQkFEx	**succeeded**	**wifi**		**udp**	**0x7e0028fffcfc5000**	**259137599 / 258572195**	**435 / 435**	**437 / 437**	**0.001s**	**2:48:43 PM / 2:48:43 PM**	**2:48:44 PM**
IZrEPvI5K	local-candidate	2401:4900:1c39:3d9d:ad3f:205d:143:115e	61133	host	0x7e7e28ff 126 | 32296 | 255						
InBWQkFEx	remote-candidate		54078	host	0x7e0028ff						
CPFSJH0TyY_gRZYCZEy	in-progress	wifi		udp	0x64003cfffcfe5000	0 / 0	1 / 0	0 / 0			2:29:41 PM
IFSJH0TyY	local-candidate	2401:4900:1c39:3d9d:498d:4a85:5ec6:836a	61132	host	0x7e7f28ff 126 | 32552 | 255						
IgRZYCZEy	remote-candidate	::1	54078	srflx	0x64003cff						
CPFSJH0TyY_nBWQkFEx	in-progress	wifi		udp	0x7e0028fffcfe5000	0 / 0	12 / 0	0 / 0			2:29:47 PM
IFSJH0TyY	local-candidate	2401:4900:1c39:3d9d:498d:4a85:5ec6:836a	61132	host	0x7e7f28ff 126 | 32552 | 255						
InBWQkFEx	remote-candidate		54078	host	0x7e0028ff						
CPFSJH0TyY_rGIRjkru	waiting	wifi		udp	0x64001efffcfe5000	0 / 0	0 / 0	0 / 0			2:29:41 PM
IFSJH0TyY	local-candidate	2401:4900:1c39:3d9d:498d:4a85:5ec6:836a	61132	host	0x7e7f28ff 126 | 32552 | 255						
IrGIRjkru	remote-candidate	::ffff:127.0.0.1	54077	srflx	0x64001eff						
CPFSJH0TyY_vbt50k18	succeeded	wifi		udp	0x6e0028fffcfe5000	0 / 0	53 / 53	1 / 1	0.001s		2:48:44 PM
IFSJH0TyY	local-candidate	2401:4900:1c39:3d9d:498d:4a85:5ec6:836a	61132	host	0x7e7f28ff 126 | 32552 | 255						
Ivbt50k18	remote-candidate		54078	prflx	0x6e0028ff						
CPQLi1oAck_77waFv2O	succeeded	wifi		udp	0x7e001efffcfa3c00	3297 / 3451	53 / 53	3 / 3	0.001s	2:29:31 PM / 2:29:31 PM	2:48:44 PM
IQLi1oAck	local-candidate	192.168.1.14	61131	host	0x7e7d1eff 126 | 32030 | 255						
I77waFv2O	remote-candidate		54077	host	0x7e001eff						
CPZrEPvI5K_gRZYCZEy	waiting	wifi		udp	0x64003cfffcfc5000	0 / 0	0 / 0	0 / 0			2:29:41 PM
IZrEPvI5K	local-candidate	2401:4900:1c39:3d9d:ad3f:205d:143:115e	61133	host	0x7e7e28ff 126 | 32296 | 255						
IgRZYCZEy	remote-candidate	::1	54078	srflx	0x64003cff						
CPZrEPvI5K_rGIRjkru	waiting	wifi		udp	0x64001efffcfc5000	0 / 0	0 / 0	0 / 0			2:29:41 PM
IZrEPvI5K	local-candidate	2401:4900:1c39:3d9d:ad3f:205d:143:115e	61133	host	0x7e7e28ff 126 | 32296 | 255						
IrGIRjkru	remote-candidate	::ffff:127.0.0.1	54077	srflx	0x64001eff						

Fig. 5. Performance of the STUN and TURN Servers

The performance of STUN (Session Traversal Utilities for NAT) and TURN (Traversal Using Relays around NAT) servers also emerged as critical factors in exchanging Session Description Protocol (SDP) information for NAT traversal during connection establishment. The performance of these servers is visualized in Fig. 5.

Fig. 6 provides insights into the volume of media data transmitted and received among users over time.

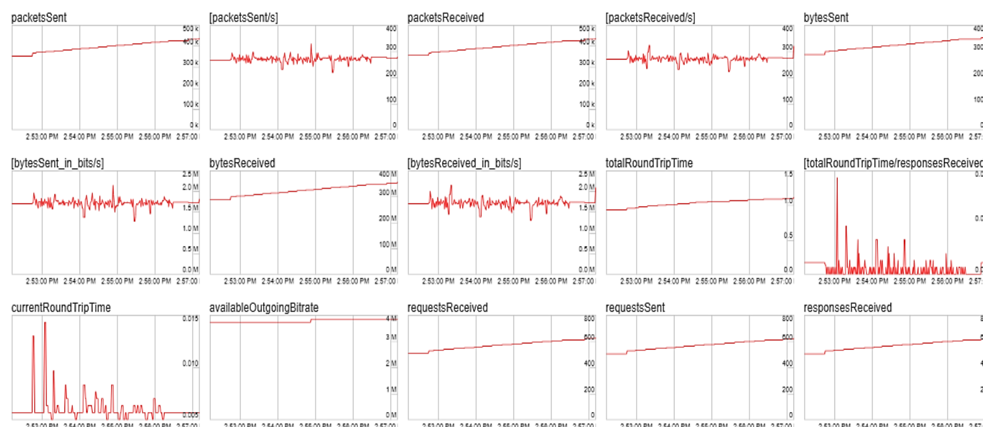

Fig.6. Statistics depicting data send and receive between peers

These results and analyses shed light on the intricacies of WebRTC performance, especially in scenarios involving varying user loads and different types of media streaming.

Conclusion

The experimental findings presented in this study draw valuable conclusions regarding the performance of the WebRTC protocol. It is evident that the WebRTC protocol functions effectively within its designed parameters,

delivering optimal results until the number of users within a room remains below a certain threshold. However, once the user count surpasses this threshold, issues related to packet redundancy, inherent to the peer-to-peer communication model, begin to surface. These issues manifest as increased processing delays and other performance limitations, rendering such scenarios impractical.

To address the limitations observed in the current research, a future direction involves the proposal of an optimized framework. This proposed framework will incorporate a hybrid model, seamlessly integrating elements of both peer-to-peer and client-server architectures. The primary objective is to mitigate resource utilization, reduce latency and jitter, and ultimately alleviate processing time and packet delays.

The experiments conducted in this paper were conducted with up to 100 users for analysis. Future research endeavors will expand upon this by considering a larger user base, thereby enhancing the comprehensiveness of the analysis. Additionally, the impact of adaptability on protocol performance will be a focal point of future investigations, aiming to provide a more detailed and nuanced understanding of WebRTC's capabilities.

References

1. Kara IJ, Suprianto A (2023) APLIKASI FILE SHARING PEER TO PEER BERBASIS WEB MENGGUNAKAN WEBRTC. Incomtech 12:16–23
2. Wu D, Hou YT, Zhu W, et al (2001) Streaming video over the Internet: approaches and directions. IEEE Trans Circuits Syst Video Technol 11:282–300. https://doi.org/10.1109/76.911156
3. Blum N, Lachapelle S, Alvestrand H (2021) WebRTC: real-time communication for the open web platform. Commun ACM 64:50–54. https://doi.org/10.1145/3453182
4. Madnani D, Fernandes S, Madnani N (2020) Analysing the impact of COVID-19 on over-the-top media platforms in India. Int J Pervasive Comput Commun
5. Mousavi H, Nasr MD (2020) Evaluating the Relationship between Overconfidence of Senior Managers and Abnormal Cash Fluctuations with respect to Financial Flexibility in Companies Listed in Tehran Stock Exchange. Tech Soc Sci J 11:210–225. https://doi.org/10.47577/tssj.v11i1.1570
6. Emmanuel EA, Dirting BD (2017) A Peer-To-Peer Architecture For Real-Time Communication Using Webrtc. 3:
7. Tarim EA, TekiN HC (2019) Performance evaluation of WebRTC-based online consultation platform. Turk J Electr Eng Comput Sci 27:4314–4327. https://doi.org/10.3906/elk-1903-44
8. Improving the Efficiency of WebRTC Layered Simulcast Using Software Defined Networking | SpringerLink. https://link.springer.com/chapter/10.1007/978-3-031-40467-2_2. Accessed 2 Oct 2023
9. Abu-AlShaeer A (2023) The Using JavaScript on 5G networks to improve real-time communication through WebRTC. Al-Rafidain J Eng Sci 09–23. https://doi.org/10.61268/xkftbq59
10. Diallo B, Ouamri A, Keche M (2023) A Hybrid Approach for WebRTC Video Streaming on Resource-Constrained Devices. Electronics 12:. https://doi.org/10.3390/electronics12183775
11. Namee K, Sawatdee P, Promsorn N, et al (2023) Applying of Websocket and WebRTC for Video Calling in Telemedicine during COVID-19 Pandemic. In: 2023 International Conference on Information Networking (ICOIN). pp 340–346
12. python-socketio — python-socketio documentation. https://python-socketio.readthedocs.io/en/latest/. Accessed 2 Oct 2023
13. Sharma T, Mangla T, Gupta A, et al (2023) Estimating WebRTC Video QoE Metrics Without Using Application Headers
14. Johansson D, Holmgren M (2014) Towards Implementing Web-Based Adaptive Application Mobility Using Web Real-Time Communications. IEEE Computer Society, pp 483–486
15. Sredojev B, Samardzija D, Posarac D (2015) WebRTC technology overview and signaling solution design and implementation. In: 2015 38th International Convention on Information and Communication Technology, Electronics and Microelectronics (MIPRO). pp 1006–1009
16. Heikkinen A, Koskela T, Ylianttila M (2015) Performance evaluation of distributed data delivery on mobile devices using WebRTC. In: 2015 International Wireless Communications and Mobile Computing Conference (IWCMC). pp 1036–1042
17. Mauro MD, Longo M, Carullo G, Tambasco M (2016) A Performance Evaluation of WebRTC over LTE
18. Fai Ng K, Yan Ching M, Liu Y, et al (2014) A P2P-MCU Approach to Multi-Party Video Conference with WebRTC. Int J Future Comput Commun 3:319–324. https://doi.org/10.7763/IJFCC.2014.V3.319
19. Edan NM, Al-Sherbaz A, Turner SJ (2017) WebNSM: a novel WebRTC signalling mechanism for one-to-many bi-directional video conferencing. IEEE, London
20. Suciu G, Stefanescu S, Beceanu C, Ceaparu M (2020) WebRTC role in real-time communication and video conferencing. In: 2020 Global Internet of Things Summit (GIoTS). pp 1–6