WEBRTC AND PJSIP IN ASTERISK: A CLOUD-BASED APPROACH

Denisa-Oana SELIN[1], Alin-Tudor SFERLE[1], Tudor-Mihai BLAGA[1], Daniel ZINCA[1], Virgil DOBROTA[1]
[1]*Communications Department, Technical University of Cluj-Napoca, Romania*
Corresponding author: Virgil Dobrota (e-mail: Virgil.Dobrota@com.utcluj.ro)

Abstract: This paper presents our design of launching WebRTC and PJSIP in an IP-based Private Branch Exchange instance running in OpenStack private cloud. This implementation was done using Ubuntu 20.04 LTS-based Asterisk in a Triangle topology (browser-server-browser). The major contributions were the following: (1) design how to integrate open-source software devices working with WebRTC in a private cloud; (2) security and encryption ensured by generating certificates and setting them appropriately in the terminals; (3) metrics monitoring along with call logs available directly in the browser by integrating Asterisk with Grafana Cloud. We had three scenarios: (1) PJSIP-to-PJSIP; (2) WebRTC-to-WebRTC; (3) PJSIP-to-WebRTC.

Keywords: *PJSIP; private cloud; WebRTC.*

I. INTRODUCTION

Call control protocols such as Session Initiation Protocol (SIP) and Web Real-Time Communication (WebRTC) have become an important benchmark in communications over Internet due to the development of multimedia technologies. The protocol stack in Figure 1 briefly shows the interworking between SIP/ WebRTC and all the other protocols at Application, Transport and Network Layers [1]. Note that Stream Control Transmission Protocol (SCTP) over Datagram Transport Layer Security (DTLS) is needed for data channel, whilst Secure Real-time Transport Protocol (SRTP) for non-media data.

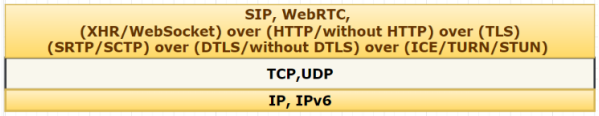


Figure 1. SIP, WebRTC and other protocols.

SIP protocol is an Application Layer protocol responsible for initiating, modifying, and terminating the session and underlies various multimedia applications and VoIP systems. Being a text-based implementation as the more popular HTTP protocol, SIP will benefit from multiple types of request messages, divided into several methods (`REGISTER`, `INVITE`, `BYE`, `ACK`, `CANCEL`, etc.) and multiple types of responses split into classes that show the state following the request (e.g., `202 Accepted`, `300 Multiple Choices`, `403 Forbidden` etc.) [1], [2]. The main purpose of WebRTC according to [3] is to develop multimedia applications directly from the browser, without installing additional software and providing peer-to-peer connectivity between users. The authors of [4] present a videoconferencing system using Scaledrone, a push messaging service that makes it simple to add real-time functionality to your website or mobile app. When it can, Scaledrone uses WebSockets, but when it is not possible, it resorts to older methods such as XMLHttpRequest (XHR) streaming, JSON with Padding (JSONP) queries, etc. According to [5], a reliable, secure, and feature-rich videoconferencing solution that meets the needs of today's organizations and people was developed using WebRTC technology, React JS and Video SDK. The authors have designed a modern and easy-to-use interface thanks to an efficient front-end framework.

The main components of the Asterisk are presented in Figure 2: PJSIP (PJ coming from the surname of its creator B. Prijono), audio and video codecs, CDR drivers, file format and system configuration drivers. The PJSIP library aims to incorporate various protocols: SIP, Session Description Protocol (SDP), RTP, Session Traversal Utilities for NAT (STUN), Interactive Connectivity Establishment (ICE).

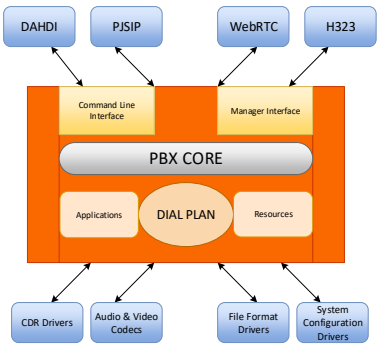


Figure 2. Asterisk architecture.

Call Detail Records has as its main purpose to store call history at table or database level and after analyzing the

Manuscript received March 31, 2024; revised May 28, 2024

stored tables statistics can be made on network usage, user charging, call duration. System compatibility and reliable communication are established by using various audio (GSM, iLBC, LPC10, Speex) and video (VP8, H.264) codecs that will encode incoming signals into bits, which are then decoded at the destination [3], [6], [7], [8], [9].

We noticed that there are few new comming commercial solutions, running both WebRTC and SIP, to be used in modernising the telecommunications solutions in general, and in education in particular. For instance, in [20] Alcatel-Lucent Enterprise shows Unified Communications as a Service (UCaaS) and Communications Platform as a Service (CPaaS). As they are rather very expensive, we wanted to investigate how to design and to integrate open-source software phones in a private cloud, e.g., in OpenStack. The second objective was to solve the security and encryption issues by generating certificates and setting them appropriately in the terminals. As we envisaged the solution to be used for charging the secure telecom services offered, we had to design the metrics monitoring along with call logs available directly in the browser by integrating Asterisk with Grafana Cloud. Overall, the pretext was to have an implementation of a videoconferencing solution using a cloud instance of Asterisk. By the time this paper was submitted, the Cyber Mega Phone browser application has proved to be a suitable environment for evaluating the performance.

The rest of the paper is organized as follows: Section II presents the related work; Section III describes the implementation solution, followed by the experimental results. The last section presents the conclusions and future work.

II. RELATED WORK

Due to the development and increased interest in multimedia services and real-time communication through IP networks or various web applications, new technologies (WebRTC) have been implemented alongside the old ones (SIP) for users to have a pleasant experience and easy to use services. In the paper [10], an integrated communication prototype was built and tested consisting of various software components such as: Kamailio SIP proxy server which has embedded WebSocket protocol, webrtc-to-sip gateway for signaling and media services. Using the Apache web server and the WebRTC client that was developed as an application using JavaScript libraries (JsSIP, sipML5) to provide the required SIP and WebSocket encoding features, various test scenarios were carried out. Thus, following the capture and analysis of the topology in order to interconnect two or more users, it was found that the sessions initiated between peers in terms of web browsers are functioning, between non-peers being more difficult to manage due to different key exchange mechanisms (SRTP-SDES/ DTLS-SRTP). In addition, the presence of a gateway was found necessary for a connection between a WebRTC client and a SIP client (registered in the Jitsi application).

Currently, IP-based services are used by many users, pupils/students, teachers and not only, and along with good real-time communication, the bases of future technologies and implementations are not missing protocols and standards such as SIP and WebRTC. Thus, according to [11] a merger between SIP and WebRTC is desired in order to implement a gateway between the two. It works by turning the IP-PBX into a WebRTC capable device. When instantiating a call from a user WebRTC gateways have the role of determining whether the user can be reached in this way, otherwise a different format such as SIP is required. For security, WebRTC uses SRTP in online data exchange, and negotiations use STUN and ICE to determine when a data transfer can be initiated. The configured gateway will ensure media transmission quality, flexibility, optimization, graphical interface, unified communications and guaranteed functionality. The analyzed results show that there are no losses/errors in the network, and in case of packet loss this is not observable. Latency is low and distance between users is not an impediment.

The article [12] analyzes the network performance in audio/ video communications, both SIP and WebRTC, observing the advantages and disadvantages of each of them. For both protocols prototypes were created as follows: in WebRTC development a virtual machine with Ubuntu 18.04 LTS operating system as server, node.js web server and Google Chrome browser were used as software resources, and in SIP development a machine was used virtual with the FreePBX system as the SIP server, the Bria softphone as the communication application and the Google Chrome web browser. Based on the previous configurations, the star test topology was created with a peer-to-peer connection using 2 clients, 4 clients, 6 clients and performance parameters such as throughput, jitter, packet loss were analyzed using Wireshark application. Analyzing the captures and performance parameters, it is observed in the case of WebRTC that the yield, jitter and packet loss are better than in the case of SIP, this being also influenced by other factors such as the codec, signaling, the platform used.

Quality in VoIP-based services analyzed in [13] can be of two types: objective and subjective. Considering the objective way in measuring the quality of VoIP services and given network performance, it is closely related to the quality of service (QoS), and the subjective way in measuring user satisfaction after using the services is closely related to the quality of experience (QoE). In measuring the quality of services, the main objective parameters are: lost packets, latency, jitter. In measuring the quality of the experience, the main subjective parameter analyzed is: Mean Opinion Score (MOS). The study analyzed numerous scientific references in order to identify the quality parameters and make a report on the observed results.

The authors of [4] present a video conferencing system using Scaledrone, a push messaging service that makes it simple to add real-time functionality to your website or mobile app. When it can, Scaledrone uses WebSockets, but when it's not possible, it falls back on older methods like

streaming XMLHttpRequest (XHR), JSON with padding (JSONP). In addition, the video conference uses a thermal camera that, unlike regular cameras, can detect people without the need for them to move, based on the detected energy (which it will later process and produce the image). This camera and related sensors will be attached to the Raspberry Pi. The browser used was Mozilla Firefox through which the application was tested. Thus, when initiating a video conference between two users, they will be able to communicate, the instance being successfully created, and to test the thermal camera, it was necessary to activate an I2C and VNC server along with a Python script. After running the code, the temperature values and the thermal image can be observed, being considered an important point for reducing human interaction during the pandemic. In addition, it is found that for a low bandwidth the latency is low, and the costs of implementing such a solution along with data security present important points of interest and advantages among users.

According to [5], a reliable, secure communication system and a safe, secure and feature-rich video conferencing solution to meet the needs of today's organizations and people has been developed using WebRTC, React JS and Video SDK technology. The authors have designed a modern and easy-to-use interface thanks to an efficient front-end framework. The application contains a login page where users only need to enter their name to join a meeting, screen recording functionality to be edited or adapted in various ways in later uses, chat for text message communication between participants and integration with various platforms to streamline media services.

The paper [14] aims to test the main video conferencing applications that use WebRTC, using KITE as the test engine, which runs numerous tests daily to report them to a web page. In this testbed consisting of virtual machines for the five client applications, tested separately in the same AWS at a private cloud, several scenarios were followed (constant increase in the number of participants, increase in transmission speed). In addition, functions have been implemented to determine bit rates, check video for each application and evaluate the quality of the transmission. These tests were done using the Java-based KITE utility, along with Selenium WebDriver for browser access and testing. After analyzing and performing the tests, it was found that in the case of Kurento if the number of participants increases above a certain value (about 40) the RTT increases suddenly, and for a larger number of users the bit rate is low, improving together with the transmission quality for shortly after about 100 participants are in the video conference. In the case of Jitsi, the stoppage of the video transmission is found if a number of more than 200 participants join, and for the other SFUs, a similar behavior is found following the tests. Thus, after testing, a comparative analysis can be made between the SFUs used in order to determine which best matches the users' requirements, following which it is possible to improve these services in the future.

How to implement WebRTC and video conferencing signaling mechanisms and improve quality in application design is the main focus of that article [15]. The testbed aims to analyze the video conferencing system using networks such as LAN and WAN and the web browsers Google Chrome and Mozilla Firefox. Various functionalities have been introduced in the implemented application, among which we mention the possibility of using the full screen, pausing audio and video during the conversation, establishing connections between counterparts using various approaches and procedures in establishing a communication. WebSocket is used as signaling protocol and four types of signaling messages are developed such as: initiator, receiver, peerChannel, SDP exchange. As for the quality of the experience, it was analyzed through different users who evaluated aspects such as: the quality of audio, video, the ease of use of the application, the quality of the connection, the desire to reuse these technologies, the results being located at a high score and a score of good to excellent.

Some recent commercial proposals showing the potential benefits of using WebRTC were published in [21], [22].

III. IMPLEMENTATION

This chapter presents SIP and WebRTC capabilities in implementing various scenarios (video conferencing application, peer-to-peer calling, SIP-WebRTC calling) using an Asterisk instance in a private cloud. The testbed includes Asterisk PBX version 20.0.1 under Ubuntu 20.0.4 LTS distribution as OpenStack cloud instance having IP address: 10.8.8.173. The architecture of the adopted solution is shown in Figure 3. A triangular topology contains a core server and several browsers. Thus, the general configurations were performed at the level of the `pjsip.conf` file, respectively the dialplan at the level of the `extensions.conf` file, and configurations such as the generation of certificates, the creation of an HTTPS server and the setting of a transport mode via WebSocket being necessary for the WebRTC standard to be functional.

The applications used to interconnect between users were both browser applications and software applications with softphone functionality (browser-based applications: Cyber Mega Phone, MizuTech-VoIP; installed softphones: Blink, Zoiper5).

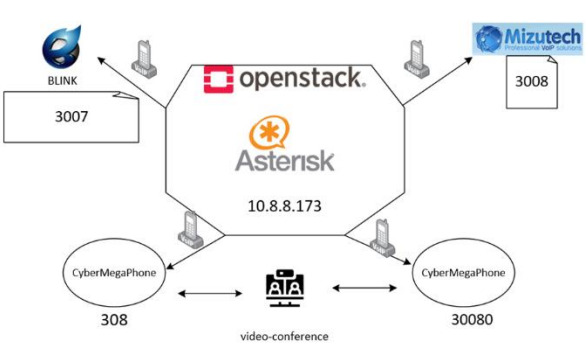


Figure 3. Proposed architecture

The first step in setting up the work environment was to access the private cloud within UC Labs through an OpenVPN client, with the import of the appropriate file client-`licenta.ovpn`, along with the credentials (user and password). It is out of the scope of this paper to present in detail the initial configuration but see it in [7].

A. Configuring Asterisk for WebRTC

A major difficulty of the whole work was to properly configure WebRTC clients. Three major steps were required: (1) creating an Asterisk HTTPS TLS server (including generation of certificates with the appropriate keys); (2) creating a PJSIP WebSocket for transport; and (3) creating a PJSIP endpoint, creating the WebRTC client with the corresponding registration address, and the specific authentication. As a first step we generated self-signed certificates to prevent attacks on the server (see Figure 4), to avoid compromising the security of the system, and, finally, to ensure encryption of streams [3].

```
$ sudo mkdir /etc/asterisk/keys
$ sudo contrib/scripts/ast_tls_cert -C
10.8.8.173 -O "My Organization" -b 2048 -d
/etc/asterisk/keys
```

Figure 4. Generating the certificates

Figure 5 shows that a new keys directory is created in which the certificates are stored and by means of the `ast_tls_cert` script located in the `contrib/scripts` directory, the certificate was generated including options, such as IP address, organization name, key length (e.g., 2048 bits) and the file in which they are stored.

```
$ ls -l /etc/asterisk/keys
total 32
-rw------- 1 root root 1399 Apr 1 08:40
asterisk.crt
-rw------- 1 root root  928 Apr 1 08:39
asterisk.csr
-rw------- 1 root root 1675 Apr 1 08:39
asterisk.key
-rw------- 1 root root 3074 Apr 1 08:39
asterisk.pem
-rw------- 1 root root  161 Apr 1 08:39 ca.cfg
-rw------- 1 root root 1781 Apr 1 08:39 ca.crt
-rw------- 1 root root 3311 Apr 1 08:39 ca.key
-rw------- 1 root root  112 Apr 1 08:39 tmp.cfg
```

Figure 5. Generating the certificates

Figure 6 highlights the configuration with the HTTP server being enabled, specifying the IP address and the port to receive connection requests (0.0.0.0: for all available network interfaces; 8088: for HTTP connections). Also, we enabled the TLS encryption, the paths to the generated certificate and to the private key in the keys directory [16].

```
[general]
enabled = yes
bindaddr = 0.0.0.0
bindport = 8088
tlsenable = yes
tlsbindaddr = 0.0.0.0:8089
tlscertfile = /etc/asterisk/keys/asterisk.pem
tlsprivatekey = /etc/asterisk/keys/asterisk.key
```

Figure 6. http.conf configuration file

The current status of the implemented HTTP server is displayed in the Asterisk's console, as in Figure 7.

```
asterisk*CLI> http show status
HTTP Server Status:
Prefix: Server: Asterisk/20.0.1
Server Enabled and Bound to 0.0.0.0:8088
HTTPS Server Enabled and Bound to 0.0.0.0:8089
Enabled URI's:
/httpstatus => Asterisk HTTP General Status
/phoneprov/... => Asterisk HTTP Phone
Provisioning Tool
/metrics/... => Prometheus Metrics URI
/static/... => Asterisk HTTP Static Delivery
/ws => Asterisk HTTP WebSocket
```

Figure 7. Status of the HTTP server

Configuring a Secure WebSocket (WSS) for encrypted communication between WebRTC and Asterisk clients is mandatory in this architecture. Figure 8 shows the second step, i.e., the selection of a transport using the WSS protocol listening on all available network interfaces. This was done in the `pjsip.conf` file located in the `/etc/asterisk` directory [16].

```
[transport-wss]
type = transport
protocol = wss
bind = 0.0.0.0
```

Figure 8. Cyber Mega Phone: WebSocket configuration

The third step was the creation of a registration address, the client authentication and a PJSIP endpoint [16]. Figure 9 shows the configuration mode for client authentication, as well as for the Address of Record (AOR) to represent the contact information, the maximum number of contacts of the customer and the removal of previously created contacts when adding a new one. On the other hand, it is needed to configure the WebRTC client: e.g., the authentication, the way in which it is performed (username, password) and the definition of the user and password for the authentication of the client.

```
[308]
type = aor
max_contacts = 5
remove_existing = yes
[308]
type = auth
auth_type = userpass
username = 308
password = …
```

Figure 9. Registration address and authentication

The configuration mode of the PJISIP endpoint with its related settings are presented in Figure 10. Thus, after specifying the endpoint type, it is associated with the AOR configuration and that of the previously generated authentication, Audio Visual Profile Feedback (AVPF) is used to monitor the quality of streams in real time and adjust various parameters (codec, rate) to ensure the quality of the transmission. Next, encryption of media streams is ensured by means of DTLS along with setting the files

corresponding to the certificates together with specifying the path to them, checking the certificate for the validity of its fingerprint to prevent possible attacks that could take place and setting the option for DTLS to highlight the role of the endpoint- (client or server). To establish a connection with WebRTC clients, it is necessary to enable ICE support to verify connectivity for NAT traversal, and the exchange of information and flows between users is efficient due to the use of optimal transport parameters. In addition, RTCP multiplexing ensures that RTP and RTCP packets are transmitted on the same port to reduce the resources used. Calls received by the configured client are made as specified in the call plan in the defined context, and media streams are handled by the Asterisk server acting as an intermediary in the exchange of streams between two communicating parties. Regarding the codecs, a first step would be to disable all of them, followed by enabling the desired ones only, (VP8, ulaw, and H264). Regarding the audio and video streams, the maximum number of them allowed for an endpoint is set to the value of 15, but it may differ depending on various factors, application, etc. [3].

```
[308]
type = endpoint
aors = 308
auth = 308
use_avpf = yes
media_encryption = dtls
dtls_ca_file = /etc/asterisk/keys/ca.crt
dtls_cert_file = /etc/asterisk/keys/asterisk.pem
dtls_verify = fingerprint
dtls_setup = actpass
ice_support = yes
media_use_received_transport = yes
rtcp_mux =yes
context=local
direct_media = no
disallow = all
allow =! all,ulaw,vp8,h264
max_audio_streams = 15
max_video_streams = 15
```

Figure 10. PJSIP endpoint configuration

Following the configurations made in the `pjsip.conf` file regarding the client authentication, the registration address and the endpoint, a similar procedure will be followed to define the other clients, finally checking the status regarding the endpoints created as in Figure 11.

```
asterisk*CLI> pjsip show endpoints
Endpoint:  <Endpoint/CID> <State> <Channels>
I/OAuth: <AuthId/UserName> Aor: <Aor>
<MaxContact> Contact: <Aor/ContactUri> <Hash>
<Status> <RTT(ms)> Transport: <TransportId>
<Type> <cos> <tos> <BindAddress>Identify:
<Identify/Endpoint> Match: <criteria>
Channel:  <ChannelId> <State> Time>
Exten: <DialedExten> CLCID: <ConnectedLineCID>
==========================================
Endpoint: 30080 In use 1 of inf InAuth:
30080/30080 Aor: 30080 5 Channel: PJSIP/30080-
00000002/ConfBridge Up 00:00:07
exten: video-conference CLCID: ""
==========================================
Endpoint:  client_webrtc Not in use 0 of inf
InAuth:  client_webrtc/client_webrtc
Aor:  client_webrtc Contact:
client_webrtc/sip:12479683@10.8.0.6:65027;
eba4dde794 NonQual
```

Figure 11. Checking the status of endpoints

Figure 12 shows how to configure the dialplan stored in `/etc/asterisk` in terms of video conferencing application or calls between users. For this, the extension `video-conference` was defined, and when a call is realized, it is forwarded to the bridge `conf`. In addition, communication between users can be carried out using the `Dial`, the call being interrupted if there is no response within 20 seconds.

```
exten  => video-conference,1,Answer()
  same => n,ConfBridge(conf)
  same => n,Hangup()
exten => client_webrtc,1,Dial(PJSIP/
client_webrtc, 20);
exten => 3008,1,Dial(PJSIP/3008,20);
exten => 3007,1,Dial(PJSIP/3008,20);
exten => 30080,1,Dial(PJSIP/30080,20);
exten => 308,1,Dial(PJSIP/308,20);
```

Figure 12. `extensions.conf` file

B. Configuring the VoIP Terminals

The following softphones were involved in this testbed: Cyber Mega Phone and MizuTech (directly in the browser), and, respectively, Zoiper5 and Blink. The certificates generated on the Asterisk were downloaded locally using WinSCP. In the case of the browsers, it was necessary to access the privacy and security settings, followed by the certificate management, where the needed file was imported. For installed client applications, certificates were imported from the phone-specific settings menu.

See in Figure 13, the steps to install and to configure the application Cyber Mega Phone. Thus, the current directory was changed to the one responsible for storing static HTTP files. Next, we cloned the application based on the code and related files in [17]. The last command had the role of changing the ownership of the directory and its contents for proper operation by granting the appropriate permissions [3]. The `http.conf` configuration file is in Figure 14.

```
$ cd /var/lib/asterisk/static-http
$ sudo git clone
https://github.com/asterisk/cyber_mega_phone_2k.
git
$ sudo chown -R asterisk:asterisk
cyber_mega_phone_2k
```

Figure 13. Cyber Mega Phone: permission grants

```
[general]
enabled = yes
bindaddr = 0.0.0.0
bindport = 8088
tlsenable = yes
tlsbindaddr = 0.0.0.0:8089
tlscertfile = /etc/asterisk/keys/asterisk.crt
tlsprivatekey = /etc/asterisk/keys/asterisk.key
enablestatic = yes
redirect = /cmp2k
```

```
/static/cyber_mega_phone_2k/index.html
prefix= …
sessionlimit = 100 […]
```

Figure 14. Cyber Mega Phone configuration: HTTP

New functionality was added, regarding redirecting the client to the destination page with the possibility of accessing static files directly from the server. Then the modules were reloaded, and the application was accessed at `https://<IP_address>:8089/cmp2k`. The configuration of the WebRTC client is contained in Figure 15, i.e., the setting of the id, name, password, IP address and extension.

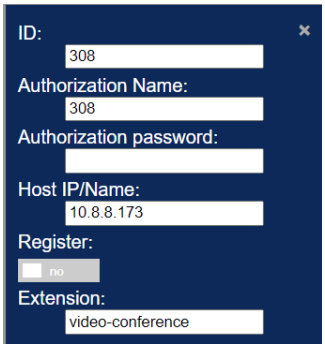


Figure 15. Cyber Mega Phone: WebRTC client using the confbridge-based extension

The configuration of the other WebRTC client, MizuTech, is illustrated in Figure 16. There was a registration part (IP address, username, and password, callee) and additional settings such as: caller id, display name, proxy server (encryption secured using TLS, IP address and port: 5061). Also, the details related to the WebRTC server referred to the secure version of WebSocket along with the IP address and port frequently used in these types of connections [18].

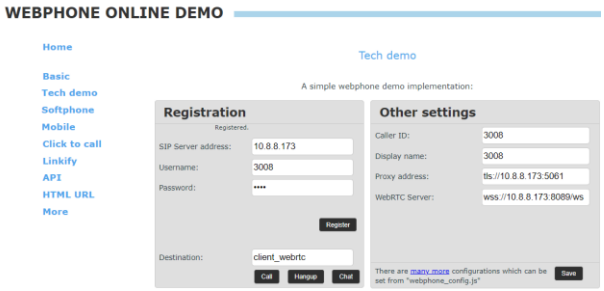


Figure 16. MizuTech: WebRTC client

For the Blink SIP client, we needed, according to Figure 17, details such as server IP address, username along with port and the required transport protocol. A tricky setting was related to the certificate and to import it to secure communication.

As the other SIP client, ZoIPer5, is well-known, herein we skip the description of its configuration, but details about it can be found in [19].

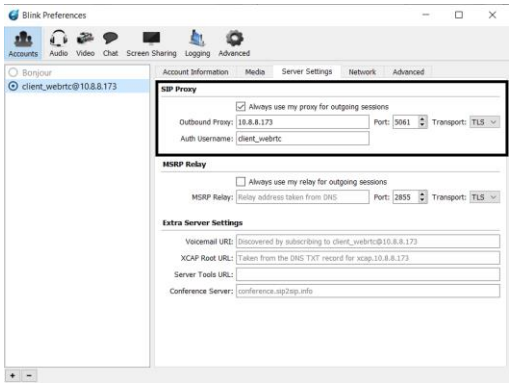


Figure 17. Blink: SIP client

IV. EXPERIMENTAL RESULTS

The scenarios were the following: (1) PJSIP-to-PJSIP with Blink and Zoiper5; (2) WebRTC-to-WebRTC with Cyber MegaPhone directly in the browser; (3) PJSIP-to-WebRTC using an installed software phone and the MizuTech VoIP browser application. Metrics monitoring along with call logs were available directly in the browser by configuring and integrating Asterisk with the Grafana Cloud environment.

A. PJSIP-to-PJSIP with Blink and Zoiper5

To test the connectivity between two SIP users, their registration was done in Blink and Zoiper5 softphones. After the successful registration of the users, the call initiated by one of the clients, the SIP protocol being analyzed by making captures from the moment of the conversation through the Wireshark application. Next, Figure 18 and Figure 19 illustrate the SIP flows generated with the requests and messages corresponding to the actions. Following the requests for the purpose of establishing the communication session, the periodic change of the status and the messages generated was observed. The status regarding the attempt and the request was received by the recipient, the call being signaled and the recipient's device ringing. The call was accepted (status: OK), and the start of audio data transmission was signaled via RTP protocol, indicating the number of transmitted packets and the duration.

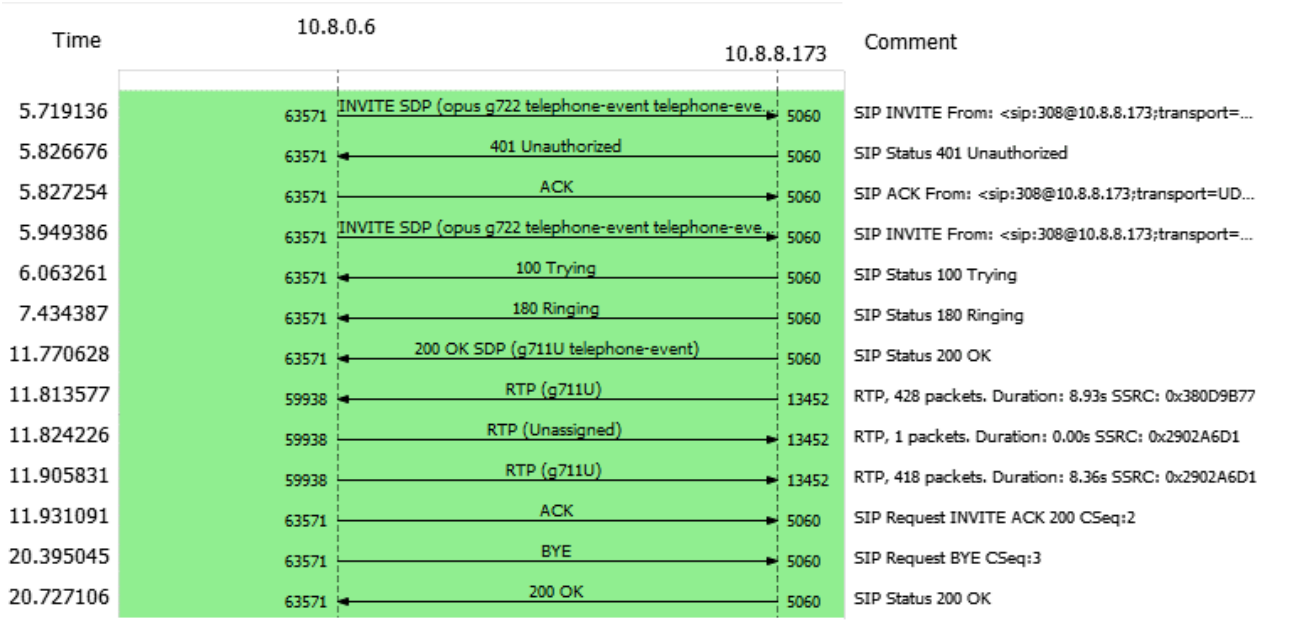
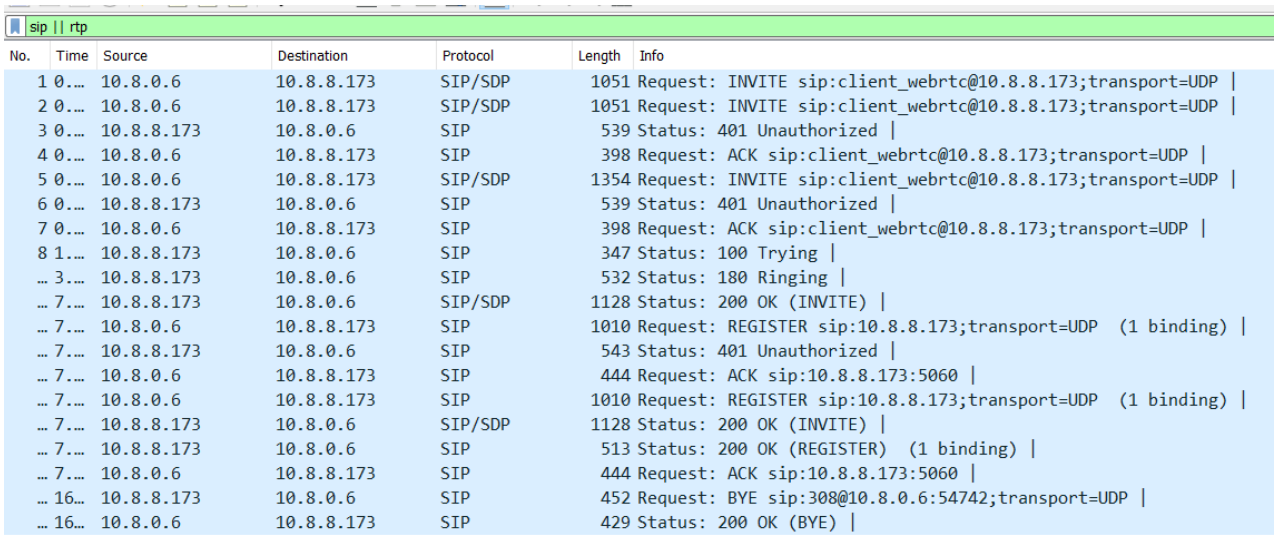


Figure 18. PJSIP-to-PJSIP flow graph.

To end the session, the "BYE" method was involved along with a sequence number and following the confirmation response, the session successfully ended. Note that both RTP packets and SIP signaling were not encrypted.



sip || rtp

No.	Time	Source	Destination	Protocol	Length	Info
1	0.…	10.8.0.6	10.8.8.173	SIP/SDP	1051	Request: INVITE sip:client_webrtc@10.8.8.173;transport=UDP |
2	0.…	10.8.0.6	10.8.8.173	SIP/SDP	1051	Request: INVITE sip:client_webrtc@10.8.8.173;transport=UDP |
3	0.…	10.8.8.173	10.8.0.6	SIP	539	Status: 401 Unauthorized |
4	0.…	10.8.0.6	10.8.8.173	SIP	398	Request: ACK sip:client_webrtc@10.8.8.173;transport=UDP |
5	0.…	10.8.0.6	10.8.8.173	SIP/SDP	1354	Request: INVITE sip:client_webrtc@10.8.8.173;transport=UDP |
6	0.…	10.8.8.173	10.8.0.6	SIP	539	Status: 401 Unauthorized |
7	0.…	10.8.0.6	10.8.8.173	SIP	398	Request: ACK sip:client_webrtc@10.8.8.173;transport=UDP |
8	1.…	10.8.8.173	10.8.0.6	SIP	347	Status: 100 Trying |
…	3.…	10.8.8.173	10.8.0.6	SIP	532	Status: 180 Ringing |
…	7.…	10.8.8.173	10.8.0.6	SIP/SDP	1128	Status: 200 OK (INVITE) |
…	7.…	10.8.0.6	10.8.8.173	SIP	1010	Request: REGISTER sip:10.8.8.173;transport=UDP (1 binding) |
…	7.…	10.8.8.173	10.8.0.6	SIP	543	Status: 401 Unauthorized |
…	7.…	10.8.0.6	10.8.8.173	SIP	444	Request: ACK sip:10.8.8.173:5060 |
…	7.…	10.8.0.6	10.8.8.173	SIP	1010	Request: REGISTER sip:10.8.8.173;transport=UDP (1 binding) |
…	7.…	10.8.8.173	10.8.0.6	SIP/SDP	1128	Status: 200 OK (INVITE) |
…	7.…	10.8.8.173	10.8.0.6	SIP	513	Status: 200 OK (REGISTER) (1 binding) |
…	7.…	10.8.0.6	10.8.8.173	SIP	444	Request: ACK sip:10.8.8.173:5060 |
…	16…	10.8.8.173	10.8.0.6	SIP	452	Request: BYE sip:308@10.8.0.6:54742;transport=UDP |
…	16…	10.8.0.6	10.8.8.173	SIP	429	Status: 200 OK (BYE) |

Figure 19. PJSIP-to-PJSIP signaling.

Although the quality of the experience was good, the major drawback in the first experiment was related to the security of communication. For instance, we were allowed to view the SIP flows, to capture the RTP messages, to store and to playback them.

B. WebRTC-to-WebRTC with Cyber MegaPhone directly in the browser

WebRTC-WebRTC communication was performed within the video conference according to the settings described in the implementation section. During the video conference, the number of users might vary. The Cyber Mega Phone application interface looked like in Figure 20. We experienced a high level of quality for both video and audio, being slightly reduced as new users accessed extensions and joined the conference. In addition, we had the possibility to use the application in full screen, along with classical settings of putting on/off the microphone and/or the camera.

Figure 20. Cyber Mega Phone: application interface

As the communication was encrypted, along with the signaling, functions such as viewing the call stream or playing the content were not possible anymore. Thus, the captures illustrated the TLS handshake process in establishing a secure connection between peers. Initiation of the TLS handshake by the client was done through the "Client Hello" message sent to the server containing information about the TLS version, supported cipher suite, session ID number. In response to the previous message, the server sent the "Server Hello" message along with the chosen options (version, random ID number, cipher suite). Following the negotiations, the exchange of certificates and keys between the server and the client, the messages (see Figure 21) were encrypted according to the negotiated options.

… 3.…	10.8.0.6	10.8.8.173	TCP	54	52470 → 8089 [ACK] Seq=1 Ack=1 Win=262656 Len=0
… 3.…	10.8.0.6	10.8.8.173	TLSv1.3	595	Client Hello
… 3.…	10.8.8.173	10.8.0.6	TCP	60	8089 → 52470 [ACK] Seq=1 Ack=542 Win=64128 Len=0
… 3.…	10.8.8.173	10.8.0.6	TLSv1.3	295	Server Hello, Change Cipher Spec, Application Data, Application Data
… 3.…	10.8.8.173	10.8.0.6	SSH	358	Server: Encrypted packet (len=304)
… 3.…	10.8.8.173	10.8.0.6	TLSv1.2	80	Application Data
… 3.…	10.8.8.173	10.8.0.6	TCP	60	8089 → 52458 [FIN, ACK] Seq=27 Ack=1 Win=501 Len=0
… 3.…	10.8.0.6	10.8.8.173	TCP	54	52458 → 8089 [ACK] Seq=1 Ack=28 Win=1023 Len=0
… 3.…	10.8.8.173	10.8.0.6	SSH	278	Server: Encrypted packet (len=224)
… 3.…	10.8.0.6	10.8.8.173	TCP	54	52016 → 22 [ACK] Seq=161 Ack=1985 Win=1023 Len=0
… 3.…	10.8.0.6	10.8.8.173	TLSv1.3	84	Change Cipher Spec, Application Data
… 3.…	10.8.8.173	10.8.0.6	SSH	134	Server: Encrypted packet (len=80)
… 3.…	10.8.8.173	10.8.0.6	SSH	406	Server: Encrypted packet (len=352)
… 3.…	10.8.0.6	10.8.8.173	TLSv1.2	84	Application Data
… 3.…	10.8.0.6	10.8.8.173	TCP	54	52016 → 22 [ACK] Seq=161 Ack=2417 Win=1022 Len=0
… 3.…	10.8.0.6	10.8.8.173	TCP	54	52458 → 8089 [FIN, ACK] Seq=31 Ack=28 Win=1023 Len=0

Figure 21. WebRTC-to-WebRTC signaling.

Figure 22 presents a TLS segment in detail, with information about the header (frame number and length), IP addresses (source and destination), transport layer protocol (TCP) with specific source and destination ports. As for the TLS protocol, its version was TLS 1.2, the size of the encrypted data in bytes was 25, and the encrypted data was stored as a string according to the negotiated cipher suite options. Thus, the use of peer-to-peer communication with WebRTC ensured the security of services through encryption.

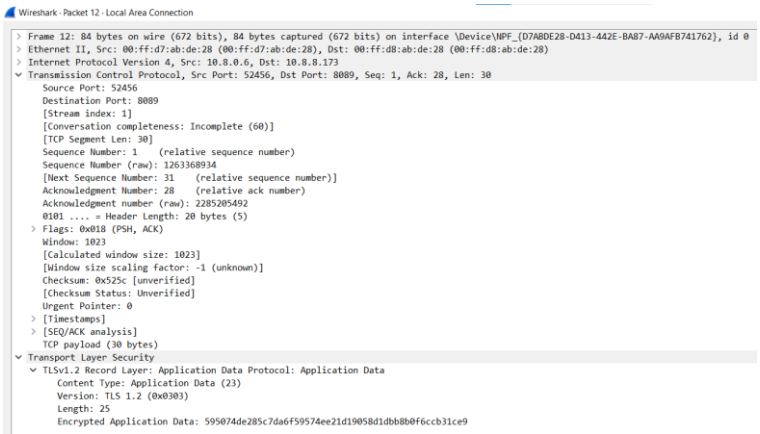


Figure 22. Example of a TLS segment

C. PJSIP-to-WebRTC using an installed software phone and the MizuTech VoIP browser application.

This experiment used a SIP client (Blink or Zoiper5) and an WebRTC client (Mizu Tech VoIP app) available in the browser. The first step was the registration of clients with the related settings and making calls between users. The connection between clients was successfully established and the audio quality was good and no delays were reported when receiving or making a call. But as a downside, the in-app video option in the browser was not available. Also, in this experiment, similar to the previous one, the signaling and data were encrypted,

making impossible to playback the call streams. Figure 23 shows the packets captured during communication, noting the presence of encryption protocols. Attempting to establish connectivity and obtaining NAT information were initiated following STUN requests.

… 15…	10.8.8.173	10.8.0.6	TCP	60 8089 → 52176 [ACK] Seq=1734 Ack=3067 Win=497 Len=0
… 15…	10.8.8.173	10.8.0.6	STUN	182 Binding Request user: afmI:00388e0b7a9fc4f12f3daea3543d9309
… 15…	10.8.8.173	10.8.0.6	TLSv1.2	511 Application Data
… 15…	10.8.8.173	10.8.0.6	STUN	126 Binding Success Response XOR-MAPPED-ADDRESS: 10.8.0.6:54148
… 15…	10.8.8.173	10.8.0.6	STUN	182 Binding Request user: afmI:00388e0b7a9fc4f12f3daea3543d9309
… 15…	10.8.8.173	10.8.0.6	STUN	126 Binding Success Response XOR-MAPPED-ADDRESS: 10.8.0.6:54148
… 15…	10.8.8.173	10.8.0.6	STUN	182 Binding Request user: afmI:00388e0b7a9fc4f12f3daea3543d9309
… 15…	10.8.8.173	10.8.0.6	STUN	126 Binding Success Response XOR-MAPPED-ADDRESS: 10.8.0.6:54148
… 15…	10.8.8.173	10.8.0.6	STUN	182 Binding Request user: afmI:00388e0b7a9fc4f12f3daea3543d9309
… 15…	10.8.8.173	10.8.0.6	STUN	126 Binding Success Response XOR-MAPPED-ADDRESS: 10.8.0.6:54148
… 15…	10.8.8.173	10.8.0.6	STUN	182 Binding Request user: afmI:00388e0b7a9fc4f12f3daea3543d9309
… 15…	10.8.8.173	10.8.0.6	TLSv1.2	1038 Application Data
… 15…	10.8.0.6	10.8.8.173	STUN	106 Binding Success Response XOR-MAPPED-ADDRESS: 10.8.8.173:10614
… 15…	10.8.8.173	10.8.0.6	STUN	126 Binding Success Response XOR-MAPPED-ADDRESS: 10.8.0.6:54148
… 15…	10.8.8.173	10.8.0.6	STUN	182 Binding Request user: afmI:00388e0b7a9fc4f12f3daea3543d9309
… 15…	10.8.8.173	10.8.0.6	STUN	126 Binding Success Response XOR-MAPPED-ADDRESS: 10.8.0.6:54148
… 15…	10.8.8.173	10.8.0.6	STUN	182 Binding Request user: afmI:00388e0b7a9fc4f12f3daea3543d9309
… 15…	10.8.0.6	10.8.8.173	DTLSv1.2	199 Client Hello
… 15…	10.8.0.6	10.8.8.173	STUN	106 Binding Success Response XOR-MAPPED-ADDRESS: 10.8.8.173:10614

Figure 23. PJSIP-to-WebRTC signaling.

Data was transmitted via the TLS and the response from the initiated request included the IP address and port from the address mapping after NAT traversal. A series of requests and responses were followed by the generation of a "Client Hello" message to negotiate parameters for the secure connection. The detailed analysis of a DTLS segment was captured in Figure 24. It includes the component elements within DTLS, e.g., type of data within a record (handshake), version (1), length in bytes etc.

```
> Frame 83: 199 bytes on wire (1592 bits), 199 bytes captured (1592 bits) on interface \Device\NPF_{D7ABDE28-D413-442E-BA87-AA9AFB741762}, id 0
> Ethernet II, Src: 00:ff:d7:ab:de:28 (00:ff:d7:ab:de:28), Dst: 00:ff:d8:ab:de:28 (00:ff:d8:ab:de:28)
> Internet Protocol Version 4, Src: 10.8.0.6, Dst: 10.8.8.173
> User Datagram Protocol, Src Port: 54148, Dst Port: 10614
v Datagram Transport Layer Security
  v DTLSv1.2 Record Layer: Handshake Protocol: Client Hello
      Content Type: Handshake (22)
      Version: DTLS 1.0 (0xfeff)
      Epoch: 0
      Sequence Number: 0
      Length: 144
    v Handshake Protocol: Client Hello
        Handshake Type: Client Hello (1)
        Length: 132
        Message Sequence: 0
        Fragment Offset: 0
        Fragment Length: 132
        Version: DTLS 1.2 (0xfefd)
      > Random: 948ba418ee031939b465e1e215b01034f835dd63312c8556d840eaac85b00d87
        Session ID Length: 0
        Cookie Length: 0
        Cipher Suites Length: 22
      > Cipher Suites (11 suites)
        Compression Methods Length: 1
      > Compression Methods (1 method)
        Extensions Length: 68
      > Extension: extended_master_secret (len=0)
      > Extension: renegotiation_info (len=1)
      > Extension: supported_groups (len=8)
      > Extension: ec_point_formats (len=2)
      > Extension: session_ticket (len=0)
      > Extension: signature_algorithms (len=20)
      > Extension: use_srtp (len=9)
        [JA3 Fullstring: 65277,49195-49199-52393-52392-49161-49171-49162-49172-156-47-53,23-65281-10-11-35-13-14,29-23-24,0]
        [JA3: c14667d7da3e6f7a7ab5519ef78c2452]
```

Figure 24. Example of a DTLS segment

V. CONCLUSIONS AND FUTURE WORK

This paper presented a simple and efficient open-source implementation of VoIP services based on SIP and WebRTC. By the time this work was carried out, we were not aware of similar solutions open-source publicly available. We proved that peer-to-peer communication PJSIP-to-PJSIP did not ensure the security of media communication between users, as the flow was unencrypted. In addition, it was needed to install additional software (e.g., softphones with multiple signaling, authentication options and codecs). As an alternative, the development of web applications through WebRTC proved to be more efficient and secure, as the flow was fully encrypted. Combining SIP (predominant in IP-based telecommunications) with the new coming WebRTC ensured the quality of the transmission and provided customers with a pleasant and secure experience in using the services. Overall, we appreciate the success of the solution by analyzing the signaling, the quality of experience and the security issues.

Once we have access to similar implementations (commercial and open-source) we can continue this work by comparing its VoIP key performance parameters (transfer rate, delay, jitter) with those of the others in similar testbeds. Also, we want to expand the topology to a trapezoidal one (Browser1, Server1, Server2, Browser2), and to involve hardware devices too. We plan to implement a high-performance videoconferencing system using Cisco Collaboration Platforms in the cloud instead of the Asterisk-based one.

ACKNOWLEDGMENT

An initial expanded version was presented by D.O. Selin as B.Sc. thesis in July 2023. A preliminary version was presented at 18th Electronics and Telecommunications Students Symposium (SSET) in May 2023.

REFERENCES

[1] R.R. Roy, “Handbook of SDP for Multimedia Session Negotiations: SIP and WebRTC IP Telephony”, CRC Press, 2020.
[2] R.R. Roy, “Handbook on Session Initiation Protocol: Networked Multimedia Communications for IP Telephony”, CRC Press, 2016.
[3] J.V. Meggelen, R. Bryant, L. Madsen, “Asterisk – The Definitive Guide”, 5th Edition, O’Reilly, 2019
[4] G. Suciu, S. Stefanescu, C. Beceanu and M. Ceaparu, "WebRTC role in real-time communication and video conferencing," 2020 Global Internet of Things Summit (GIoTS), Dublin, Ireland, 2020, pp. 1-6, doi: 10.1109/GIOTS49054.2020.9119656.
[5] S. Ajay, et.al, "Integrable Video Conferencing APP", International Research Journal of Modernization in Engineering Technology and Science, Vol. 05, Issue 04, April-2023.
[6] D.O. Selin, V. Dobrota, “Evaluation of WebRTC and SIP Performance in a Private Cloud”, in 18th Student Symposium on Electronics and Telecommunications, SSET 2023, Technical University of Cluj-Napoca, 2023.
[7] D.O. Selin, “WebRTC and SIP Using Asterisk in the Cloud”, B.Sc. in Telecommunications Technologies and Systems, Technical University of Cluj-Napoca, 2023.
[8] “About PJSIP”, PJSIP, 2023, [Online], Available: https://www.pjsip.org/about.htm.
[9] “Codec”, TKO Video Conferencing, 2023, [Online], Available:http://www.video-conferencing.com/definition/codec.html.
[10] P. Segec, P. Paluch, J. Papan and M. Kubina, "The integration of WebRTC and SIP: Way of enhancing real-time, interactive multimedia communication," 2014 IEEE 12th IEEE International Conference on Emerging eLearning Technologies and Applications (ICETA), Stary Smokovec, Slovakia, 2014, pp. 437-442, doi: 10.1109/ICETA.2014.7107624.
[11] G.P. Shreya, P. Pradhyumna and Mohana, "Internetworking Gateway between WebRTC to SIP to Integrate Real-Time Audio Video Communication," Third International Conference on Inventive Research in Computing Applications (ICIRCA), Coimbatore, India, 2021, pp. 1480-1485, doi: 10.1109/ICIRCA51532.2021.9544559.
[12] S.K. Akmal, A.G. Putrada, F. Dawani, "A Network Performance Comparison of WebRTC and SIP Audio and Video Communications," 2021 Volume 4 Journal of Information Technology and Its Utilisation, 2021, pp. 1-5, doi: https://doi.org/10.30818/jitu.4.1.3939.
[13] B. Adhilaksono, B. Setiawan, "A study of Voice-over-Internet Protocol quality metrics", Sixth Information Systems International Conference (ISICO 2021), IETF, July 2008. Procedia Computer Science Vol.197, 2022, pp.377-384, doi: https://doi.org/10.1016/j.procs.2021.12.153.
[14] E. André, N. Le Breton, A. Lemesle, L. Roux and A. Gouaillard, "Comparative Study of WebRTC Open Source SFUs for Video Conferencing," 2018 Principles, Systems and Applications of IP Telecommunications (IPTComm), Chicago, IL, USA, 2018, pp. 1-8, doi: 10.1109/IPTCOMM.2018.8567642.
[15] A.T. Kalil, S.A. Mahmood, “Peer-to-peer media streaming with HTML5”, International Journal of Electrical and Computer Engineering (IJECE) Vol. 13, No. 2, April 2023, pp. 2356~2362, doi:10.11591/ijece.v13i2.pp2356-2362.
[16] G. Joseph, “WebRTC”, Asterisk, 2018, [Online], Available: https://wiki.asterisk.org/wiki/display/AST/WebRTC.
[17] Cyber Mega Phone GitHub repository, 2024, [Online], Available: https://github.com/asterisk/cyber_mega_phone_2k
[18] “WebPhone Online Demo”, Mizu VoIP Solutions, 2024, [Online], Available: https://www.mizu-voip.com/Software/WebPhone/WebphoneOnlineDemo.aspx.
[19] V. Dobrota, “Unified Communications in Cloud”, Technical University of Cluj-Napoca, 2024, [Online], Available: https://el.el.obs.utcluj.ro/cuc/.
[20] “Network modernization in education”, Alcatel-Lucent Enterprise, 2024, [Online], Available: https://www.al-enterprise.com/en/.
[21] “The First Sales-Oriented Unified Communications Solution”, Wildix, 2024, [Online], Available: https://www.wildix.com/.
[22] “Embrace the Future: WebRTC Revolutionizes Contact Center Communication”, NobelBiz, 2024, [Online], Available: https://nobelbiz.com/omnichannel-contact-center/unified-agent-desktop/integrated-webrtc/.