# Real-Time Low-Latency VoIP Systems

## Project Report

Victor Kamel

Matthew Dardango

December 11, 2023

CSC 461: Multimedia Systems

Fall 2023

Dr. Jianping Pan

University of Victoria

**Table of Contents**

## 1 Background

Voice over Internet Protocol (VoIP) does not refer to a single protocol, but rather a collection of related protocols, standards, and techniques that allow voice to be transmitted over an IP network such as the internet. Although the internet was only created a few decades ago, transmitting voices over long distances has been possible since at least 1876 when the first telephone was demonstrated [1]. The difficulties in producing reliable VoIP systems stem from the fundamental differences between it and traditional telephony.

The global network of interconnected telephones is usually called the Public Switched Telephone Network (PSTN). This is analogous to the internet, which is the global network of interconnected computers. However, the PSTN has some very important differences from the internet. For one, it was designed around the idea of circuit switching, which provides a dedicated line between both speakers. It also was designed to carry only voice, rather than arbitrary data.

At its peak in 2003, the PSTN had an estimated 1.263 billion telephone lines in use [2]. As technology improved in the 1980s, telecommunications companies attempted to add the capability to use the PSTN for other services, such as video and data transfer. This was called Integrated Services Digital Network (ISDN) [3], but because the telephone lines were not designed for high bandwidth, it saw little commercial success. Since VoIP allowed calls to be completed for much cheaper, and had the bandwidth for additional services like videoconferencing, PSTN usage started to decline in favour of the internet. Today, many countries are retiring their PSTN, such as the UK, which is scheduled to complete their retirement by 2025 [4].

### 1.1 VoIP Systems

The first VoIP systems that achieved widespread usage used the H.323 standard. Formally published by the ITU-T as "Packet-Based Multimedia Communications Systems" in 1996, H.323 describes several other binary protocols that can be used together to achieve multimedia communication over a packet-switched network such as the internet. H.323 also describes the use of gateways that allow interoperability with other systems like the PSTN. However, the complexity of H.323, especially compared with more modern VoIP protocols, has led to a decline in its usage [5].

Other popular early VoIP systems used a protocol called the Extensible Messaging and Presence Protocol (XMPP). XMPP was standardized in 2004 and based entirely around XML. It drew heavy inspiration from SMTP, as XMPP is also decentralized and uses addresses that resemble email addresses. Though XMPP was primarily designed to be an instant messaging protocol, it is also very extensible. One of the most important XMPP extensions is Jingle, which provides peer-to-peer signaling for establishing VoIP sessions [6]. Jingle was developed by Google in the early 2000s for Google Talk. Although Google Talk was discontinued in 2013, Jingle has found usage in other projects [7].

There have also been many proprietary VoIP systems. One of the most well-known VoIP applications, Skype, used a proprietary protocol up until 2014 [8]. TeamSpeak still uses a proprietary protocol to the present day [9].

Because of the wide array of available VoIP systems and lack of standardization for many of them, along with the increasing quantity of web applications, a desire grew to have a single interface that anyone could use to enable real-time multimedia communication. The

WebRTC project fulfilled this need, and indeed it is what modern VoIP applications like Discord, Zoom and Microsoft Teams use behind the scenes. Our project will focus on how we have used WebRTC to produce our own simple VoIP application.

## 2 Components of a VoIP System

Before analyzing WebRTC in detail, we conducted a high-level overview of the key components in a VoIP system. These include signalling, media description, delivery, and NAT traversal. Other components, such as quality of service, security, and encryption, are also important, but they are not the focus of our project. Unlike the circuit switched nature of the PSTN, the internet is packet switched, so this demands we design our protocols differently.

### 2.1 Signalling

Signalling refers to the mechanisms needed to initiate, terminate, and manage a call. A signalling protocol is responsible for several tasks, including allowing clients to register their address, discover the existence and location of other clients, dial other clients, and negotiate capabilities between one another, allowing for extensibility without sacrificing compatibility with older protocols and software. Many signalling protocols include extra call control features as well, such as the ability to place a call on hold or transfer the call to another location [10].

In the PSTN, signalling between household telephone equipment and exchanges is usually achieved by a 12- or 16-digit keypad in what is known as dual-tone multi-frequency signaling (DTMF) [11]. Signalling between telephone exchanges uses Signaling System

No. 7 (SS7). For VoIP, several different protocols are available, such as H.225 which is used by H.323 systems. However, the most common signalling protocol used today is the Session Initiation Protocol (SIP).

### 2.1.1 Session Initiation Protocol

SIP, first standardized in 1999 in RFC 2543, is an application-layer text-based protocol used to manage two- and multi-party communication sessions. It can be used not only for VoIP but also for other types of multimedia, such as video conferencing. SIP is designed to be transport layer agnostic, so it can be carried over TCP, UDP, or other transport protocols as needed.

Each endpoint of a SIP system is referred to as a user agent. This might be a physical device, such as an IP phone, or a program installed on a general-purpose computer, referred to as a softphone. Each user agent must register themselves with a registrar server, allowing other user agents to locate the correct IP address for whom they wish to call. Proxy servers handle passing SIP requests around the internet to ensure they reach the intended recipient. Gateways can also be used, allowing SIP to interoperate with other networks such as the PSTN [12].

The design of SIP draws heavy inspiration from both SMTP and HTTP. Each user agent is identified with a URI that uses the same syntax as an email address, though DNS SRV records are used instead of MX records. SIP protocol messages use a request and response model that closely resembles that of HTTP. For example, response code 301 means “moved permanently” for both SIP and HTTP [13]. The familiarity and simplicity of SIP,

especially compared to older signalling protocols like H.225, means that it has now become the dominant signalling protocol used worldwide.

### 2.2 Media Description

There are many different types and standards of multimedia available today. To establish a multimedia communication session between two peers, they must agree on a common set of encodings that both parties can support, along with other attributes that define the session in question, such as IP addresses, start and end times, and encryption keys.

#### 2.2.1 Session Description Protocol

A common solution to the problem is the Session Description Protocol (SDP), developed alongside SIP and published in 1998. It describes the entire session using a group of `<type>=<value>` fields, one per line. Each `<type>` is a single case-sensitive character, while the `<value>` can be longer but must conform to certain rules depending on the `<type>`. Some examples of fields are `v` for the protocol version (always 0), `c` for connection information, `s` for the session name, and `m` for the media type. SDP can be extended by using the `a` (attribute) field [14].

SDP does not specify how it is distributed between clients. In practice, it is usually carried as a payload within SIP messages, but it can also be transmitted over HTTP.

### 2.3 Delivery

Delivery refers to the process of transmitting the multimedia payload over the network from one device to another. In the case of VoIP, this payload consists of individual audio samples. Although VoIP largely traces its origins to the 1990s, the first phone call over a

computer network was made in August 1974 using the Network Voice Protocol (NVP) over ARPANET [15]. This protocol evolved over the next few decades, becoming the real-time transport protocol (RTP) used today.

### 2.3.1 Real-time Transport Protocol

RTP addresses several issues with transporting multimedia over TCP, which provides reliability guarantees that conflict with the real-time desires of many multimedia applications. The RTP header contains a sequence number identifying the packet count and a timestamp containing the sample count, making it simple to determine what samples are missing. Since multimedia applications are more sensitive to latency and jitter than traditional internet applications, RTP implements its own detection and compensation techniques for packet loss and reordering rather than relying on TCP's techniques. Since the loss of a single RTP packet may lead to the loss of only a fraction of a second of audio samples, errors can often be concealed rather than requesting and waiting for the source to resend them. Naturally, since the point of RTP is to solve issues with TCP, it is usually transmitted using UDP [16].

RTP is used with an out-of-band companion protocol called the RTP Control Protocol (RTCP) that monitors the RTP stream and provides control information and quality of service. It requires both the sender and receiver to share periodic reports that can provide insight on the amount of packet loss occurring. RTCP bandwidth is designed to be much smaller than that of the RTP stream. Usually, RTP uses an even port number and RTCP uses the odd port number immediately after, but both protocols can be multiplexed together over a single port [17].

### 2.4 NAT Traversal

An IPv4 address is 32 bits long. This means that the number of unique addresses is $2^{32}$, or about 4.3 billion. In practice, due to large blocks of reserved address space, the number of usable addresses is actually much smaller. It became apparent by the 1990s that there would not be nearly enough addresses to handle the accelerating growth of the internet, so a new standard with 128-bit addresses, called IPv6, was conceived. However, since existing equipment and software only supported IPv4, a "short-term" solution proposed was Network Address Translation (NAT) [18].

NAT works by rewriting the source IP address on all outbound packets that travel through a NAT-enabled router with the router's own IP address. This allows for the assignment of private, non-routable IP addresses to devices hidden behind such routers. To keep track of which internal IP address and port is related to each public IP address and port, routers must maintain a NAT table. In practice, NAT means that only one public IP address is used for each local network, rather than one address for each device.

For a VoIP application to achieve the lowest possible latency, it is desirable to establish a direct peer-to-peer connection without intermediaries. This is problematic, since almost all devices today access the internet from behind a NAT-enabled router. Establishing a peer-to-peer connection in this scenario is very tricky, since inbound traffic from a public IP address and port that isn't in the NAT table will be dropped, acting similarly to a firewall. Most NAT-enabled routers have a feature called port forwarding that allows manual configuration of which internal IP address and port should be mapped to a

particular external port, but this is woefully inadequate given the dynamic nature of IP addresses (DHCP) and wide range of VoIP protocols and software that are in use.

### 2.4.1 ICE

A modern technique to get around the issue of NAT is Interactive Connectivity Establishment (ICE). ICE attempts to find the most direct and efficient way for two peers to communicate with each other. It does this by producing a list of candidate addresses—a combination of IP addresses and ports—that get included in the SDP offer that is sent to the remote peer. ICE describes three different types of candidates that are checked in order. The first type is a host candidate. These are derived from local network interfaces attached to the computer generating the offer and might include Ethernet, Wi-Fi, or any active VPNs [19].

### 2.4.2 STUN

The next type of candidate address comes from a protocol called Session Traversal Utilities for NAT (STUN). STUN provides a mechanism to discover if a computer is located behind a NAT-enabled router by asking a STUN server located on the public internet to respond with the IP address and port of the client as seen from the internet. The address it provides is known to ICE as a server reflexive candidate. Crucially, STUN uses UDP, which means that the entry created in the client's NAT table(s) can be reused by the remote peer later for UDP VoIP traffic without it getting dropped (though, this

doesn't always work, particularly for symmetric NATs) [20]. STUN provides a method using keepalive packets to prevent the NAT table entry from being removed [21].

### 2.4.3 TURN

The last type of candidate address is a relayed candidate. This uses an extension to STUN called Traversal Using Relays around NAT (TURN). Like STUN, this relies on a public TURN server being available on the public internet. It can also provide the server reflexive candidates, so contacting a separate STUN server is not necessary. The main feature of TURN is that, if for whatever reason the other candidates fail to establish a connection to the remote peer, then the TURN server can relay traffic between both peers. This, of course, is not direct peer-to-peer communication, so it comes at a cost of higher latency. It is also very expensive for the TURN server, so in many VoIP deployments a TURN server is not included. But, communicating using a relay is often preferable to not being able to communicate at all. Additionally, if all UDP traffic is being blocked, TURN can operate using TCP.

## 3 WebRTC

Web Real-Time Communication (WebRTC) [22] is a free and open-source project created to simplify the task of transmitting audio, video, and other arbitrary data over the internet. WebRTC was born out of Google's 2010 acquisition of Global IP Solutions, a VoIP software company [23]. Much like VoIP, WebRTC is not itself a protocol. Rather, it is a standardized interface (JavaScript API) to a collection of established protocols that

have already been in use for a long time to facilitate real-time communication. Its first specification was drafted by the W3C in 2011 [24]. WebRTC provides an API—now supported by most modern browsers—for real-time, encrypted, peer-to-peer data transmission. Since many modern chat apps such as Discord are browser-based, WebRTC has seen wide adoption in this domain—separate, although compatible, with existing VoIP infrastructure. This report will mainly focus on the facilities of WebRTC that allow for connection establishment and voice communication.

### 3.1 WebRTC Protocols

Internally, WebRTC uses a collection of different protocols to provide both high and low-level functionality for establishing peer-to-peer voice communication channels. To truly understand this process, we must attend to signalling, delivery, and NAT traversal.

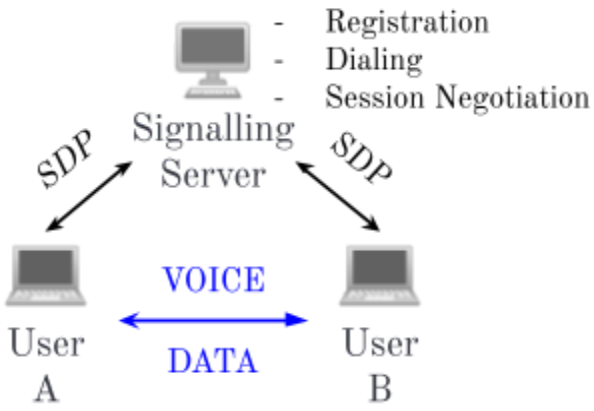


*Figure 3.1.1 Role of the Signalling Server*

#### 3.1.1 Signalling

WebRTC does not provide signaling. This may seem like a baffling choice, since a signalling channel is required to establish a peer connection. However, in the context of

web applications, this is reasonable. In order to use a modern chat platform, clients must already register an account and connect to their server to log in. Thus, this connection can simply be reused as the signalling channel for peer discovery and negotiation, as seen in Figure 3.1.1. In addition, when there are more than two parties present in the voice call, this server can also be used to relay traffic as having a direct connection between all pairs of peers rapidly becomes untenable. Finally, external, public servers are also required for STUN and TURN.

For session negotiation, WebRTC provides APIs to generate SDP offers and answers such as those in Appendix A and B, respectively, to be exchanged via the chosen signalling mechanism. Although it is possible to use a custom signalling protocol, it is still common to see SIP or XMPP used for this purpose.

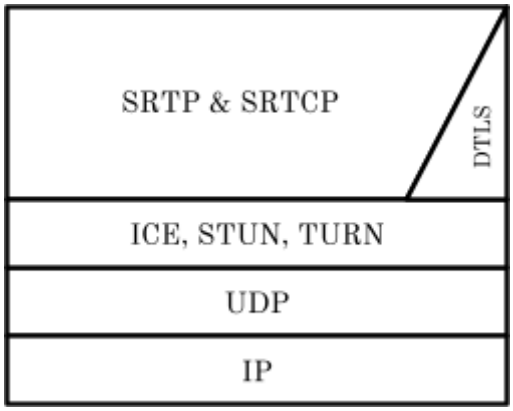


*Figure 3.1.2 Delivery Protocol Stack (Adapted from [21])*

### 3.1.2 Delivery

Figure 3.1.2 illustrates a "vertical slice" of the protocols used to deliver voice data. Since WebRTC requires that the data exchanged between peers is end-to-end encrypted, a secure variant of the RTP/RTCP combination (SRTP/SRTCP [25]) is used. Although

these protocols are typically used on consecutive ports, NAT makes this difficult to arrange. Thus, both protocols can be multiplexed on the same port. Since SRTP and SRTCP use UDP at the transport layer, the Datagram Transport Layer Security (DTLS) protocol [26] is used for cryptographic key exchange during the initiation of the connection [27]. DTLS is the datagram variant of the Transport Layer Security (TLS) protocol notably used by HTTPS [28] for encryption. This handshake occurs both after ICE has successfully established a route to the other peer, and the SDP offer has been answered. This ensures that each party has the information necessary to validate the other's identity.

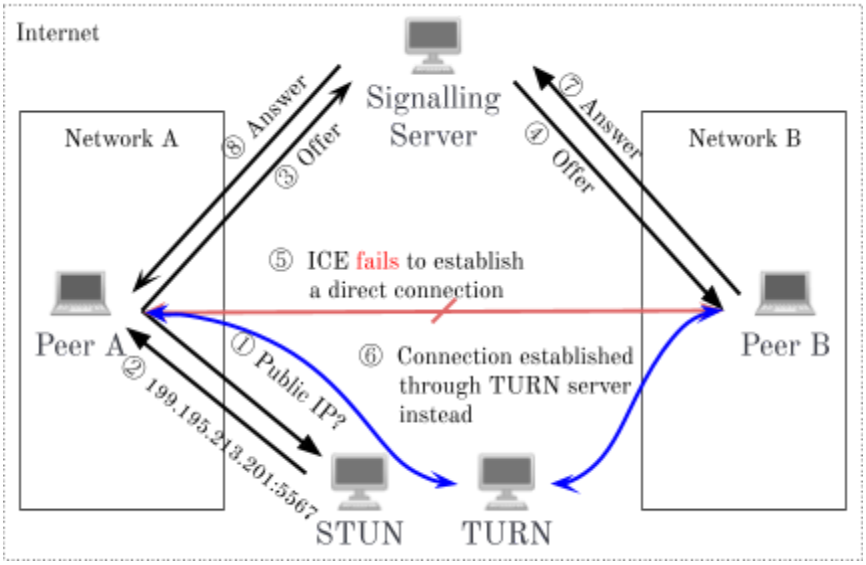


*Figure 3.1.3 WebRTC Connection Establishment (Fall-back to TURN)*

### 3.1.3 NAT Traversal

ICE plays a central role in WebRTC. Without it, it would be impossible to establish a peer-to-peer connection in all but the most trivial cases. Some sources even report (albeit colloquially) that as many as 20% of connections fail when STUN alone is used [29, 30].

As seen in Figure 3.1.3, ICE is relied on heavily for connection establishment. When Peer A attempts to connect to Peer B, they must first gather ICE candidates. Then, these candidates are sent to the other peer in the SDP offer. The other peer can now start to try to establish a direct route to the other host using these candidates. Since gathering ICE candidates takes time due to the round-trip time between Peer A and one (or multiple) STUN servers, WebRTC supports Trickle ICE [31], which is an extension that allows ICE candidates to be sent incrementally via the signalling channel as they are discovered rather than all at once. Finally, ICE is also responsible for keeping connections alive (as NAT rules may expire), searching for faster routes, and re-routing if the route disappears.

## 3.2 Overview of the WebRTC API

The WebRTC API [32] is formally specified in ECMAScript and is implemented in Chrome, Firefox and Safari. For voice communication, there are two main components of the API that are of immediate interest: the `MediaStream` and `PeerConnection`.

### 3.2.1 MediaStream

A `MediaStream` [33] object represents a collection of synchronized audio or video tracks. In the browser, the `getUserMedia` function can be used to create a `MediaStream` given a dictionary of constraints on the requested media, such as type (audio/video), number of channels, sample rate, and other parameters. This will automatically prompt the user for permission to gain access to the requested hardware (such as their microphone). This is very convenient, since the developer can provide very

few constraints, and get a passable default, or specify much more strict requirements as needed.

### 3.2.1 PeerConnection

The `RTCPeerConnection` [34] object represents a connection to a remote peer. The connection can be provided with a configuration containing a set of ICE servers (STUN/TURN servers). Next, tracks can be added to the connection. Finally, an SDP offer can be generated based on the media tracks and the ICE status of the connection. This offer can thus be sent to the peer via the signalling channel. Both sides of the connection have their own `RTCPeerConnection`. Each will set their respective local and remote descriptions based on the SDP offer and response generated.

## 4 WebRTC Demonstration

To further explore the functionality of WebRTC, we constructed a simple demonstration with the ability for two peers to establish a voice connection between them. Conventionally, WebRTC is used with JavaScript APIs in a web browser. While this makes sense for common usage, we want more control over how WebRTC works, as well the ability to use our own code for audio processing in a language we are familiar with. As such, we used a Python implementation of WebRTC called `aiortc` [35]. This project follows the official WebRTC standard closely, with JavaScript's Promise API [36] replaced by `asyncio` coroutines from Python's standard library. Since even the voice

component alone of WebRTC is extensive and comprises multiple mature protocols, creating our own implementation exceeded the scope of this project.

### 4.1 Implementation Architecture

In our implementation, both the server and the client are rolled into a single program. One instance of the program will pull up an HTTP server using `aiohttp` [37], while another instance, run with the `--connect` command line option, will attempt to establish a connection using this web server as the signalling server. The SDP offer is sent to the `/offer` endpoint via POST request, and the response received contains the SDP answer. Thus, we are using SDP over HTTP in order to establish the connection. Although our tests have been on a local network, we can still use Google's public STUN server in order to observe ICE in action.

We can see the connection establishment process in action in Appendix D and E. In this example, both the client and the server are running on the same machine. Hooks are attached to the different changes in ICE, so that we can see the state changes. Although multiple ICE candidates are advanced, a connection is established through the local network and all others fail. At the point where we see "`Connection state is connected`," we can hear the voice coming through the system.

### 4.2 Implementation Challenges

Although `aiortc` has proven to be surprisingly capable, there were a few areas that were lacking. In particular, retrieving and playing audio from the host in a cross-platform manner proved to be an issue. `aiortc` manages audio frames using PyAV [38], a Python

API for `ffmpeg` [39]. Although `ffmpeg` has very broad support for different audio/video codecs, we were surprised to learn that it cannot play audio on Windows. Instead, it relies on a different tool called `ffplay` [40], which does not have a Python API. Although other options for manipulating audio in Python exist such as PyAudio [41] and sounddevice [42], they are not natively compatible with `asyncio` since they are APIs to C libraries and would require wrappers around their callback interface in order to work correctly. In addition, we would need to convert PyAudio frames to their own respective formats as they arrive, which is latency-critical, a task poorly suited to Python. Thus, audio playback was tested on a Linux machine.

## 5 Conclusion

Although the existence of the telephone makes VoIP seem like a deceptively simple problem, the wide collection of protocols and techniques required to make it work in a real-time manner indicate that it is anything but straightforward. Issues that we did not cover, such as quality of service and encryption, further increase its difficulty. It is fortunate, then, that projects like WebRTC exist, allowing developers to create the next generation of VoIP software without having to worry about every small detail. Soon, there will come a time when the PSTN in North America, along with the rest of the world, is shut down, leaving our ability to communicate amongst ourselves fully reliant on VoIP systems.

**Project Contribution Breakdown**

*Project Proposal*

- Both Victor & Matthew collaborated to produce initial references, motivate the topic and set out a schedule.

*Bi-Weekly Update 1*

- Matthew focused on H.323 and XMPP.
- Victor focused on SIP, RTP, and WebRTC.

*Midterm Update*

- Matthew did the initial research / proof of concept leading to `aiortc` being chosen for a demo implementation.
- Victor worked on converting JavaScript client example code to Python.
- Both contributed to update text.

*Bi-Weekly Update 3*

- Victor worked on completing demo code, and did additional research into WebRTC.
- Matthew helped produce update text.

*Project Demo*

- Matthew created and presented historical background slides.
- Victor created and presented WebRTC slides and diagrams.

*Project Report*

- Matthew created sections 1, 2, and 5 (background, protocols, conclusion).
- Victor created sections 3 and 4 (WebRTC, Demo Implementation), figures, references, and appendices.

**References**


[1] "Ahoy! Alexander Graham Bell and the first telephone call | Science Museum." Accessed: Dec. 11, 2023. [Online]. Available: https://www.sciencemuseum.org.uk/objects-and-stories/ahoy-alexander-graham-bell-and-first-telephone-call

[2] "Fixed telephone subscriptions worldwide 2000-2022," Statista. Accessed: Dec. 11, 2023. [Online]. Available: https://www.statista.com/statistics/273014/number-of-fixed-telephone-lines-worldwide-since-2000/

[3] G. DiNardi, "What is ISDN? What are its Advantages? (Updated)," Nextiva Blog. Accessed: Dec. 11, 2023. [Online]. Available: https://www.nextiva.com/blog/what-is-isdn.html

[4] "The UK's PSTN network will switch off in 2025," BT Business. Accessed: Dec. 11, 2023. [Online]. Available: https://business.bt.com/why-choose-bt/insights/digital-transformation/uk-pstn-switch-off/

[5] A. Omar, "Voice OVER IP (VOIP)", Accessed: Dec. 11, 2023. [Online]. Available: https://www.academia.edu/39621401/Voice_OVER_IP_VOIP_

[6] S. Ludwig, J. Beda, P. Saint-Andre, R. McQueen, S. Egan, and J. Hildebrand, "Jingle." Accessed: Dec. 11, 2023. [Online]. Available: https://xmpp.org/extensions/xep-0166.html

[7] "Talk Help." Accessed: Dec. 11, 2023. [Online]. Available: https://support.google.com/talk/

[8] G. Keizer, "Update: Skype to retire recent editions for Windows, Mac; upgrades mandatory," Computerworld. Accessed: Dec. 11, 2023. [Online]. Available: https://www.computerworld.com/article/2491100/update-skype-to-retire-recent-editions-for-windows-mac-upgrades-mandatory.html

[9] [1] "TeamSpeak - TeamSpeak 3." Accessed: Dec. 11, 2023. [Online]. Available: https://web.archive.org/web/20080525080530/http://www.teamspeak.com/?page=teamspeak3

[10] R. Arora, "Voice Over IP : Protocols and Standards." Accessed: Dec. 11, 2023. [Online]. Available: https://www.cse.wustl.edu/~jain/cis788-99/ftp/voip_protocols/

[11] "Q.23 : Technical features of push-button telephone sets." Accessed: Dec. 11, 2023. [Online]. Available: https://www.itu.int/rec/T-REC-Q.23/en

[12] A. Montazerolghaem, M. H. Y. Moghaddam, and A. Leon-Garcia, "OpenSIP: Toward Software-Defined SIP Networking," IEEE Transactions on Network and Service Management, vol. 15, no. 1, pp. 184–199, Mar. 2018, doi: 10.1109/TNSM.2017.2741258.

[13] E. Schooler et al., "SIP: Session Initiation Protocol," Internet Engineering Task Force, Request for Comments RFC 3261, Jul. 2002. doi: 10.17487/RFC3261.

[14] A. C. Begen, P. Kyzivat, C. Perkins, and M. J. Handley, "SDP: Session Description Protocol," Internet Engineering Task Force, Request for Comments RFC 8866, Jan.

2021. doi: 10.17487/RFC8866.
[15] "First voice over internet protocol (VoIP)," Guinness World Records. Accessed: Dec. 11, 2023. [Online]. Available: https://www.guinnessworldrecords.com/world-records/112518-first-voice-over-internet-protocol-voip
[16] H. Schulzrinne, S. L. Casner, R. Frederick, and V. Jacobson, "RTP: A Transport Protocol for Real-Time Applications," Internet Engineering Task Force, Request for Comments RFC 3550, Jul. 2003. doi: 10.17487/RFC3550.
[17] C. Perkins and M. Westerlund, "Multiplexing RTP Data and Control Packets on a Single Port," Internet Engineering Task Force, Request for Comments RFC 5761, Apr. 2010. doi: 10.17487/RFC5761.
[18] K. B. Egevang and P. Francis, "The IP Network Address Translator (NAT)," Internet Engineering Task Force, Request for Comments RFC 1631, May 1994. doi: 10.17487/RFC1631.
[19] A. Keränen, C. Holmberg, and J. Rosenberg, "Interactive Connectivity Establishment (ICE): A Protocol for Network Address Translator (NAT) Traversal," Internet Engineering Task Force, Request for Comments RFC 8445, Jul. 2018. doi: 10.17487/RFC8445.
[20] J. Rosenberg, C. Huitema, R. Mahy, and J. Weinberger, "STUN - Simple Traversal of User Datagram Protocol (UDP) Through Network Address Translators (NATs)," Internet Engineering Task Force, Request for Comments RFC 3489, Mar. 2003. doi: 10.17487/RFC3489.
[21] I. Grigorik, "High Performance Browser Networking (O'Reilly)," High Performance Browser Networking. Accessed: Dec. 11, 2023. [Online]. Available: https://hpbn.co/
[22] "WebRTC," WebRTC. Accessed: Dec. 11, 2023. [Online]. Available: https://webrtc.org/
[23] J. Tartakoff, "Google to buy VoIP technology firm Global IP solutions for $68.2m," The Guardian, May 18, 2010. Accessed: Dec. 11, 2023. [Online]. Available: https://www.theguardian.com/media/pda/2010/may/18/google-acquisition-globalip
[24] I. Hickson, "WebRTC 1.0: Real-time Communication Between Browsers." Accessed: Dec. 11, 2023. [Online]. Available: https://www.w3.org/TR/2011/WD-webrtc-20111027/
[25] E. Carrara, K. Norrman, D. McGrew, M. Naslund, and M. Baugher, "The Secure Real-time Transport Protocol (SRTP)," Internet Engineering Task Force, Request for Comments RFC 3711, Mar. 2004. doi: 10.17487/RFC3711.
[26] E. Rescorla and N. Modadugu, "Datagram Transport Layer Security Version 1.2," Internet Engineering Task Force, Request for Comments RFC 6347, Jan. 2012. doi: 10.17487/RFC6347.
[27] D. McGrew and E. Rescorla, "Datagram Transport Layer Security (DTLS) Extension to Establish Keys for the Secure Real-time Transport Protocol (SRTP)," Internet Engineering Task Force, Request for Comments RFC 5764, May 2010. doi: 10.17487/RFC5764.
[28] E. Rescorla, "HTTP Over TLS," Internet Engineering Task Force, Request for

Comments RFC 2818, May 2000. doi: 10.17487/RFC2818.
[29] T. Levent-Levi, “WebRTC TURN: Why you NEED it and when you DON’T need it,” BlogGeek.me. Accessed: Dec. 11, 2023. [Online]. Available: https://bloggeek.me/webrtc-turn/
[30] “WebRTC TURN server: Everything you need to know,” 100ms.live. Accessed: Dec. 11, 2023. [Online]. Available: https://100ms.live/
[31] E. Ivov, E. Rescorla, J. Uberti, and P. Saint-Andre, “Trickle ICE: Incremental Provisioning of Candidates for the Interactive Connectivity Establishment (ICE) Protocol,” Internet Engineering Task Force, Request for Comments RFC 8838, Jan. 2021. doi: 10.17487/RFC8838.
[32] I. Hickson, “WebRTC: Real-Time Communication in Browsers.” Accessed: Dec. 11, 2023. [Online]. Available: https://www.w3.org/TR/webrtc/
[33] “MediaStream - Web APIs | MDN.” Accessed: Dec. 11, 2023. [Online]. Available: https://developer.mozilla.org/en-US/docs/Web/API/MediaStream
[34] “RTCPeerConnection - Web APIs | MDN.” Accessed: Dec. 11, 2023. [Online]. Available: https://developer.mozilla.org/en-US/docs/Web/API/RTCPeerConnection
[35] “aiortc.” aiortc, Dec. 11, 2023. Accessed: Dec. 11, 2023. [Online]. Available: https://github.com/aiortc/aiortc
[36] “Promise - JavaScript | MDN.” Accessed: Dec. 11, 2023. [Online]. Available: https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Promise
[37] “Async http client/server framework.” aio-libs, Dec. 12, 2023. Accessed: Dec. 11, 2023. [Online]. Available: https://github.com/aio-libs/aiohttp
[38] “PyAV.” PyAV, Dec. 12, 2023. Accessed: Dec. 11, 2023. [Online]. Available: https://github.com/PyAV-Org/PyAV
[39] “FFmpeg.” Accessed: Dec. 11, 2023. [Online]. Available: https://www.ffmpeg.org/
[40] “ffplay Documentation.” Accessed: Dec. 11, 2023. [Online]. Available: https://ffmpeg.org/ffplay.html
[41] “PyAudio: Cross-platform audio I/O for Python, with PortAudio.” Accessed: Dec. 11, 2023. [Online]. Available: https://people.csail.mit.edu/hubert/pyaudio/
[42] “Play and Record Sound with Python — python-sounddevice, version 0.4.6.” Accessed: Dec. 11, 2023. [Online]. Available: https://python-sounddevice.readthedocs.io/en/0.4.6/index.html

## Appendix A - SDP Offer

Listing of an SDP offer.

```
v=0
o=- 8498233979625313580 2 IN IP4 127.0.0.1
s=-
t=0 0
a=group:BUNDLE 0
a=extmap-allow-mixed
a=msid-semantic: WMS 140ad551-512d-4ebe-8fad-04f60ccd1caf
m=audio 55175 UDP/TLS/RTP/SAVPF 111 63 9 0 8 13 110 126
c=IN IP4 192.168.3.106
a=rtcp:9 IN IP4 0.0.0.0
a=candidate:3742922256 1 udp 2122260223 192.168.3.106 55175 typ host
generation 0 network-id 1 network-cost 10
a=candidate:565355140 1 tcp 1518280447 192.168.3.106 9 typ host tcptype active
generation 0 network-id 1 network-cost 10
a=ice-ufrag:5DH+
a=ice-pwd:coLPjdqH9norYQpnPyFS4JP9
a=ice-options:trickle
a=fingerprint:sha-256
A6:2E:2E:1F:1E:29:D1:53:B1:71:B5:12:9B:8D:DE:44:B3:48:A0:99:92:42:00:04:A5:6F:
F6:A7:59:35:0D:DA
a=setup:actpass
a=mid:0
a=extmap:1 urn:ietf:params:rtp-hdrext:ssrc-audio-level
a=extmap:2 http://www.webrtc.org/experiments/rtp-hdrext/abs-send-time
a=extmap:3
http://www.ietf.org/id/draft-holmer-rmcat-transport-wide-cc-extensions-01
a=extmap:4 urn:ietf:params:rtp-hdrext:sdes:mid
a=sendrecv
a=msid:140ad551-512d-4ebe-8fad-04f60ccd1caf
96260768-0bcc-40ac-ade0-9d89c4f1514d
a=rtcp-mux
a=rtpmap:111 opus/48000/2
a=rtcp-fb:111 transport-cc
a=fmtp:111 minptime=10;useinbandfec=1
a=rtpmap:63 red/48000/2
a=fmtp:63 111/111
a=rtpmap:9 G722/8000
a=rtpmap:0 PCMU/8000
a=rtpmap:8 PCMA/8000
a=rtpmap:13 CN/8000
a=rtpmap:110 telephone-event/48000
a=rtpmap:126 telephone-event/8000
a=ssrc:2962455826 cname:QCVZMO/hYO4WHDB0
a=ssrc:2962455826 msid:140ad551-512d-4ebe-8fad-04f60ccd1caf
96260768-0bcc-40ac-ade0-9d89c4f1514d
```

## Appendix B - SDP Answer

Listing of an SDP answer.

```
v=0
o=- 3909534172 3909534172 IN IP4 0.0.0.0
s=-
t=0 0
a=group:BUNDLE 0
a=msid-semantic:WMS *
m=audio 45632 UDP/TLS/RTP/SAVPF 111 0 8
c=IN IP4 192.168.3.106
a=sendrecv
a=extmap:1 urn:ietf:params:rtp-hdrext:ssrc-audio-level
a=extmap:4 urn:ietf:params:rtp-hdrext:sdes:mid
a=mid:0
a=msid:b8cff6c1-be83-499e-b258-ec158b3aaf87
7f491ad5-f76b-4af9-a09d-f6922ca77662
a=rtcp:9 IN IP4 0.0.0.0
a=rtcp-mux
a=ssrc:3118474735 cname:54fd8049-429c-469a-9dc0-fa318353484d
a=rtpmap:111 opus/48000/2
a=rtpmap:0 PCMU/8000
a=rtpmap:8 PCMA/8000
a=candidate:62072eb9c6e1999dc6bfffd7e63b7452 1 udp 2130706431 192.168.3.106
45632 typ host
a=candidate:b01e63478a980080b68324ebe32767e7 1 udp 1694498815 99.96.187.180
45632 typ srflx raddr 192.168.3.106 rport 45632
a=end-of-candidates
a=ice-ufrag:ljko
a=ice-pwd:T3qJ5NFNo7zgUuyJH6qLeJ
a=fingerprint:sha-256
28:7F:06:CB:23:F7:97:DE:83:6B:39:97:F6:29:EF:54:4C:1A:BA:E5:DE:AE:C6:57:35:2C:
79:39:B2:8A:D3:D9
a=setup:active
```

## Appendix D - Server ICE Connection Log

Listing of server log.

```
======== Running on http://0.0.0.0:8080 ========
(Press CTRL+C to quit)
INFO:pc:PeerConnection(81615e60-5574-4b6f-b68a-78da021ffa70) Created for 127.0.0.1
INFO:pc:PeerConnection(81615e60-5574-4b6f-b68a-78da021ffa70) Track audio received
INFO:aioice.ice:Connection(0) Check CandidatePair(('10.0.0.32', 37674) -> ('10.0.0.32', 37840))
State.FROZEN -> State.WAITING
INFO:aioice.ice:Connection(0) Check CandidatePair(('b9b5:6a8b:15c2:e4e8:ace1:ddc3:dc42:a7e1',
35403) -> ('b9b5:6a8b:15c2:e4e8:ace1:ddc3:dc42:a7e1', 43972)) State.FROZEN -> State.WAITING
INFO:aioice.ice:Connection(0) Check CandidatePair(('b9b5:6a8b:15c2:e4e8:ace1:ddc3:dc42:a76f',
46858) -> ('b9b5:6a8b:15c2:e4e8:ace1:ddc3:dc42:a7e1', 43972)) State.FROZEN -> State.WAITING
INFO:aioice.ice:Connection(0) Check CandidatePair(('10.0.0.32', 37674) -> ('99.96.187.180',
37840)) State.FROZEN -> State.WAITING
INFO:pc:PeerConnection(81615e60-5574-4b6f-b68a-78da021ffa70) Connection state is connecting
INFO:aioice.ice:Connection(0) Check CandidatePair(('10.0.0.32', 37674) -> ('10.0.0.32', 37840))
State.WAITING -> State.IN_PROGRESS
INFO:aioice.ice:Connection(0) Check CandidatePair(('10.0.0.32', 37674) -> ('10.0.0.32', 37840))
State.IN_PROGRESS -> State.SUCCEEDED
INFO:aioice.ice:Connection(0) Check CandidatePair(('b9b5:6a8b:15c2:e4e8:ace1:ddc3:dc42:a7e1',
35403) -> ('b9b5:6a8b:15c2:e4e8:ace1:ddc3:dc42:a7e1', 43972)) State.WAITING -> State.FAILED
INFO:aioice.ice:Connection(0) Check CandidatePair(('b9b5:6a8b:15c2:e4e8:ace1:ddc3:dc42:a76f',
46858) -> ('b9b5:6a8b:15c2:e4e8:ace1:ddc3:dc42:a7e1', 43972)) State.WAITING -> State.FAILED
INFO:aioice.ice:Connection(0) Check CandidatePair(('b9b5:6a8b:15c2:e4e8:ace1:ddc3:dc42:a7e1',
35403) -> ('b9b5:6a8b:15c2:e4e8:ace1:ddc3:dc42:a76f', 39361)) State.FROZEN -> State.FAILED
INFO:aioice.ice:Connection(0) Check CandidatePair(('b9b5:6a8b:15c2:e4e8:ace1:ddc3:dc42:a76f',
46858) -> ('b9b5:6a8b:15c2:e4e8:ace1:ddc3:dc42:a76f', 39361)) State.FROZEN -> State.FAILED
INFO:aioice.ice:Connection(0) Check CandidatePair(('10.0.0.32', 37674) -> ('99.96.187.180',
37840)) State.WAITING -> State.FAILED
INFO:aioice.ice:Connection(0) ICE completed
INFO:pc:PeerConnection(81615e60-5574-4b6f-b68a-78da021ffa70) Connection state is connected
```

## Appendix E - Client ICE Connection Log

Listing of client log.

```
INFO:pc:PeerConnection(b0d6181c-4399-43ee-b580-c12310198e88) Track audio received
INFO:aioice.ice:Connection(0) Check CandidatePair(('10.0.0.32', 37840) -> ('10.0.0.32', 37674))
State.FROZEN -> State.WAITING
INFO:aioice.ice:Connection(0) Check CandidatePair(('b9b5:6a8b:15c2:e4e8:ace1:ddc3:dc42:a7e1',
43972) -> ('b9b5:6a8b:15c2:e4e8:ace1:ddc3:dc42:a7e1', 35403)) State.FROZEN -> State.WAITING
INFO:aioice.ice:Connection(0) Check CandidatePair(('b9b5:6a8b:15c2:e4e8:ace1:ddc3:dc42:a76f',
39361) -> ('b9b5:6a8b:15c2:e4e8:ace1:ddc3:dc42:a7e1', 35403)) State.FROZEN -> State.WAITING
INFO:aioice.ice:Connection(0) Check CandidatePair(('10.0.0.32', 37840) -> ('99.96.187.180',
37674)) State.FROZEN -> State.WAITING
INFO:aioice.ice:Connection(0) Check CandidatePair(('10.0.0.32', 37840) -> ('10.0.0.32', 37674))
State.WAITING -> State.IN_PROGRESS
INFO:aioice.ice:Connection(0) Check CandidatePair(('10.0.0.32', 37840) -> ('10.0.0.32', 37674))
State.IN_PROGRESS -> State.SUCCEEDED
INFO:aioice.ice:Connection(0) Check CandidatePair(('b9b5:6a8b:15c2:e4e8:ace1:ddc3:dc42:a7e1',
43972) -> ('b9b5:6a8b:15c2:e4e8:ace1:ddc3:dc42:a7e1', 35403)) State.WAITING -> State.FAILED
INFO:aioice.ice:Connection(0) Check CandidatePair(('b9b5:6a8b:15c2:e4e8:ace1:ddc3:dc42:a76f',
39361) -> ('b9b5:6a8b:15c2:e4e8:ace1:ddc3:dc42:a7e1', 35403)) State.WAITING -> State.FAILED
INFO:aioice.ice:Connection(0) Check CandidatePair(('b9b5:6a8b:15c2:e4e8:ace1:ddc3:dc42:a7e1',
43972) -> ('b9b5:6a8b:15c2:e4e8:ace1:ddc3:dc42:a76f', 46858)) State.FROZEN -> State.FAILED
INFO:aioice.ice:Connection(0) Check CandidatePair(('b9b5:6a8b:15c2:e4e8:ace1:ddc3:dc42:a76f',
39361) -> ('b9b5:6a8b:15c2:e4e8:ace1:ddc3:dc42:a76f', 46858)) State.FROZEN -> State.FAILED
INFO:aioice.ice:Connection(0) Check CandidatePair(('10.0.0.32', 37840) -> ('99.96.187.180',
37674)) State.WAITING -> State.FAILED
INFO:aioice.ice:Connection(0) ICE completed
```