WebRTC: A Study On Real-Time Peer-To-Peer Communication

Amit Sureshchandra Kesarwani[1], Surajnarayan Raut[2], Prof. Sneha Yadav[3]
[1, 2, 3] Dept of MCA
[1, 2, 3] Mumbail University IDOL Mumbai, India

Abstract- *Real-time communication is one of the vital component in system design, enabling seemless interaction across applications involving video conferencing, data transfer. WebRTC has emerged as technology that facilitates peer to peer communi- cation without relying on third party plugins.*

It reduces latency & optimizes the bandwidth usage.

This paper explored the architecture of WebRTC, NAT traver- sal(STUN, TURN) & process of media exchange. This paper demonstrates the use of WebRTC for seemless & low latency communication.

Keywords- webrtc, p2p, real-time, communication

I. INTRODUCTION

In current time for real-time communication we have many technologies like web-sockets, messaging queues, server-sent- events, gRPC, etc. These technologies involves central server that is being used to relay the messages. In case we want to send message directly to receipant without involving any relay server then the choice would be WebRTC. A WebRTC web application (typically written as a mix of HTML and JavaScript) interacts with web browsers through the standardized WebRTC API, allowing it to properly exploit and control the real-time browser function[1]. The WebRTC web application also interacts with the browser, using both WebRTC and other standardized APIs[1]. The WebRTC API must therefore provide a wide set of functions, like connection management (in a peer-to-peer fashion), encoding/decoding capabilities negotiation, selection and control, media control, firewall and NAT element traversal, etc[1].

In this paper we tried to create demo of webRTC application using below

- Java & spring-boot based websocket backend app as signaling server
- HTML for UI element
- VueJS for webapp
- webRTC for peer-to-peer media transfer
- For two user only with roomId concept

Before webRTC there are several technologies available

- Flash based solution : Adobe Flash Player and Flash Me- dia Server had supported RTMP (Real-Time Messaging Protocol) for the live streaming of audio and video, which was used before the creation of WebRTC.

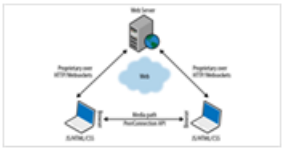

Fig. 1. WebRTC Trapezoid model

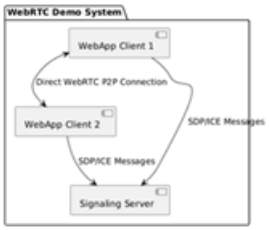

Fig. 2. Demo App Architecture Diagram

- Proprietary Peer-to-Peer System : Skype uses a propri- etary protocol for the transmission of multimedia streams, plus it requires the installation of a mobile application or desktop to access services such as phone calls, messages, and video conferences[2].

II. WebRTC ARCHITECTURE & KEY COMPONENTS

In the WebRTC Trapezoid model(Fig. 1)[4], both browsers are running a web application, which is downloaded from a different web server[1]. The purpose of Signaling messages are used to set up and terminate communications. They are trans- ported by the HTTP or WebSocket protocol via web servers that can modify, translate, or manage them as needed[1]. The Signaling process between browser and server is not standardized in WebRTC. A PeerConnection allows media to flow directly between browsers without any intervening servers[1].

In our demo we have used the spring-boot based websocket server as signaling server[3].

A. *Media Stream*

Fig. 3. Connection Seq Diagram

allows the browsers to discover enough information about the topology of the network where they are deployed to find the best exploitable communication path[1].

The Session Traversal Utilities for NAT (STUN) protocol (RFC5389) allows a host application to discover the presence of a network address translator on the network, and in such a case to obtain the allocated public IP and port tuple for the current connection. To do so, the protocol requires assistance from a configured, third-party STUN server that must reside on the public network[1]. The Traversal Using Relays around NAT (TURN) protocol (RFC5766) allows a host behind a NAT to obtain a public IP address and port from a relay server residing on the public Internet[1]. Thanks to the relayed transport address, the host can then receive media from any peer that can send packets to the public Internet[1].

The PeerConnection mechanism uses the ICE protocol MediaStream is an abstract representation of an actual stream of data of audio and/or video[1]. It serves as a handle for managing actions on the media stream, such as displaying the stream's content, recording it, or sending it to a remote peer[1]. A Media Stream may be extended to represent a stream that either comes from (remote stream) or is sent to (local stream) a remote node[1].

A LocalMediaStream represents a media stream from a local media-capture device (e.g., webcam, microphone, etc.). To create and use a local stream, the web application must request access from the user through the getUserMedia() function[1]. The application specifies the type of media—audio or video—to which it requires access[1]. The devices selector in the browser interface serves as the mechanism for granting or denying access. Once the application is done, it may revoke its own access by calling the stop() function on the LocalMediaStream[1].

Browsers provide a media pipeline from sources to sinks. In a browser, sinks are the ¡img¿, ¡video¿, and ¡audio¿ tags. A source can be a physical webcam, a microphone, a local video or audio file from the user's hard drive, a network resource, or a static image. The media produced by these sources typically do not change over time. These sources can be considered static. The sinks that display such sources to the users (the actual tags themselves) have a variety of controls for manip- ulating the source content. The getUserMedia() API method adds dynamic sources such as microphones and cameras. The characteristics of these sources can change in response to application needs. These sources can be considered dynamic in nature[1].

B. Peer Connection

A PeerConnection enables two users to communicate di- rectly browser to browser. These communications are coordi- nated via a signaling server like websocket, after establishing connection media streams can be sent directly to the remote browser.

The PeerConnection mechanism uses the ICE protocol together with the STUN and TURN servers to let UDP- based media streams traverse NAT boxes and firewalls[1]. ICE together with the STUN and TURN servers to let UDP- based media streams traverse NAT boxes and firewalls. ICE allows the browsers to discover enough information about the topology of the network where they are deployed to find the best exploitable communication path[1]. Using ICE also provides a security measure, as it prevents untrusted web pages and applications from sending data to hosts that are not expecting to receive them[1].

Calling new RTCPeerConnection(configuration) creates an RTCPeerConnection object. The configuration has the infor- mation to find and access the STUN and TURN servers (there may be multiple servers of each type, with any TURN server also acting as a STUN server). Optionally, it also takes a MediaConstraints object "Media Constraints". When the RTCPeerConnection constructor is invoked, it also creates an ICE Agent responsible for the ICE state machine, controlled directly by the browser. The ICE Agent will proceed with gathering the candidate addresses when the IceTransports con- straint is not set to "none." An RTCPeerConnection object has two associated stream sets. A local streams set, representing streams that are currently sent, and a remote streams set, representing streams that are currently received through this RTCPeerConnection object. The stream sets are initialized to empty sets when the RTCPeerConnection object is created[1].

C. Data Channel

The DataChannel API is designed to provide a generic trans- port service allowing web browsers to exchange generic data in a bidirectional peer-to-peer fashion[1]. The encapsulation of SCTP over DTLS over UDP together with ICE provides a NAT traversal solution, as well as confidentiality, source authentication, and integrity protected transfers. Moreover, this solution allows the data transport to interwork smoothly with the parallel media transports, and both can potentially also share a single transport layer port number[1].

SCTP has been chosen since it natively supports multi- ple streams with either reliable or partially reliable delivery modes[1]. It provides the possibility of opening several inde- pendent streams within an SCTP association towards a peering

SCTP endpoint. Each stream actually represents a unidirec- tional logical channel providing the notion of in-sequence delivery. A message sequence can be sent either ordered or unordered. The message delivery order is preserved only for all ordered messages sent on the same stream. However, the DataChannel API has been designed to be bidirectional, which means that each DataChannel is composed as a bundle of an incoming and an outgoing SCTP stream[1].

The DataChannel setup is carried out (i.e., the SCTP as- sociation is created) when the CreateDataChannel() function is called for the first time on an instantiated PeerConnection object[1]. Each subsequent call to the CreateDataChannel() function just creates a new DataChannel within the existing SCTP association[1].

The createDataChannel() method creates a new RTCDat- aChannel object with the given label. The RTCDataChan- nelInit dictionary can be used to configure properties of the underlying channel, such as data reliability[1]. The RTC- DataChannel interface represents a bidirectional data channel between two peers. Each data channel has an associated underlying data transport that is used to transport data to the other peer. The properties of the underlying data transport are configured by the peer as the channel is created[1]. The properties of a channel cannot change after the channel has been created. The actual wire protocol between the peers is SCTP[1].

An RTCDataChannel can be configured to operate in dif- ferent reliability modes. A reliable channel ensures that data is delivered to the other peer through retransmissions. An unreliable channel is configured to either limit the number of retransmissions (maxRetransmits) or set a time during which retransmissions are allowed[1]. These properties cannot be used simultaneously and an attempt to do so will result in an error. Not setting any of these properties results in the creation of a reliable channel[1].

III. METHODOLOGY

A. Signaling Server

In (Fig. 3) sequence diagram[7] is presented that demon- strates the flow of webRTC.

- **Connection to signaling server** : first websocket con- nection to signaling server is established with onmessage & onclose handling. onmessage event handler is used to process message for various offer, answer, end call & ICE data exchange.
- **RTCPeerConnection connection** : It is created with stun & turn server config & all localstream track are added to RTCPeerConnection track. At the same 2 event listeners are added on onicecandidate & ontrack to share ice candidate data to other peer & get remote stream from RTCPeerConnection.
- **SDP offer generation from sender** : Using RTCPeer- Connection an offer is created & sent using signaling server.
- **SDP offer transmission through signaling server & acceptance from receiver** : other peer accepts the offer

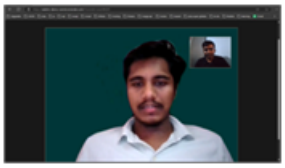

Fig. 4. Live Demo of webRTC

& adds the remote description in RTCPeerConnection. Then sends the SDP answer back over signaling server.

- **sending the SDP answer from receiver to sender back**: Once receiving the SDP answer sender also updates the remote description of RTCPeerConnection.
- **sharing ice candidates over signaling server** : both parties exchanges the ice candidates.
- **Final Step** : a final step with help of ice candidates the peer-to-peer connection is established & Now media track can be exchanged.

IV. CHALLENGES & LIMITATIONS

- **NAT Traversal and Firewall Issues** : Making a connection across many networks can be challenging due to Network Address Translation (NAT) and firewall restrictions. Although techniques like STUN, TURN, and ICE is helpful, it adds the complexity and may not always guarantee the successful connection.
- **Scalability** : WebRTC is designed for peer-to-peer communication but scaling it to support many peers requires additional architectures changes, like mesh networks or SFU/MCU (Selective Forwarding Unit/Multipoint Control Unit), which adds further complexities & challenges.
- **Security Concerns** : Although WebRTC includes built-in encryption for media streams but it relies on secure signaling channels which are not common standard. Any mistakes in the signaling communication or in the handling of ICE candidates can expose lot of vulnerabilities.
- **Quality of Service** : Change in network condition can impact the quality of media streams.

V. CONCLUSION & FUTURE SCOPE

- Implementing adaptive bitrate and error correction strategies.
- Advanced NAT Traversal Techniques
- Scalable Architectures for Multi-Party Communication
- Integration with Emerging Technologies like AI/ML
- Enhanced Security Protocols

VI. ACKNOWLEDGMENT

I sincerely appreciate the guidance and support of Professor Sneha Yadav from Vidyavardhini's College of Engineering Technology. Her valuable insights and encouragement have played a pivotal role in shaping this research.

I am also deeply grateful to my parents for their constant en- couragement, patience, and unwavering belief in my abilities. Their support has been instrumental in my academic journey.

Furthermore, I extend my heartfelt thanks to my colleagues and friends for their collaboration, constructive discussions, and motivation throughout this work. Their shared knowledge and assistance have been invaluable in the successful comple- tion of this research.

REFERENCES

[1] S. Loreto and S. Pietro, Real-Time Communication with WebRTC: Peer- to-Peer in the Browser. O'Reilly Media, 2014.
[2] 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.
[3] Pivotal Software, Inc., "Spring Boot," Spring.io. [Online]. Available: https://spring.io/projects/spring-boot.
[4] PlantUML, "PlantUML: The UML Diagram Generator," Available: https://plantuml.com/.