JOURNAL OF WORLD SCIENCE
https://jws.rivierapublishing.id/index.php/jws
Volume 2 No. 2 February 2022
P-ISSN: 2828-8726
E-ISSN: 2828-9307

TRAVERSAL USING RELAYS AROUND NAT BY COTURN FOR WEBRTC SYSTEMS

Raswa[1], Ahmad Lubis Ghozali[2], Kurnia Adi Cahyanto[3]
Politeknik Negeri Indramayu, Jawa Barat, Indonesia

drraswa@gmail.com[1], lubis@polindra.ac.id[2], kurnia@polindra.ac.id[3]

ABSTRACT
Real-time communication between people in the form of audio, images, video, and data is now a necessity. Web Real-Time Communication Technology, WebRTC, is implemented as an open web standard and is available to build such communication. WebRTC requires a server that functions as both signaling and media servers. Signaling server for establishing communication channels, exchanging information, and synchronizing changes. Media servers exchange media, codecs, bandwidth, and rate control. Different Real-time Protocol (RTP) topologies have different requirements on servers, including those caused by the NAT configuration used. This research focuses on the role of the TURN server using CoTurn to create web-based real-time communication services. How is the quality of WebRTC service when CoTurn is a TURN server to pass signaling and media. The method used is divided into three stages, namely: (1) installation and configuration, (2) building WebRTC, and (3) testing. This study shows that real-time communication in the form of audio and video can be carried out even through a symmetrical NAT configuration. It is just that with more than five users in the WebRTC system, communication is slowing down. The proposed further research is developing and engineering the RTP topology to increase the number of users. The CoTurn application can be installed on a Virtual Private Server (VPS) and configured as a TURN Server as a media server service in the WebRTC system.

Keywords: coturn, quality of service, turn, webrtc.

Corresponding Author: Raswa
E-mail: drraswa@gmail.com

INTRODUCTION

Real-time communication on the network is direct communication through applications that require a network (Aswad et al., 2017); (Aditya & Permadi, 2018). Web Real-Time Communication (WebRTC) is an application that enables peer-to-peer (device-to-device) communication without a server. It enables data exchange between connected devices (Azzam et al., 2019). The role of a server in a WebRTC system is limited to helping two or more devices find each other and set up a direct connection. For most WebRTC applications to work, servers must relay traffic between peers because direct sockets are often only possible between clients if they are on the same local network. A common way around this is to use a NAT Traversal server.

Building an effective NAT traversal solution is fraught with complex problems due to NAT layers and firewalls blocking ports and protocols. Some RTC clients have had significant problems sending voice and video through firewalls and NAT networks. Users may receive calls, but they cannot hear the other person, or they may not be able to connect at all (Widyarto & Anwar, 2022). One of the main problems with end-to-end communication between pairs of endpoints is that, in most cases, those endpoints are not on the public Internet but in a private address space behind a Network Address Translator (Port and NAT).

How two peers can communicate via NAT (Network address translation) is done by different methods: (1) Full-cone NAT, less secure where anyone can connect and send data, (2) Address-restricted-cone NAT, addresses The sending IP must match one of the addressees of the intended NAT table visitor, (3) port-delimited cone NAT, such as address-delimited cone NAT, but its constraints include port numbers, and (4) symmetric NAT, allowing only full pair matches, at most safe. STUN cannot work with symmetric NAT, but TURN works with Symmetrical NAT (Davids et al., 2014). STUN is a relatively lightweight process—since STUN provides a publicly reachable IP address for an applicant, it is no longer involved in the conversation. Symmetrical NAT. The public IP address of the STUN process is not sufficient to establish connectivity here because ports also require translation (Damayanti, 2018).

The protocol that allows two devices to use an intermediary to exchange bids and answers even though NAT separates the two devices is ICE (Interactive Connectivity Establishment). ICE defines a systematic way of discovering possible communication options between two endpoints (via NAT and Firewall), including using relays where necessary (Drnasin et al., 2020). STUN provides an endpoint that requests its public IP address. Collected information describing the optimal connection path between peers is entered into objects called ICE candidates. An ICE candidate consists of a local IP address, security, and routing protocols such as reflexive addresses (STUN) and relay addresses (TURN), and supported formats. All ICE candidates or collected information is sent to the peer via SDP (Session Description Protocol).

TURN is a network protocol format (IETF RFC 5766), sometimes used with Transmission Control Protocol (TCP) and User Datagram Protocol (UDP) and used to discover peer-to-peer paths on the Internet. TURN extends the Session Traversal utility protocol for NAT (STUN). The TURN server performs NAT traversal and is responsible for conveying media in multimedia applications (Sivasankar & Sakthivel, 2017). TURN helps to run the server on the known port in the private network via NAT but supports user connection behind NAT to only one peer (Emanuel, 2015). The TURN server lets peers send streams to each other behind firewalls. There is no peer-to-peer stream transmission in this case. All media traffic passes through the TURN server (Alamsyah et al., 2020).

To exchange media, WebRTC uses a session description protocol (SDP) to initiate and run a “bid” and “answer” mechanism between endpoints or peers (Feher et al., 2018). Supported codecs, connectivity, and protocols are added to SDP so clients can decide what media codecs they can send and receive and where to send them. SDPs are created at the application layer and then sent over the cable, back and forth, as they negotiate how the endpoints will connect and transfer the media. That is fine, but the NAT mapping is at the network layer. NAT will only change the IP header of a TCP/IP or UDP/IP packet. NAT does not know what an SDP is or how to modify it so it can safely enter or exit the network. The media will likely be discarded depending on the type of NAT between the two peers. Based on callstats.io data, on average, about 30% of P2P conferences have one endpoint connected via the TURN server. This user cannot communicate without assistance from the TURN relay server.

Traversal Using Relays around NAT (TURN), used with Transmission Control Protocol (TCP) and User Datagram Protocol (UDP), is a protocol that assists in traversing network address translators (NAT) or firewalls for multimedia applications. TURN is useful for clients on networks obfuscated by symmetric NAT devices.

WebRTC has many important components–user devices, signaling servers, application servers, TURN and STUN servers, and sometimes media servers. The ICE and TURN protocols were developed to address this problem. Operating a TURN server with public IP addresses on each site and using client software that supports ICE and TURN provides the best user experience. The WebRTC TURN server is an essential part of almost any WebRTC implementation. TURN is a relay-both clients send data to the TURN server, which passes it on to the other client. STUN is not a relay — the STUN server helps "establish a connection" between clients (by finding and exchanging external host: port pairs), after which they send data to each other directly (Gupta & Vani, 2018).

WebRTC allows media to move from one computer to another, regardless of the NAT that exists between them. The WebRTC application functions as a server that serves traffic between peers. STUN, TURN, and ICE are IETF standard protocols developed to overcome the problem of Network Address Translator (NAT) traversing when establishing peer-to-peer communication sessions (Kfoury & Khoury, 2018); (Nayyef et al., 2018); (Pasha et al., 2017). Symmetric NAT not only translates IP addresses from private to public, and vice versa but also translates ports. The Interactive Connectivity Establishment (ICE) comes into play. ICE is a framework that enables WebRTC to address the complexities of real-world networks. In the case of asymmetric NAT, ICE will use STUN (Session Traversal Utilities for NAT). If the STUN server cannot connect, ICE can switch to Traversal Using Relay NAT (TURN). TURN servers are often used in cases of symmetric NAT (Nalamwar et al., 2016). Whether to use STUN or TURN is governed by a protocol called ICE (Rashid et al., 2020); (Tiamiyu, 2019).

Finding a free TURN server is hard because no server has finally implemented a STUN/TURN server. How to install and configure coturn from scratch to create your STUN/TURN server on Ubuntu 18.04 LTS. Use CoTURN, a free, open-source server to control TURN/STUN servers (TARIM & Tekin, 2019). WebRTC testing is the process of ensuring that WebRTC-based applications always run smoothly. Testing is an automated process that assesses several conditions: the stability of the TURN/STUN server working on the backend and the performance of the Media Server that has been used (Maruschke et al., 2015).

Most of the research conducted by previous researchers on the topic of using STUN Server or WebRTC server platforms provided by vendors as done by (Gopinath et al., 2016) applies the decentralized method of gadgets in building Peer to Peer Streaming Media Using WebRTC. Furthermore, (Chodorek et al., 2022) researched the Prototype Monitoring System for Pollution Sensing and Online Visualization with the Use of UAV and a WebRTC-Based Platform carried out using Kestrel which was built using a universal platform based on UAV and WebRTC. (Garcia et al., 2017) used these tools and methodologies in the context of the Kurento open-source software project and concluded that they are suitable for validating large and complex WebRTC systems at scale. In fact, when the STUN Server as a server that helps establish peer-to-peer device communication fails, relay transmission (TURN) will be used. TURN servers are not free, even building them is expensive. ICE, STUN, TURN protocols in WebRTC systems need to be available, and there is still very little research on inexpensively built TURN Servers. In this research, we propose to use CoTurn to build a TURN Server.

METHODS

The method used is divided into three stages, namely: (1) installation and configuration, (2) building WebRTC, and (3) testing. The installation and configuration stages are carried out, as shown in Figure 1.

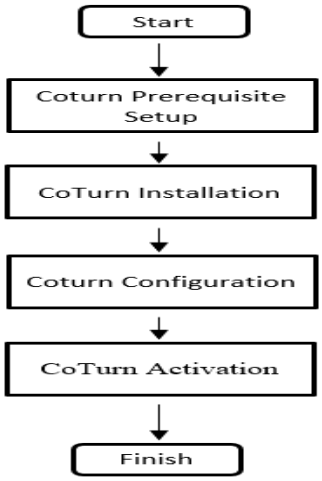

Figure 1. CoTurn Installation and Configuration

To build a TURN server using CoTurn required:

- Ubuntu server (18.04 in our case).
- Know the server's public IP, in this case, the server's public IP is 103.163.139.105.
- Own the domain latcirebon.com
- SSL certificate for WebRTC applications over HTTPS.

Install CoTurn with the following script steps:

```
// install coturn
sudo apt-get install coturn
// edit the default file
sudo nano /etc/default/turn
// run demonically
TURNSERVER_ENABLED=1
// then save
// start turns
systemctl start coturn
```

The turn server configuration file is done to configure applications that function as turn servers:

```
fingerprint
user=USER: PASSWORD
lt-cred-mech
realm=DOMAIN.COM
log-file=/var/log/turnserver/turnserver.log
simple-log
external-ip=IP_SERVER
```

The second stage is building WebRTC, as shown in Figure 2 below.

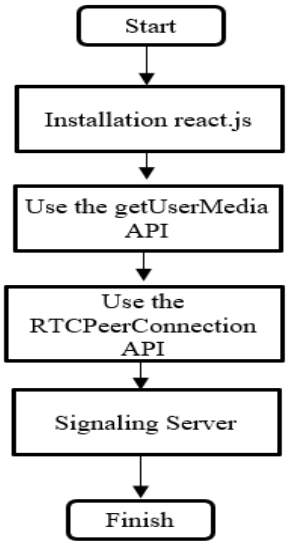

Figure 2. Building a WebRTC System

The third stage is testing the quality of service for the WebRTC system, as shown in Figure 3 below.

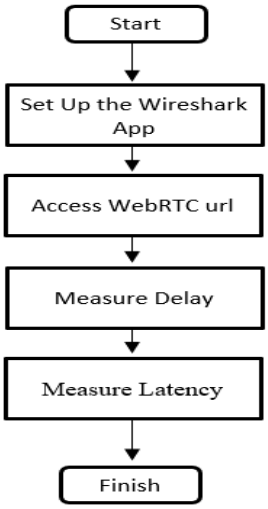

Figure 3. WebRTC System Testing

RESULTS AND DISCUSSION

Coturn is a free and open-source TURN and STUN server implementation for VoIP and WebRTC. This project evolved from rfc5766-turn-server. To check the STUN/TURN CoTurn server functionality. STUN and TURN server functionality can be tested using Trickle ICE (https://webrtc.github.io/samples/src/content/peerconnection/trickle-ice/) and interest (https://icetest.info/). This tool tests the functionality of your TURN server by establishing a peer

connection displaying your TURN server information, and then collecting candidates for WebRTC sessions.

Open a browser to Trickle ICE and add servers in the ICE servers' box, remove the google entry, then click Gather candidates; if everything works, the result is:

0.001 1 hosts 89435858 udp 10.0.0.5 35948 126 | 32542 | 255

0.037 1 srflx 976374523 udp 71.222.38.190 35948 100 | 32542 | 255.0.101 1 host 1272402466 tcp 10.0.0.5 9 9 0 | 32542 | 255

0.101 Done

The use of STUN and TURN (coturn) servers to guarantee the possibility of establishing a connection between two peers is tested via blocking UDP using a packet filter which will force the application to fallback to a TURN/TCP or TURN/TCP server, and setting the iceTransportPolicy RTCPeerConnection which will relay force the connection to only emit relay candidate.

Using getStats() to check the connection and contact a remote TURN server, only one relay server will be used for communication. Use "iceTransports": "relay" in the RTCPeerConnection web application configuration to force both browsers to use the TURN server.

Do the same test but force Firefox to use server TURN or replay. Open a new tab and type about: config. Search for media. Peer connection. Ice. relay_only and set it to true. Firefox only uses the TURN relay. If the WebRTC application is working now, can rest assured that your TURN server is working properly.

Methods for detecting traffic jams in voice and video or image streams fall into two categories: (1) packet loss and (2) delay. Traffic jam detection can be either end-to-end measurement performed at the endpoint or explicit, i.e., measured directly within the network element by monitoring router buffer length. Losses that cause delays can detect congestion before packets are lost due to buffer overflows. However, losses cannot control queue delays because they constantly probe available network bandwidth by filling and depleting internet buffers, resulting in significant delay variations.

The ice server configuration value in the RTCPeerConnection object between two browsers on different hosts is null:

{"candidate": "candidate:77142221 1 udp 2122260223 192.168.137.1 61384 typ host generation 0 ufrag jNaW network-id 1", "sdpMid": "0", "sdpMLineIndex":0}

WebRTC system connectivity, from sender to receiver and peer to peer, is analyzed with the getStats function of RTCPeerConnection. The basic statistical object used in the WebRTC statistical monitoring model is RTCStats.

Table 1: CandidateInfo parameters of the RTCiceCandidate object in CoTurn

Parameter	String value
foundation	Are the two candidates the same
component	ICE transport (RTP=1 or RTCP=2)
protocol	UDP or TCP
priority	higher/lower or null
IP	Source devise IP address/ origin candidate or null
ports	The candidate device port number can be contacted or null
type	host/reflux/prefix/relays

The one-way delay (PSA) is the receive time minus the transmit time, PSA = ti-Ti. PSA variations show differences in delivery intervals and arrival times, VPSA = ti-ti-1 – (Ti-Ti-1).

Table 2. One-way Delay Criteria (PSA)

Mark	delay characteristics
PSA > 0	Queue delay increases
PSA<0	Fewer queue delays
PSA = 0	The delay is maintained at a constant value

The results show that if the bandwidth is sufficient, there are no traffic jams, no large delays, and jitter, the sending and receiving of voice and video are smooth, and everything is fine. However, the reality is that unexpected things always happen, the network goes down, packets get lost, slow down, queues at the routing nodes, and packets get congested. Video conferencing requires low latency and high bandwidth, which happens in real situations. High bandwidth is difficult to guarantee. After the traffic network is congested, the delay becomes large, so it is necessary to control congestion. If the delay is too high in web conferencing, it can impact online communication. The sound is clear, but the video could be clearer.

Video transfer, based on observations of its relationship with the following quality of service characteristics:

Table 3. Characteristics of Video Service Quality

Delays	User Perception
0~400ms	No delay is felt during communication
400~800ms	Delays could be felt, but communication and exchanges were not affected.
800ms and up	Delays can be felt and affect communication.

With 640*480 resolution of 24 frames per second, the video stream is encoded in H.264; the relationship between video bit rate and user perception is as follows:

Table 4. Video Bitrate and User Perception

Bit Rate	User Perception
>800kbps	Users are satisfied with the clarity of the video and do not feel the loss of video image information.
480~800kbps	Users are satisfied with the clarity of the video, and some people may feel the loss of video image information.
< 480kbps	Users are not satisfied with the clarity of the video, often discriminating against the details of the image.

To produce delivery speeds as close as possible to available end-to-end bandwidth while keeping queues as low as possible. Media streams generated by WebRTC applications must share network bandwidth fairly with other concurrents.

The design of using the TURN server as an intermediate server on the WebRTC system wants to see voice, image, or video delivery services coherently and smoothly. Regarding the smoothness of the traffic, see that the latency is kept as low as possible for real-time interaction traffic that still provides the amount of bandwidth used: lowest possible latency or less or equal to 100ms.

CONCLUSION

The CoTurn application can be installed on a Virtual Private Server (VPS) and configured as a TURN Server as a media server service in the WebRTC system. The quality of service in real-time communication using CoTurn is quite good. The delay is very low in communications involving less than five users. In the number of users, more than five levels of latency increase in conditions that are not good for real-time communication. The STUN server test works if the TricleIce test can collect candidates of type "srflx." The TURN server test works if it can collect candidates with the "relay" type. The implication of the findings of this study is that using CoTurn to build a TURN Server can reduce the frequency of peer-to-peer connection failures. WebRTC traffic between different networks requires a TURN Server to relay traffic between peers located on different networks. Peer-to-peer connection failures between networks are a result of network configuration, NAT configuration, and the way WebRTC communicates between clients. Suggestions for further research are the application of QoS measurement methods to improve WebRTC system services. WebRTC technology is a communication that will continue to grow and will be widely used in various real-time communication solutions. In future research, it is possible to allow real-time communication in this application using a parallel system with automatic server sharing.

REFERENCES

Aditya, B. R., & Permadi, A. (2018). Implementation of UTAUT model to understand the use of virtual classroom principle in higher education. *Journal of Physics: Conference Series*, (1) *978*, 12006. Doi: 10.1088/1742-6596/978/1/012006

Alamsyah, F., Kartikasari, D. P., & Bakhtiar, F. A. (2020). Implementasi WebRTC Pada Sistem Broadcast Pembelajaran Untuk Menampilkan Bahasa Isyarat. *Jurnal Pengembangan Teknologi Informasi Dan Ilmu Komputer E-ISSN*, *2548*, 964X, 3 (10), 10331-10336.

Aswad, I., Niswar, M., & Ilham, A. A. (2017). Pengembangan Media Proxy untuk Mendukung Komunikasi Real Time Berbasis Web (WebRTC). *Jurnal Penelitian Enjiniring*, *21* (2), 45-51.

Azzam, F. N., Kartikasari, D. P., & Bakhtiar, F. A. (2019). Implementasi Video Conference dengan File Sharing menggunakan WebRTC. *Jurnal Pengembangan Teknologi Informasi Dan Ilmu Komputer E-ISSN*, *2548*. 3 (10), 10102-10109.

Chodorek, A., Chodorek, R. R., & Yastrebov, A. (2022). The Prototype Monitoring System for Pollution Sensing and Online Visualization with the Use of a UAV and a WebRTC-Based Platform. *Sensors*, *22* (4), 1578. https://doi.org/10.3390/s22041578

Damayanti, F. U. (2018). Research of web real-time communication-The unified communication platform using node. js signaling server. *Journal of Applied Information, Communication and Technology*, *5* (2), 53–61. https://doi.org/10.33555/ejaict.v5i2.54

Davids, C., Ormazabal, G., & State, R. (2014). Real-time communications: topics for research and methods of collaboration. In *ACM SIGCOMM Computer Communication Review* (Vol. 44, Issue 3, pp. 112–115). ACM New York, NY, USA. https://doi.org/10.1145/2656877.2656894

Drnasin, I., Grgic, M., & Gledec, G. (2020). Exploring WebRTC Potential for DICOM File Sharing. *Journal of Digital Imaging*, *33* (3), 697–707.

Emanuel, D. (2015). Real-Time Communication. *JEMS: A Journal of Emergency Medical Services*, 17.

Feher, B., Sidi, L., Shabtai, A., Puzis, R., & Marozas, L. (2018). WebRTC security measures and weaknesses. *International Journal of Internet Technology and Secured Transactions*, *8* (1), 78–102. https://doi.org/10.1504/IJITST.2018.092138

Garcia, B., Gortazar, F., Lopez-Fernandez, L., Gallego, M., & Paris, M. (2017). WebRTC testing: challenges and practical solutions. *IEEE Communications Standards Magazine*, *1* (2), 36–42. DOI : 10.1109/MCOMSTD.2017.1700005

Gopinath, S., Senthooran, V., Lojenaa, N., & Kartheeswaran, T. (2016). *Real-time Lecture Delivery Approach to e-Learning Using WebRTC*.

Gupta, B., & Vani, M. P. (2018). An overview of web sockets: The future of real-time communication. *Int. Res. J. Eng. Technol. IRJET*, *5* (12).

Kfoury, E., & Khoury, D. (2018). Securing natted iot devices using ethereum blockchain and distributed turn servers. *2018 10th International Conference on Advanced Infocomm Technology (ICAIT)*, 115–121. DOI: 10.1109/ICAIT.2018.8686623

Maruschke, M., Haensge, K., Zimmermann, J., & Bach, T. (2015). The role of QoS in WebRTC and IMS-based IPTV services. *Int J Adv Telecommun*, *8* (1 & 2), 59–68.

Nalamwar, S. R., Kalhapure, S., Khatake, A., Gandhi, S., & Jain, K. (2016). Real time communication using embedded system beyond videoconferencing and towards telepresence. *International Journal of Computer Applications*, *134* (14).

Nayyef, Z. T., Amer, S. F., & Hussain, Z. (2018). Peer to peer multimedia real-time communication system based on WebRTC technology. *International Journal of Engineering & Technology*, *7* (2.9), 125–130.

Pasha, M., Shahzad, F., & Ahmad, A. (2017). Analysis of challenges faced by WebRTC videoconferencing and a remedial architecture. *ArXiv Preprint ArXiv:1701.09182*. https://doi.org/10.48550/arXiv.1701.09182

Rashid, N. A. M., Buja, A. G., Arshad, M. A., Rizal Iskandar, R., & Galihkusumah, A. H. (2020). Real-time webrtc-based application for psychological support during COVID-19. *International Journal of Advanced Trends in Computer Science and Engineering*, 208–216.

Sivasankar, D., & Sakthivel, P. (2017). Enhancing Quality of Service in Real Time Communication services for Network on Chip. *Vol*, *9*, 233–240.

Tarim, E. A., & Tekin, H. C. (2019). Performance evaluation of WebRTC-based online consultation platform. *Turkish Journal of Electrical Engineering and Computer Sciences*, *27* (6), 4314–4327. Doi : 10.3906/elk-1903-44

Tiamiyu, O. A. (2019). On The Design And Implementation Of Peer–To Peer Communication Using WebRTC. *Abacus (Mathematics Science Series)*, *44* (1), 507-516.

Widyarto, E. Y., & Anwar, C. (2022). Build Social Networks-Based Audio Engineering (Design And Build An Audio-Based Social Network). *Eduvest - Journal Of Universal Studies*, *2* (1), 48–54. https://doi.org/10.36418/edv.v2i1.330