# Fixing IP Leaks related to WebRTC – DHT

WebRTC - Western Illinois University Graduate Thesis

Dennis McMeekan: December 2020

## Abstract

As previously discussed in a survey describing security concerns related to a WebRTC based application, it is vital that IP Leaks be masked and covered in such a way that a client does not get punished for using the application. “Any two devices talking to each other directly via WebRTC, however, need to know each other’s real IP addresses” [1]. This will allow for the other client, and more frightening, a 3rd party application the ability to detect and misuse the original client’s real IP address. Although this may be apparent currently, our committee has come under the impression that through a similar fashion as the Tor browser, a Distributed Hash Table (DHT) server representation may be a possible solution to the issue.

This would require vigorous and intensive testing and implementation to acquire this sort of product, but this paper will touch the surface and dive into how exactly this would be possible and the different areas of action that would need to occur.

## Current Approach

With the current implementation of a WebRTC application, this is seen where a peer is able to build a RTC Peer Connection with another peer, through the use of a STUN server. In most cases, this is built throughout the application’s provider, or by using Google’s implementation. In our specific example, this connection is made throughout the use of Google’s provided STUN protocols. STUN stands for session transverse utilities for NAT, and this allows the foundation of making the public network address available to other individuals using the application. This process provides the basis for all communication

patterns between each peer, but this does present a large security flaw. Although this is intended to be secure, this does provide a large concern when using a WebRTC application that is not internally hosted. This address can be used to determine the specific location of a network, or even possibly grant unauthorized access. It is common to combat this concern with the use of VPN's (Virtual Private Networks) [2], but this is a paid service and free versions almost always have a data limit. And even through the use of a VPN, it is still possible that an IP leak can still occur. Instead, with the implementation of a Distributed Hash Table server, this can achieve the same anonymity, but with no extra cost to the client at attempting to remain secure as possible. In reality, the user shouldn't have to worry about their personal data remaining protected, as that should be left to the application administrator.

The main reason IP leaks is a concern is because of the STUN protocols, but in creating a new method and switching to a DHT server representation, this will also have an effect on the way the server is implemented. TURN protocols (signaling information) is used to send data to the location at which the STUN protocols were able to relay. Currently, a basic WebSocket server is established, which then protocol calls occur once a connection is initiated. With the new approach, this is something that hopefully could still occur in a similar fashion in terms of signaling, but differently in obtaining each client's IP address.

## New Approach

Our new approach to combat the idea that the public IP address of a client can be exposed, IP leaks, would be by using a Distributed Hash Table server, changing from the direction of using current STUN protocols when obtaining each client's IP address. The Tor browser, which has been made famous for its ability for users to remain anonymous, is a project that gave our committee the idea at which this would be possible. The way that a Tor server is capable of keeping individuals secure in relation to their data

confidentiality is by using a series of nodes that are capable of relaying and communicating with each other. This brings us to where a WebRTC application would be able to implement a very similar type of server, using a Distributed Hash Table.

## Implementation

Expanding upon the Tor browser will provide a basis on our project implementation. Mainly, a series of nodes will need to be implemented. An example set of transfer nodes can exist as follows:

> "connects at random to one of the publicly listed entry nodes, bounces that traffic through a randomly selected middle relay, and finally spits out your traffic through the third and final exit node" [3].

Three levels will need to exist: entry, relay, and exit nodes. Roughly 3 to 5 per level, allowing that when one peer-to-peer connection is established, the nodes do not become overwhelmed by the traffic coming in and out. Through this relay of data transfer, the original address that is being initiated and the receiving end will both remain confidential in relation to each other. With all nodes initiated and being created with the purpose to actively send and receive data, it is vital to create a Distributed Hash Table that creates a series of transfers between each client. In the creation of a Distributed Hash Table, this can be done using C/C++ or Google's programming language Go [4]. This process involves creating a lookup function, which will randomly allow for any of the 9-15 data transfer nodes being online, with at least more than 60% being required to be online to ensure that the route is different each time. This can then be set to alter or change periodically, giving an even greater chance to continuously change course. Once the Distributed Hash Table has the correct nodes initiated, a lookup function needs to be created. This lookup function will have to choose, at random, an entry or exit level node dependent upon who is receiving data, and then create a route based on a DHT algorithm ((n+2^{i-1}) mod 2^m) [4]. Extending upon this, which is something that exactly may not have any prior work in relation to using a DHT, a send and receive function need to be created. These both will have to send and receive the data that is

being transmitted by each client, properly eliminating data after being sent. In doing so, once the data is transferred throughout the nodes, using WebRTC API a peer connection can be built and the audio and video tracks can be sent, with complete anonymity in relation to the client's IP address.

To sum up the structure of how this process would work:

1. Nodes are initiated at different levels and made online/offline.
2. A peer connection is initiated.
3. Lookup function determines a functional route with at least one entry node, one relay node, and one exit node.
4. Send function takes place, which will relay the IP address from each node, tossing away the data at each point until each entry and exit node are given to the respective client
5. Using the IP address of the entry and exit node, a peer connection is built.
6. Data transfer will occur using the WebRTC API, sending tracks of video and audio to the nodes.
7. Receive Function will take place which will act the same as the send function but relay it back to the respective client.

## Benefits & Concerns

The ultimate goal is to prevent IP leaks from occurring when using a WebRTC application, and with a Distributed Hash Table server representation this will be possible. Through a full implementation of DHT with a WebRTC application, it may become to realize that the upkeep and or computing power needed is not sufficient in transporting real time video and audio data. But, if kept lightweight enough while still remaining secure, I do believe that this sort of server presentation would be sufficient. Beyond a WebRTC application and through a previous project, I have seen the incredible power and transfer of node capability that a simple DHT server can implement.

## Sources


1. ExpressVPN. “How to use the WebRTC leak checker,” *ExpressVPN.* Accessed on: Dec. 4, 2020. [Online] Available: https://www.expressvpn.com/webrtc-leak-test

2. T. Fisher. “What is a Public IP Address?,” *Lifewire.* Accessed on Dec. 5, 2020 [Online] Available: https://www.lifewire.com/what-is-a-public-ip-address-2625974

3. J.M. Porup. “What is the Tor Browser? And how can it help protect your identity,” *CSO*. Accessed on: Dec. 10, 2020 [Online] Available: https://www.csoonline.com/article/3287653/what-is-the-tor-browser-how-it-works-and-how-it-can-help-you-protect-your-identity-online.html

4. F. Ali Khan. “Chord: Building a DHT in Golang,” *Medium.* Accessed on: Dec. 10, 2020 [Online] Available: https://medium.com/techlog/chord-building-a-dht-distributed-hash-table-in-golang-67c3ce17417b