WebRTC Covert Channels by Image Filtering

An Abstract of a

Thesis Presented to the

Department of Computer Science

Western Illinois University

In Partial Fulfillment

of the Requirements for the Degree

Master of Computer Science

By

Dennis McMeekan

May, 2021

ABSTRACT

"WebRTC is an open-source web-based application technology, which allows users to send real-time media without the need for installing plugins" [1]. This type of technology has become a main-stay in all industries, and more specifically WebRTC being open-source has allowed for researchers and developers across the world to discover new advancements and heights of real-time communication without prior installations or requirements. Although this has provided a great source of real-time communication, there are still concerns with this being a relatively new application. This study will look further into data integrity regarding these types of applications, along with briefly discussing the issue of solving IP leaks through a distributed hash table. To further develop this topic, a simple WebRTC application has been created with the purpose of sending a bit through a covert channel based on a delay in data being received from one peer to another, focusing on the bitrate, framerate, covert channel bandwidth and error rate. Furthermore, this issue will then be mitigated to ensure that individuals developing a WebRTC application can further prepare for defense against such attacks or misusage.

APPROVAL PAGE

This thesis by **Dennis McMeekan** is accepted in its present form by the Department of Computer Science of Western Illinois University as satisfying the thesis requirements for the degree Master of Computer Science.

_____
Chairperson, Examining Committee

_____
Member, Examining Committee

_____
Member, Examining Committee

_____
Date

WebRTC: Covert Channels by Image Filtering

A Thesis Presented to the

Department of Computer Science

Western Illinois University

In Partial Fulfillment

of the Requirements for the Degree

Master of Computer Science

By

Dennis McMeekan

May, 2021

ACKNOWLEDGMENTS

Grateful acknowledgment is extended to Dr. Binto George, thesis supervisor, and committee members Dr. Nilanjan Sen and Dr. Chunying Zhao for their valuable suggestions and guidance given in this thesis project.

TABLE OF CONTENTS

LIST OF TABLES

CHAPTER ONE

INTRODUCTION

In a cultural and technology environment that is continuously evolving, real-time communication is more essential than ever for individuals to complete daily tasks such as school, work, and personal communication. “Live chat has become the leading digital contact method for online customers, as a staggering 46% of customers prefer live chat compared to just 29% for email, and 16% for social media” [2]. With the reach of internet to all points of the world, it is realistic to see that real-time calls between two or more individuals will be the main product for online communication methods if it is not already there today. In recent years, WebRTC applications have been used to establish real-time communication between two or more peers. This product provides limitless advantages to web developers across the entire world, but with these advantages, there are also disadvantages. The main focus of this study is to determine the security factors that may be exploited with WebRTC and mitigating these issues.

In establishing an understanding of the vulnerabilities, a security study was conducted to look further into possible security flaws with WebRTC and real-life examples or discoveries of data integrity problems with these applications. Roughly in the past ten years, there have been a number of open-source examples that have been centered around WebRTC that focused on the altering and transmission of data in real-time. These were vital in providing a sense of direction on how to discover covert channels and initiating/preventing data transformation. The most common way to develop a WebRTC application is using the open-source API implemented by the creators Google, focused around the two main languages of JavaScript and HTML5.

The emphasis for the research and project portion was data integrity related to covert channels, along with a brief survey on IP leaks. Data integrity is an issue because of WebRTC's open-source model, clients hold the ability to alter the actual data that is being sent from one peer to another using the API provided. Specifically, the application can be altered to create covert channels via image filtering, a process that takes the video input, alters the data, and then outputs the video. This proves to be a problem, because having a secure data transfer from one individual to another is a vital process in application development. It was discovered that by implementing a delay in data transfer from one client to another, a bit can be sent and received based on this delay by the other client. This would provide the idea that data (messages) can be transferred secretly through clients without any control or knowledge by the administrator. There is a prevention method to prevent this from occurring, which is implementing random delays on the server or administrative side. This will increase the error rate due to the inability of each client to send and then sense the delay in data. These discoveries were mainly done using WebRTC's API and different test elements such as the bitrate, framerate, covert channel bandwidth in an effort to determine the error rate.

Beyond this, IP leaks is highly discussed as a security flaw with WebRTC applications, as in most cases the user's public IP address is used directly. This is commonly addressed with the use of a VPN (Virtual Private Network), but in almost all cases this can be a costly effort and can be limited in the data limitations of the user.

To prove and further research data integrity, two prototypes have been created. An unsecure version, which focuses on creating a covert channel, delaying/sensing data, and then receiving a bit based on the delay. And a secure version which combats this issue, implementing random delays to further increase the error rate at which the bit is being received.

CHAPTER TWO

BACKGROUND AND RELATED WORK

This chapter focuses on describing the thesis problem statement and looking into prior research that has been completed to work towards implementation and mitigation. This sets the groundwork for the thesis and finding next courses of action.

2.1 Problem Statement

Due to the demand of real-time communication, WebRTC has the foundation to allow for a secure and simple connection to be made by two users without installing native apps or plugins. With this being an open-source application, not all security protocols and features have been fully examined.

2.2 Security Concerns

Through discussion and examination by committee members two security features were chosen. The first one being data integrity concerns, which involves the altering or transmitting data without any knowledge to the administrator. The second being IP leaks in relation to a connection being established by two or more peers.

2.2.1 Data Integrity

Going beyond server and client authenticity, it is vital to ensure that the *communication stage*, the main segment that is being used throughout this thesis implementation, holds secure. When creating a WebRTC application, two or more peers will create a connection, and send real-time data between each other through the use of a server. The security concern in doing so with a WebRTC application comes mainly as a result of this being open-source project, and the API is available to the public. This obviously provides great opportunities for developers, but also for

hackers and exploits. Specifically, the video can be altered between each client, creating a vital security concern.

2.2.2 Covert Channels via Image Filtering

Expanding upon data integrity, WebRTC real-time image filtering can be implemented to exploit the vulnerability of data being altered from one client to another. This has been seen implemented using two different types of video input, *local* and *remote*. Separating these two types of videos simulates a network connection with WebRTC API in which one peer connects to a server, and then sends data to a receiving client. With image filtering, this instead involves taking the input (local) video and transmitting that through a canvas element. A canvas element is a type of HTML5 element that is intended for the use of drawing graphics or in this case, videos. After the video is transmitted into the canvas element, then that video is then sent to the receiving client. There is a very powerful example done previously that does this sort of action, but also applies a filter on top of the incoming video to alter the appearance [3].

Through this examination, it appears that action can be taken similar to having a filter on top of the data, but instead delaying the video being sent from the canvas element to the output (remote) client. This basis of implementing a delay using image filtering, allows the creation of a covert channel. A covert channel allows for individuals to send secret or sensitive information without the administrator knowing. In this example, this involves not only one client first implementing a delay to send data, but also the receiving client sensing that delay to receive the sent data. This also to the network/server administrator would only appear to be noise, as the data rates tend to fluctuate very briefly, and then return to normal.

2.2.3 IP Leaks

A large issue with WebRTC applications, is IP leak concerns. This is due to the fact that a Peer-to-Peer connection requires that each client has each other's communication address. A similar issue was discussed and implemented with the creation of the TOR browser [4], but instead of using a simple WebSocket or HTTPS server, a Distributed Hash Table server was implemented that would allow for each client to still communicate directly with each other but remain anonymous. This provides the idea that in a similar fashion, this type of server communication can be implemented with a WebRTC application. Although this ideally would be implemented and have a prototype as is done with covert channel implementation, time did not permit during this thesis. Instead, later explained is a research document further describing this possibility and possible future courses of action.

CHAPTER THREE

RESEARCH METHODOLOGY & IMPLEMENTATION

This chapter describes the research steps taken to establish an understanding of how to approach and implement the subject of covert channels, as well as describing the steps needed in creating a distributed hash table server with a WebRTC application. With the extensive research and examination of open-source examples, a prototype implementation involving an unsecure and secure version have been created.

3.1 Open-Source Project Examples

Expanding upon the previous chapter that discussed the main two issues at hand, there are a number of examples that help provide a starting point in establishing covert channels and then preventing them.

3.1.1 WebRTC API

The center of the programming revolves around holding a deep understanding of WebRTC API and concepts involved. The beginning of this research begins with a great publicly available API webpage that lays out each function, method, interfaces, and references available in relation to WebRTC [5]. This allowed a basis understanding to be established and continued as a reference for attacking specific angles in prototype implementation. Further than API, actual examples also allowed for a great resource in provide a base to begin with in testing and developing a simple, but complex, WebRTC application. Specifically, an “as simple as it gets” application was found that built a client and server connection, and then allowed the media to then be transferred through a WebRTC connection using the corresponding protocols [6]. Examining these two pieces of research helped tie together the API provided and actual

implementation of creating a remote and local connection, capturing a stream, and then altering that data stream.

3.1.2 Image Filtering

Image filtering is the tool that connects WebRTC and Covert Channels. This first came to discussion to the ability of discussing the idea of altering data in a WebRTC application. The canvas HTML5 element is a common element used throughout web development, and this brought the idea that by transmitting a video element, into a canvas element, that the data would then be altered before being sent. An example that provided a great basis to this implementation took all aspects explained earlier, a local video, canvas element, and an output/remote video but instead of manipulating the pixels, the example allows for filters to be draw on top of the video feed that is being transmitted [7]. With these two great examples, the actual implementation is able to take place through an unsecure prototype, and a secure prototype.

3.2 Unsecure Prototype

The creation of an unsecure prototype is solely emphasized on creating covert channels while establishing a WebRTC connection between one peer to another. To begin, a simple application was created which built the connection using WebRTC API, a great example was found which made this as dead-simple as possible and provided a base to build upon [7]. The application was altered to allow for the connection and remote connection to be simulated and displayed on the same webpage. This meant altering the way that the clients interacted with each other, and instead of waiting on the server to send a signal once a connection is made, a video element is used which conquers a similar action by adding a “Get Media”, “Start Call”, and “Hangup” button through a series of WebRTC connection functions and methods. This allows

the ability to have a local and remote connection appear on the same webpage, instead of having needed two completely separate clients to alleviate testing and trail purposes. There were two main pieces of code that required being altered, one being HTML5, and one being JavaScript. In terms of HTML5 (index.html), this was just formatting and allowing the objects and video elements in relation to the corresponding JavaScript. In terms of JavaScript (webrtc.js), this was used to implement the change in functioning of the WebRTC application specifically to implement and mitigate covert channels. Once the connection was altered to simulate local and remote in one webpage, image filtering needed to occur which would allow for covert channel implementation to be possible. This involved adding an extra step in between the local connection and the remote connection, using a canvas element. This expanded upon the example looked at, and instead of applying a filter to the data being transmitted, a delay instead occurs. This utilizes a drawToCanvas function which will relay the data from the local video to the canvas element, which then allows the remote connection to call the data being transmitted from the canvas. Once this was implemented, it was simple possible to delay the data based on user input (one or zero) by adding this into the drawToCanvas function. The delay would then occur if an input bit were one, and a very small delay would occur if the input was zero. There was a slight anomaly that allowed this to happen, because when adding in a very small delay, this manipulated the bitrate to increase drastically for a very short period allowing the bit to be sensed on the receiving end. This is believed to be due to data compression techniques in relation to data transformation with media elements in HTML5 and JavaScript.

3.3 Secure Prototype

After successful implementation of an unsecure prototype, the application needs to be altered to mitigate the low error rates of transmitting of bits through covert channels. There are

two routes in which this approach can be taken. One, with a constant delay, and two, with a random delay. These two methods simulate an implementation by the administrative side through the server. This involves adding in an interval section that will add in a delay, with intentions of increasing the error rate. With a constant delay, this may appear to work somewhat, but it would be very common that the sender and receiver could see the constant delay, and then alter the sending and receive methods accordingly. With a random delay, this is the best approach to take when mitigating covert channels via image filtering, due to the inability of the exploiter to sense the random delay each time.

3.4 Distributed Hash Table

Outside of prototype implementation, it is vital to address IP leaks and discuss the possibility of implementing a distributed hash table. “Any two devices talking to each other directly via WebRTC, however, need to know each other’s real IP addresses” [8]. 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. Instead, it is possible to shift focus to implementing a similar server side as seen with the Tor browser. The current approach of the server-side implementation of a WebRTC application involves sending data through a series of *STUN/TURN* protocols using web sockets.

A new approach would be taking similar action to the Tor browser’s implementation which has been made famous due to its ability for users to remain completely anonymous.

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" [9]

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 [10]. 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) [10]. 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.

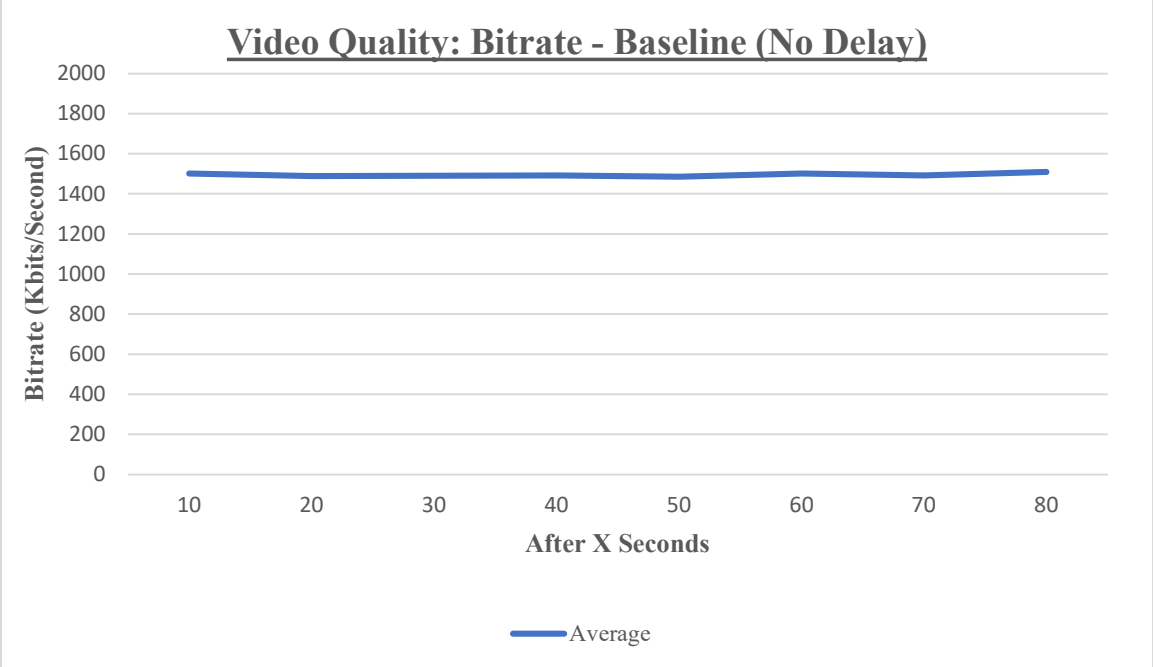
CHAPTER FOUR

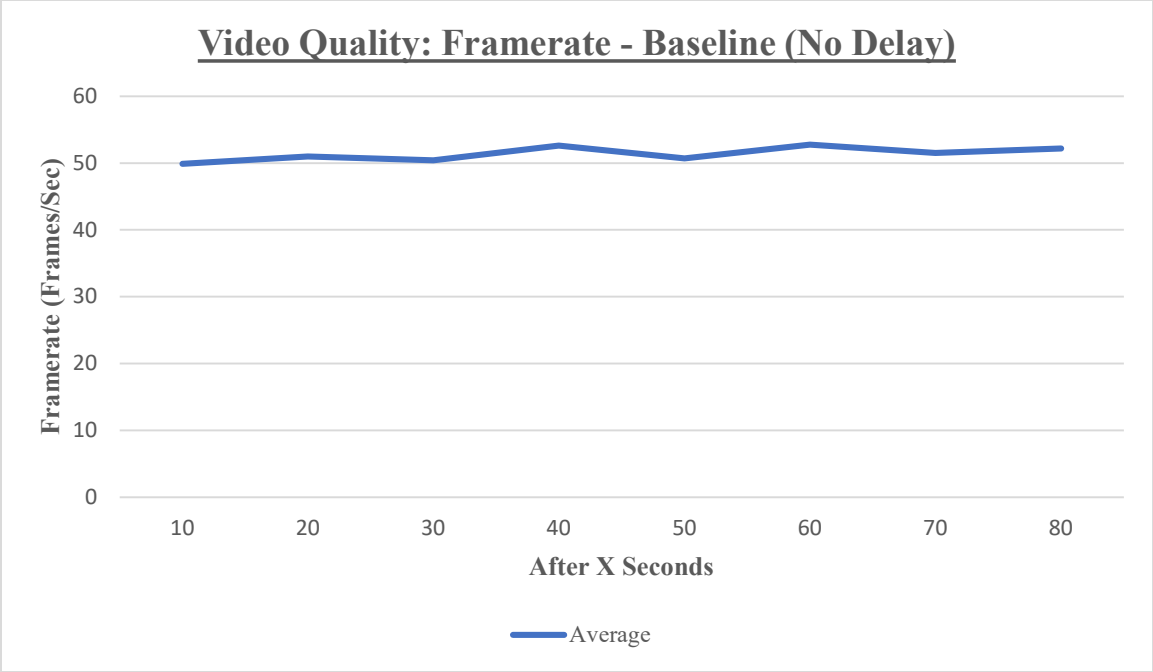
FINDINGS & ANALYSIS OF DATA

This chapter discusses the findings that were discovered in the implementation of an unsecure prototype, establishing covert channels, and a secure prototype, mitigating this with a series of delays.

4.1 Covert Channels

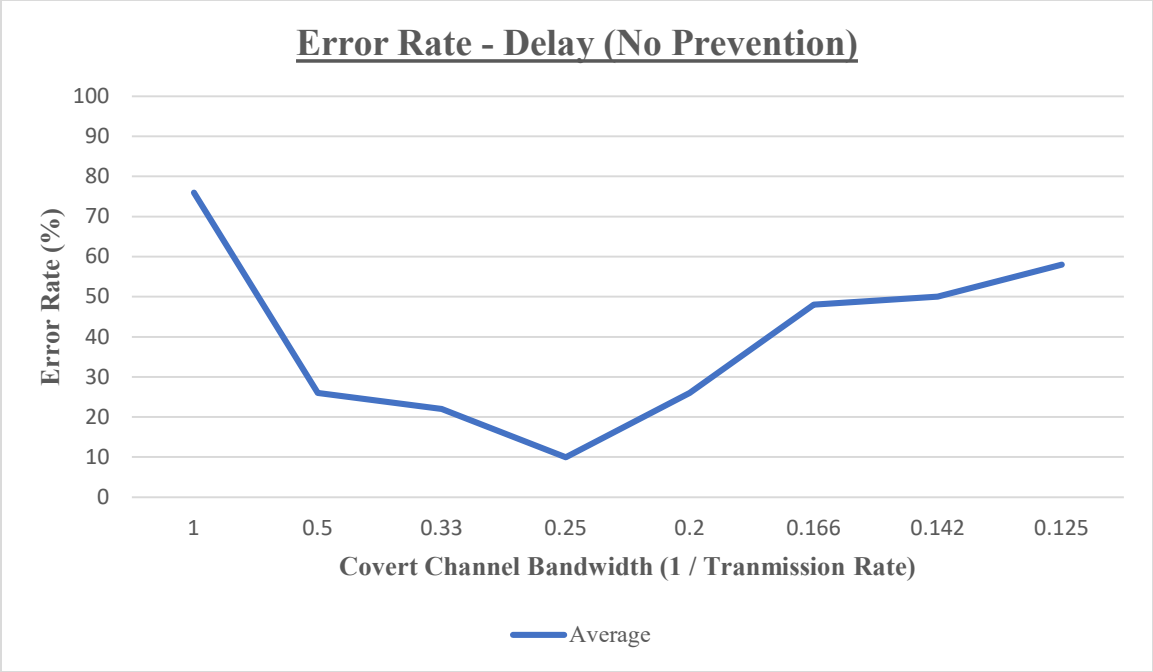
Expanding upon an unsecure prototype implementation, proper testing needs to occur in which the effectiveness can be determined of this method. There are two main issues which can occur from implementing delays in a real-time media application, video quality and the error rate at which the bit is being sensed. To test this, baseline statistics need to be measured in which no delay or prevention method have occurred. For baseline measurements, ten different tests were taken for each chart, five using a *black screen* as the media source, and five using a *sample presentation.*

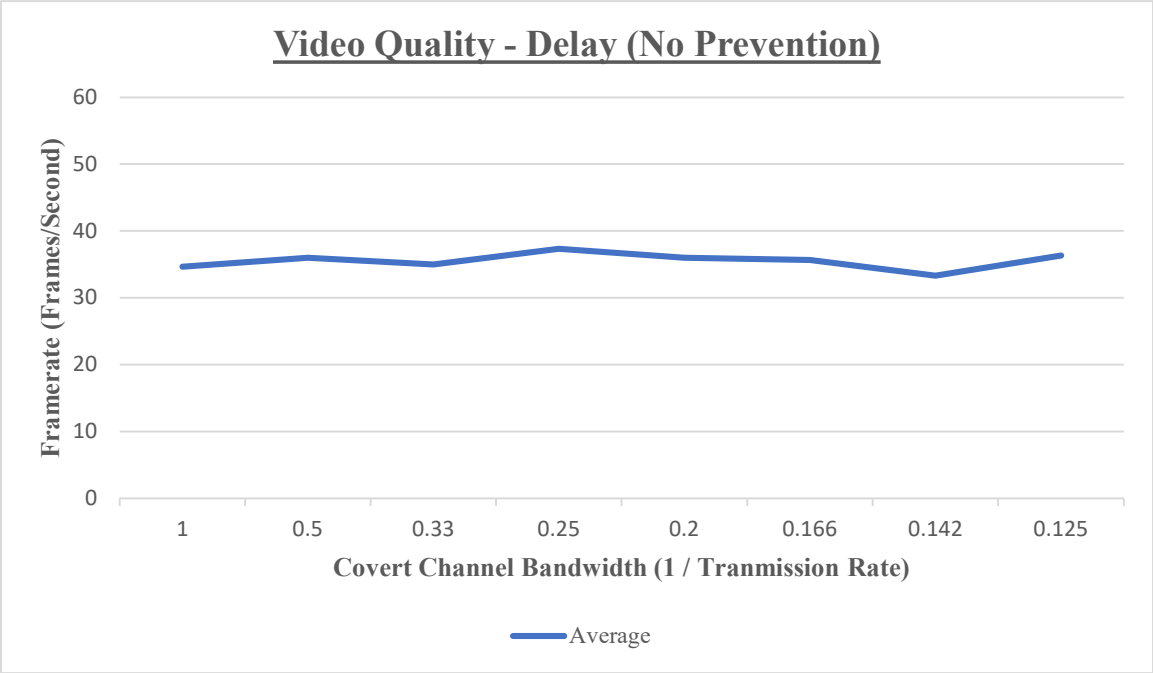




As seen with the charts above, the baseline measurements for this WebRTC application without any delay or mitigation methods involved is the bitrate sitting around 1500 kbits/second, and the framerate sitting around 50 frames/second. This provides a baseline measurement to judge performance statistics in comparison to the delay method implemented along with the mitigation/prevention method. For all testing purposes for these tests and later were completed using the same network, and equipment.

Next, testing needs to occur involving using the delay method which allows covert channels to occur and a bit to be sent and received from one client to another. Each test involved measuring two separate elements, the bitrate and framerate based on the covert channel bandwidth. Covert channel bandwidth is a statistic that takes the number of bits being sent divided by the time at which the input is being calculated. (1 / inputRate or T).



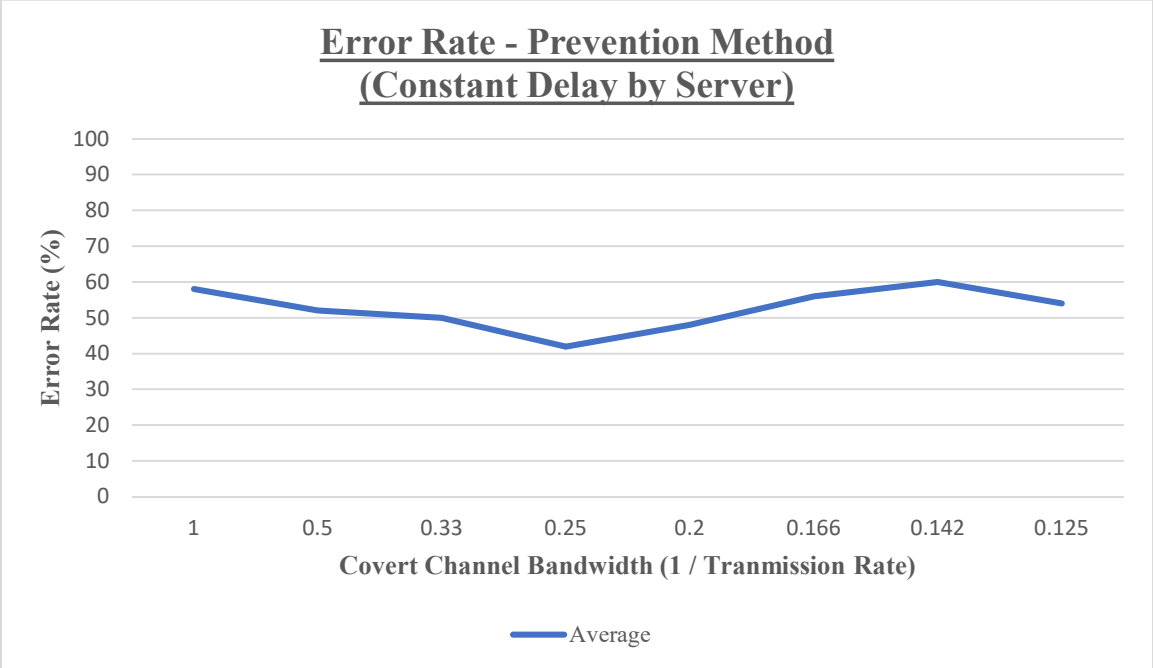


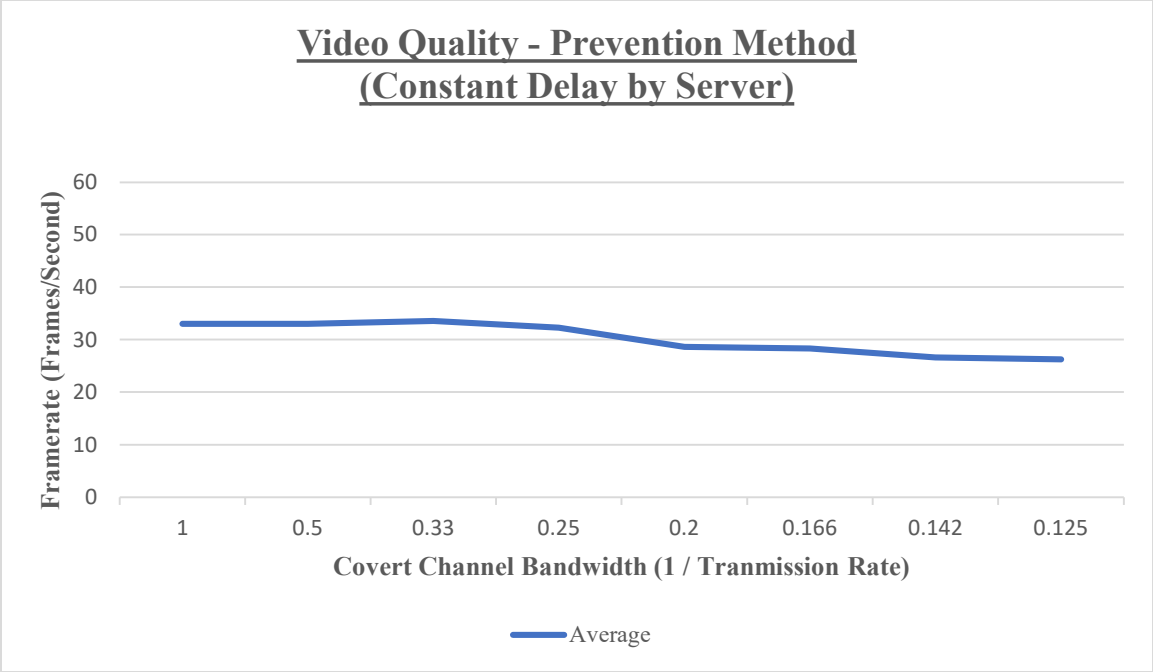
The error rate appears to flucuate greatly depending upon the amount of time between each bit being sent (covert channel bandwidth). This is because after a small amount of time, the bitrate stabilizes more to the averages we see around 1500, which means the error rate would increase. It seems that roughly two to four seconds is the ideal in being able to send, and then

receive the bit on the remote connection. When looking at the video quality, it seems that the average is roughly 15 frames less per second in comparison to the baseline. Although this is drastic in comparison, this is a small price to pay when implementing covert channels with a WebRTC Application.

4.2 Prevention Method

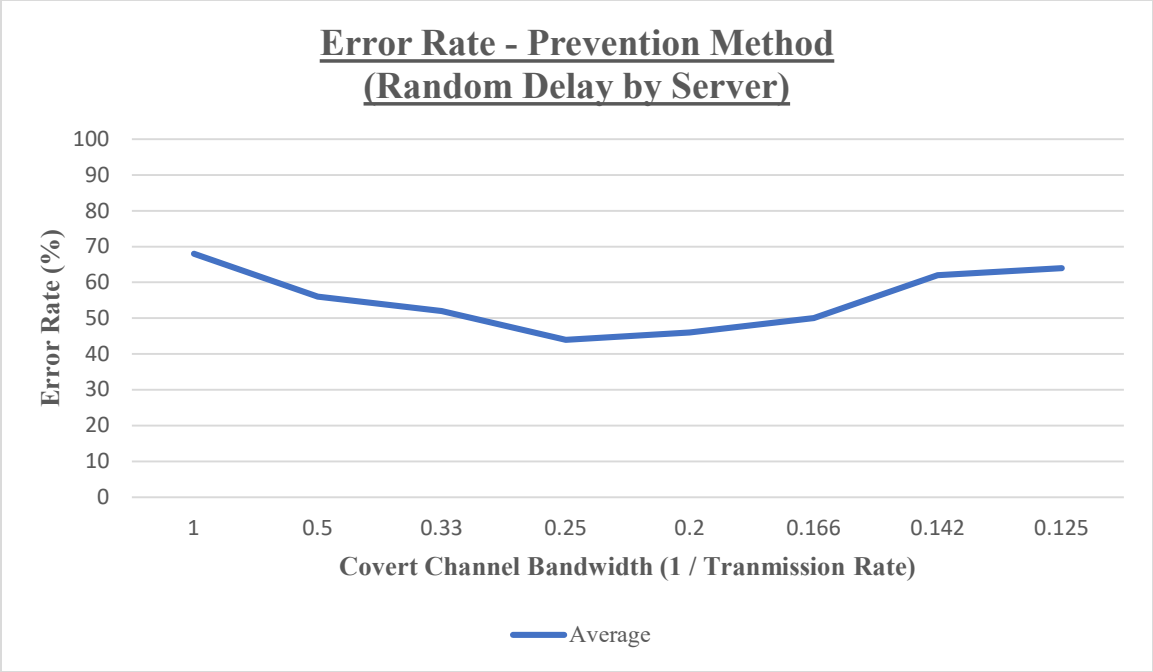
Expanding upon a secure prototype implementation, proper testing needs to occur. This will be compared to the previous results seen with baseline measurements as well as the delay method. The same number and variables are going to be used in extension of secure testing, and the results are as followed. First, the error rate and video quality will be examined using a constant delay.

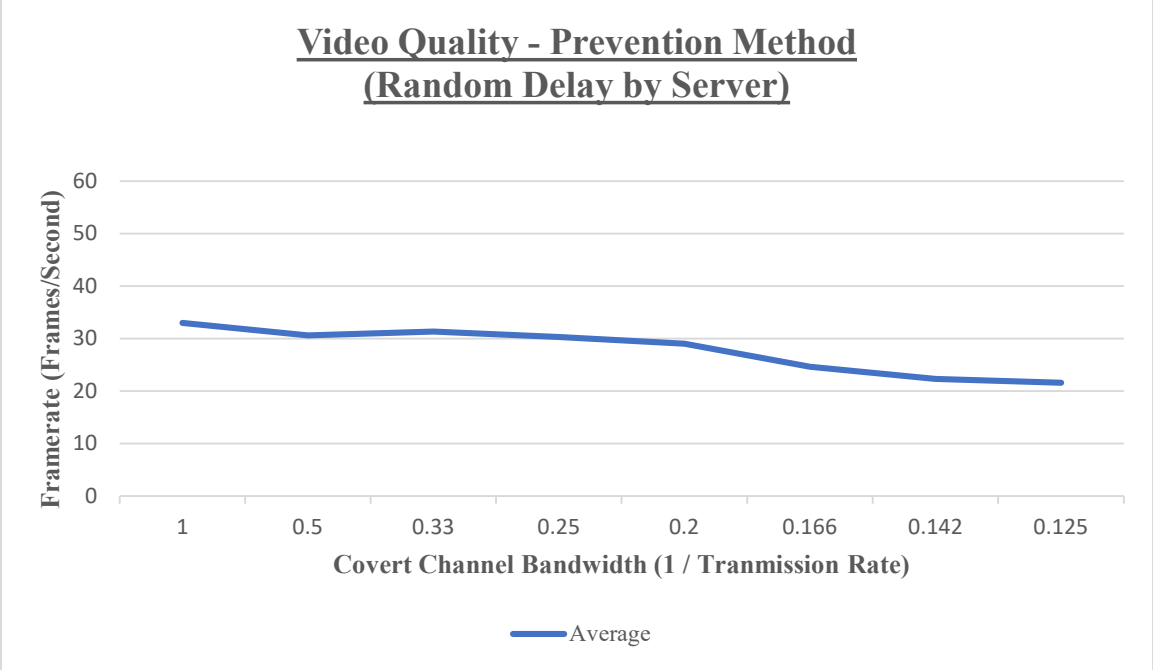




Although the error rate is not 100%, it is still an increase in comparison to the error rate from the previous chart. Many of these statistics and variables used throughout implementation can vary greatly depending on the network and equipment being used. In terms of framerate, it is seen that video quality is a little bit worse in comparison to the delay method, but not enough to allow for added data integrity.

Next, the same tests have been conducted using the same variables and measures but instead with a random delay





These results are quite similar to the delay implementation as a constant, but in terms of error rate this seems to fluctuate a tad bit more than as a constant. Although this may appear to be very similar in performance in comparison to a constant delay, this would definitely be the

ideal choice due to the fact that when using a covert channel, the client would not be able to determine the delay and alter the code specifically to counter the delay.

CHAPTER FIVE

SUMMARY AND CONCLUSIONS

Through the development of this thesis, security protocols surrounding a WebRTC application have been examined in terms of data integrity and IP leaks. Covert channel implementation allowed for a bit to be sent and received by a peer-to-peer connection without any knowledge to the administrative side. This seemed to provide some decrease in performance, but overall, nothing too over the top. This was then mitigated through the result of constant and random delays simulating a server-side implementation. This further resulted in performance concerns, but greatly increased the error rate at which a bit can be sent and received through convert channel implementation. It is important to notice that results may vary greatly considering each testing procedure was done with a series of random bits being sent each time. Also, with data transfer, it is very common to fluctuate the ability at which a network is able to transfer the media being sent and received. It also is vital to acknowledge that since we are only sensing two different data rates, that ideally using a middle ground to reset the bit received would greatly decrease the error rate for the delay method, while increasing the error rate involved with the mitigation method. This is due to the fact that the value being sensed would be "reset" at a much higher rate, which would prove to be an advantage for both sides of the prototype implementation. It is also noticed that using the sample presentation testing method allowed for the data to remain more consistent, due to not using such a high consistent level of data compression.

5.1 Future Work

In terms of future work, there are quite a bit of possibilities on the direction of which this project could continue. First, it is vital to address network concerns. This is the thought that instead of implementing a WebRTC application with covert channel implementation through one webpage, this be done using separate networks and with completely separate clients. This may raise quite a few concerns and issues, but simply tweaking variables and measures would be a possible outlook. Next, ideally, an audio delay would need to match up with the video delay. Currently, only the image and video being displayed is altered, while matching up the audio would mask the data transformation even more. Finally, it would be incredible to see an actual prototype implementation of the distributed hash table server implementation with a WebRTC application that was examined previously. This could provide a drastic change in how real-time communication applications are implemented, and would be ground breaking to say the least.

References

[1] NTT Communications (2015). *A Study of WebRTC Security*. Retrieved August 20, 2020 (https://webrtc-security.github.io)

[2] MacDonald, Steven (2021). *25 Reasons Live Chat Can Help You Grow Your Business in 2021*. Retrieved February 3, 2021 (https://www.superoffice.com/blog/live-chat-statistics)

[3] Mozilla (2021). *Pixel Manipulation with Canvas.* Retrieved February 12, 2021 (https://developer.mozilla.org/en-US/docs/Web/API/Canvas_API/Tutorial/Pixel_manipulation_with_canvas)

[4] Anon, Dennis (2018). *Everything you wanted to know about Tor but were afraid to ask.* Retrieved October 10, 2020 (https://privacy.net/what-is-tor/)

[5] Mozilla (2021). *WebRTC API.* Retrieved November 1, 2020 (https://developer.mozilla.org/en-US/docs/Web/API/WebRTC_API)

[6] Tully, Shane (2018). *GitHub-Shanet-WebRTC-Example.* Retrieved October 1, 2020 (https://github.com/shanet/WebRTC-Example)

[7] Donaldson, Scott (2018). *Using WebRTC for Real-Time Image Filtering.* Retrieved November 10, 2020 (https://sudo.isl.co/webrtc-real-time-image-filtering/)

[8] ExpressVPN (2021). *How to use the WebRTC leak checker.* Retrieved December 4, 2020. (https://www.expressvpn.com/webrtc-leak-test)

[9] Porup, J.M. (2019) *What is the Tor Browser? And how can it help protect your identity.* Retrieved: Dec. 10, 2020 (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)

[10] Khan, Farhan Ali (2018) *Chord: Building a DHT in Golang* Retrieved December 10, 2020 (https://medium.com/techlog/chord-building-a-dht-distributed-hash-table-in-golang-67c3ce17417b)

[11]

Appendices:

Communication Stage

There are two separate stages involved with WebRTC applications, a communication stage, and a signaling stage. The communication strictly focuses on how the two clients interact with each other, while the signaling stage focuses on how the server side is implemented. This project's almost entire focus is on the communication stage.

Local

Local means that when looking at a connection being made between two clients, the local connection is the initial client to be sending a connection request, and in doing so, is the client which will send a bit.

Remote

Remote means that when looking at a connection being made between two clients, the remote connection is the former client in which establishes a connection based on a previous request. In this implementation, this is simulated through WebRTC API and this client will be sensing and receiving the bit based on the local connection.

STUN/TURN

STUN/TURN are protocols that are extremely common with WebRTC applications. These are simply called using WebRTC API and most famously created by Google.

Black Screen

This is a type of testing that occurs by using a webcam and covering the camera, to allow for a media element to still be sent but to find hold similar results throughout various tests.

Sample Presentation

This is a type of testing that involves real-life examples of connection testing, and involved a webcam being on and an individual (myself) to be “giving a presentation” to simulate a real-time example of a WebRTC application.

WebRTC Application Implementation

Index.html (HTML5)

adfgadf