Securely modifying adaptive bitrate streaming to hide information

Livia Earl
Mentored by Mr. William Newton

Introduction

Steganography, the process of hiding information in plain sight, through adaptive bitrate streaming is relatively undiscovered. While many other techniques have been perfected, steganography in streamed video is hard to come across. Adaptive bitrate streaming (ABR) is a technique of streaming over HTTP which encodes the content at multiple bitrates. This allows the client, any device that requests access to a server, to choose which bandwidth cooperates best with their network quality. HTTP Live Streaming (HLS), a streaming protocol created by Apple to play live audio and video with ABR that consists of .m3u8 and .ts file formats, was chosen due to its wide range of use (Fecheyr-Lippens, 2010). Since HLS is commonly used, the steganographic method has many opportunities to be applied in everyday transactions such as efficient and undetectable data security.

The purpose of this project was to create a new steganographic method to securely hide information by modifying the adaptive bitrate streaming of a video file. It employed both steganography and cryptology, the process of encoding information, to secure data.

Materials & Methods

GitHub and Wireshark were used, in addition to Python, to develop this new steganographic technique. GitHub functions as a repository for git projects, enabling multiple engineers to collaborate on a code base while maintaining the history of changes. Whereas Wireshark allows the engineer to inspect each data packet being sent across the network after the program has run.

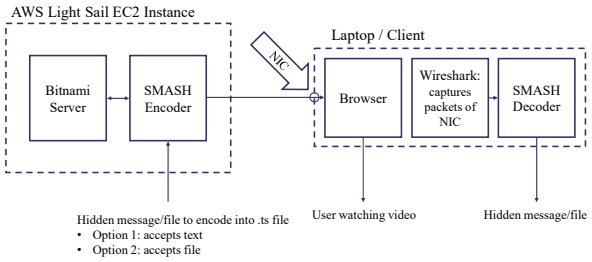


Figure 1: This diagram explains how the communication between the server and client was sent. The network interface connection (NIC), a hardware component installed on a computer to connect to the network, is marked by a circle where Wireshark captures the packets to examine. The message was input into the encoder and outputted by the decoder in its original form.

Materials & Methods (continued)

This project was organized into five sprints. These included research, ensuring communication between the server and client, framework code building, technique development, and validation.

Research time was used to build an understanding of tools, including Python, GitHub, and PyCharm. Python is packaged with many built-in libraries from the community. It was the best approach for this project since it supports rapid development, scripting, and object-oriented programming. GitHub utilizes commands to share changes to the program with the cloud and other engineers. Lastly, PyCharm is an integrated development environment that communicates with GitHub directly, so uploading and saving code is efficient and simple.

The second sprint started by creating a web server and GitHub repository. Figure 1 shows the server, an AWS Light Sail EC2 Instance, which allows access to many files as well as the ability to connect to the two programs. Additionally, framework code was developed to request certain parameters from the user such as the hidden message and the video stream. An .mp4 to HLS, as well as the master playlist that consists of both the high- and low-resolution streams, was created.

The third and fourth sprints programmed the technique to conceal the data (Figure 2). Afterward, the steganographic stream file was replayed through Wireshark as a video to test that the file was not corrupted.

The fifth sprint involved debugging and testing. Once the code worked with both text and files embedded, the packets were examined in Wireshark and decoded in PyCharm to ensure the message was fully received.

```
ind = temp.find(b'\xff\xff\xff\xff\xff\xff\xff\xff\xff')
if ind >= 0: # checking 9 values do exist in packet
    endIn = ind
    while temp[endIn] == 255 and endIn < len(temp) - 1:
        endIn += 1
    strt = ind + 4
    endIn = endIn - 4
    numOfBytesToHide = endIn - ind + 1 # bytes hidden
    if lc < len(message): # checking length of message
        ntemp = temp[:strt] + message[lc:lc +
      numOfBytesToHide] + temp[endIn:]
        lc += numOfBytesToHide
        packet["Raw"].load = ntemp
```

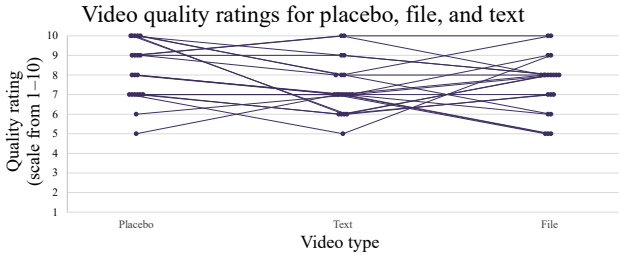
Figure 2: The data was hidden within the padding, expressed as hexadecimal value '0xFF', of each network packet. The padding was replaced with the hidden message after it was encoded using DES encryption, found in a Python library.

Results

In this study, 20 subjects were given three videos to watch and evaluate. Each of these had the same video content. The only change to each video was either no hidden data (placebo), hidden text, or a hidden file. The three videos were shown in random order to each subject. They were then asked to rate the video quality on a scale from 1–10, 10 being the highest quality (no buffering or interruptions and clear visuals).

Results (continued)

The quality ratings by each person for the videos were graphed in a scatterplot (Graph 1). To account for a Likert scale, a Friedman test was used to compare the overall ratings. The results revealed no significant difference between the ratings of the three videos, regardless of the steganographic technique's presence.



Graph 1: The Friedman test showed that there was not a significant difference in the quality of the placebo (*Mdn* = 8.0), hidden text (*Mdn* = 7.0), and hidden file (*Mdn* = 7.0) videos, $\chi^2(2) = 3.80$, $p = .150$. Using the alpha level of 0.05 the null hypotheses cannot be rejected, indicating there is not a statistically significant difference in the quality of the videos regardless of the steganography.

Conclusion

The goal of this project, to develop an effective steganographic technique to hide information within a video stream using adaptive bitrate streaming was achieved. The results of the video quality survey revealed no significant difference between the videos, meaning the steganographic technique caused no detectable changes by the human eye within the stream. This implies that the quality of the stream was not noticeably worsened by the hiding of data. While this technique was successful, more development could be done to apply this same technique to a live video stream instead of after the stream concluded.

In further research, this technique could be implemented to more advanced adaptive bitrate streaming that includes additional bitrates, so it is available for multiple data sizes. Additionally, this project could be used as a baseline for other projects involving steganography and cryptology in live streamed data.

References

Fecheyr-Lippens, A. (2010). *A Review of HTTP Live Streaming*. ISUU, Inc. https://issuu.com/andruby/docs/http_live_streaming