Capstone Project 36

Virtual Reality Livestream

Design Document

Group 36:

Kaleb Graham

Alex Macdonald

Atif Murtaza Mahmud

Yingmin Tu

Andrey Varlamov

Table Of Contents

Changelog

Date	Author	Location	Change
2020-02-09	KG	References/Glossary	Move References section to Glossary.
2020-02-09	KG	Design Overview	Add additional description to the design overview and add a networking component.
2020-02-09	KG	Video Server	Expanded the description of the potential solutions and added evaluation criteria
2020-02-09	KG	Client-Side Application Platform	Renamed from Video-Viewing Platform. Added additional information to the Background section, added evaluation criteria, and split design considerations into a Considered Designs and Design Comparison section. Finished the Design Justification section.
2020-02-09	KG	Encoding Design Decision	Large refactor of this section. Removed the Encoding/No-Encoding section. Added more info to the hardware vs software encoding section. Added the Codec design decision
2020-02-09	KG	References	Reformatted the references. Removed unused references.
2020-02-09	AMM	Introduction	Rewrote Introduction and Scope based on feedback received in M2
2020-02-09	AMM	Design Overview	Added details on specific components; modified high-level overview diagram
2020-02-09	AMM	Conclusion	Made changes to conclusion
2020-04-07	AV	Client-Side Application Platform	Updated the section to match the finished state of the project
2020-04-08	KG	3.1 Technical Limitations	Add this section
2020-04-08	KG	3.5. Application-Level Streaming Protocol	Add this section
2020-04-08	KG	3.3 Codec decision	Update with testing results and confirm design decision

1. Introduction

1.1 Purpose

This document provides details on our design decisions and implementation strategies for the *Virtual Reality Livestream* project.

As discussed in the Requirements document, the goal of this project is to enable multi-user support for Virtual Reality (VR) applications developed by our client, EML. In this document we will introduce and describe the features, limitations, and design trade-offs of every key component of our proposed solution.

This will provide the reader with further context on the architecture and building blocks of our project, and will serve as a starting point for anyone who wishes to improve upon the product in the coming years.

1.2 Scope

The design document is intended to give an overview of the three different components that make up our final product. Details of implementation like the code and specific configurations will be omitted from this document for sake of brevity. This document is written assuming the reader possesses the general technical knowledge in software and hardware that is expected from final year ECE students. As such, certain technical terms won't be explained in detail.

2. Design Overview

The design can be broken down into three major components.

1. **Instructor-Side:** The solid left rectangle in Figure 1 below shows the first component of the design, which is considered to be the server-side of the solution. This side of the design will consist of the instructor's laptop, which will be provided by EML and will run the virtual reality demo. The instructor will interact with this application in VR as is already possible, and will have 6 DOF. The added functionality is the ability to export the VR environment that the instructor is experiencing so that it can be shared with the students. This functionality will be built into a Unity plugin (**C1**), which will be integrated into the Unity project that is running on the computer in Figure 1. Encapsulating the functionality into a Unity plugin will make it easy to integrate with EML's existing and future projects.

2. **Client-Side**: The solid rectangle that can be seen on the right-hand side of Figure 1, is the second component in the design and is considered to be the client-side of the solution. This component will be used by students and will run on a separate device from

the instructor-side, with one device for each student. It will receive and display the exported VR environment. Students viewing this VR environment will only have 3DOF.

3. **Networking-Component:** The two previous components will communicate via networking protocols running over UBC's existing network infrastructure. The instructor's laptop and the students' devices will need to be connected to the same access network for this solution to work. This network is considered the third component of the solution. The 360 degree video extracted from the instructor's VR application by our plugin will be served over the network from the instructor's device directly to the student's devices. The system will be designed to be compatible with both wireless and wired networks.

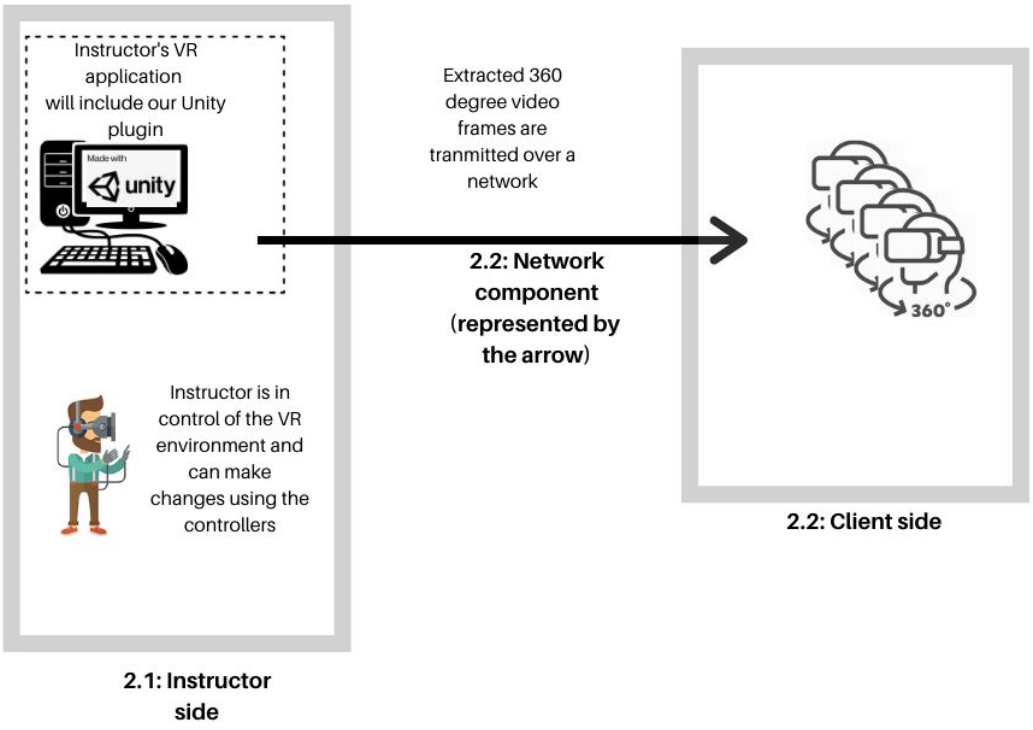

Figure 1: Design Overview as described in Section 2

3. Design Rationale

3.1. Technical Limitations

Before discussing the design decisions in detail, it is important to note some of the technical limitations imposed on this project. The three important limitations to consider are the bandwidth of the network being used, the processing power of the instructor computer, and the processing power of student devices.

As can be referenced in the Requirements document, the network bandwidth is being constrained (**C2)** to the determined theoretical bandwidth of the UBC network, and is 72Mbps. This will limit the amount of data that we can send from the instructor-side to the client side.

C4 outlines the minimum specifications of the instructor computer. This will impact quite a few of the design decisions in this document. However, while the minimum hardware was specified, processing power is difficult to quantify in a useful way. As such, some of our design decisions were based off of certain assumptions which are outlined in their respective sections.

There was no constraint or non-functional requirement relating to the processing power of a student-side device. However, **C3** does specify that the student-side must run on Android. The only decision affected by the student-side processing power is the video-codec decision in section 3.3.3. Unfortunately, due to the Covid-19 crisis, a lack of a sufficiently powerful test server prevented meaningful data being collected, as the maximum fps of a student-device is tied directly to fps of the server. Please see this section for more detail.

3.2. Data Transmission

3.2.1. Background

A key component of the project is the ability to extract data from the VR application and transmit it to the secondary users, who in this case are the students viewing the VR world in 3DOF.

3.2.2. Design Considerations

There were two designs considered for this component:

1) **Extract and send 360 degree video**: This method involves developing a unity plugin that would be able to extract 360 degree video and send the video frames to the client application (see figure 2).

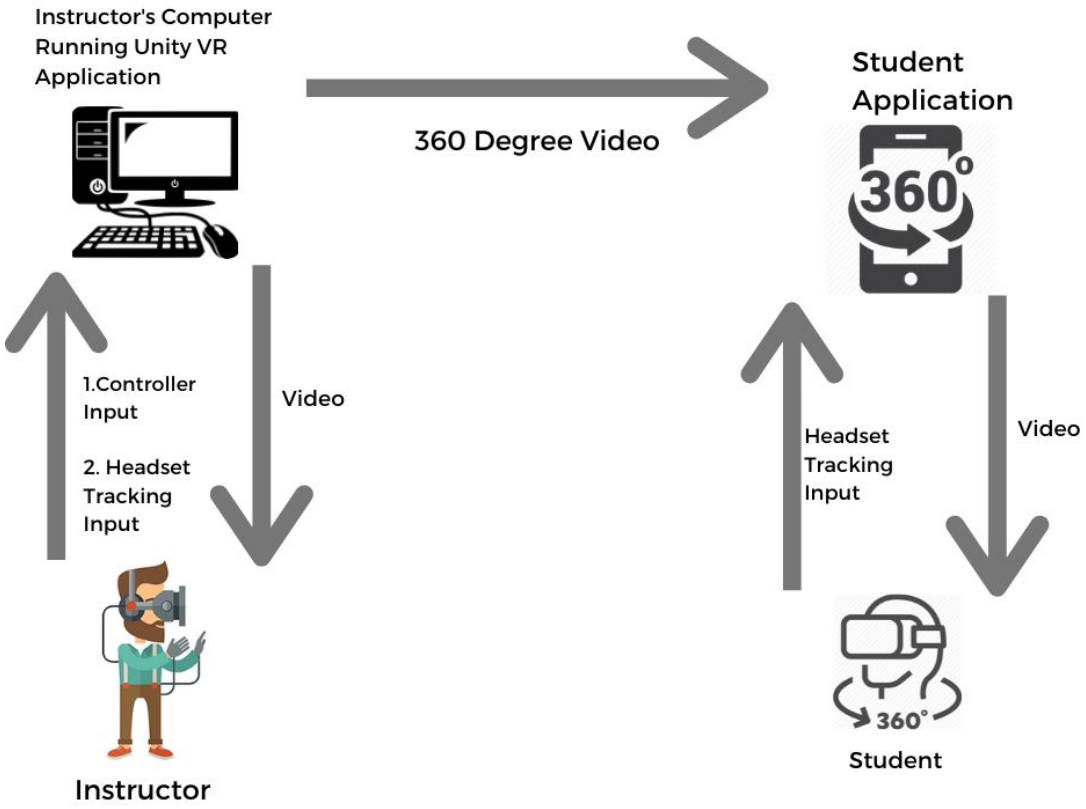

Figure 2: Design Overview for Sending 360 Video

2) **Clone VR world and transmit controller input from instructor:** This method involves running a mobile build of the instructor's VR application on the client device, and mirroring any changes to the environment made by the instructor by transmitting the inputs from their controllers to the mobile build via a unity plugin (see figure 3).

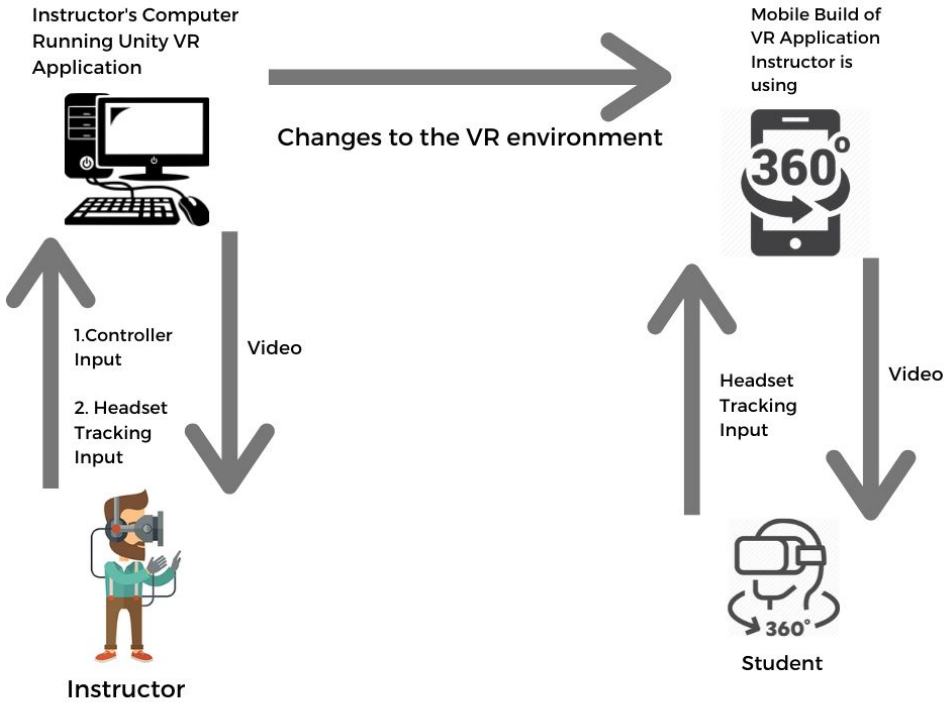

Figure 3: Design Overview for Synchronizing VR Worlds

Alternatives	Pros	Cons
1. **Send 360 degree video**	- Can create universal plugin that can be integrated into as many projects as needed	- Sending video frames would consume a lot of bandwidth
2. **Synchronize VR World**	- Only send instructor input signals, therefore consumes less bandwidth	- Client side devices may need to run very demanding applications - Need to create as many client side applications as existing unity projects. **Not a universal solution**

Table 1. Livestream vs World Synchronization

3.2.3. Design Justification

We are choosing to send a 360 degree video instead of synchronizing the VR environment because it would be more feasible for us to send raw video data given the non-functional requirements of minimum resolution (1080p) and maximum allowable latency (5 seconds) as opposed to creating multiple client side applications for the existing EML projects. Streaming

360 video is considered feasible, and it should be possible to stream 360 video to 3 or more clients (**NFR4**) on the UBC network, or any other reasonably respectable network. Given 72Mbps of bandwidth in total, this equates to 24Mbps for each of the 3 clients. The bandwidth utilized by an HD video stream varies, but Netflix suggests at least 5Mbps if streaming from Netflix [9]. This is much less than the 24 Mbps that would be available for each student. In fact at a bitrate of 5Mbps, a simple calculation of 72Mbps / 5 Mbps shows that up to 14 clients would be able to connect at one time given this average bitrate and network bandwidth.

The reason synchronizing two worlds is not a universal solution is that project specific code needs to be implemented. For example, EML has a range of projects from VR orchestra to VR human skull analysis. For this architecture, the EML developers would need to create a client side mobile application in parallel to the server side for each project.

Another problem is dealing with packet loss when synchronizing movement, since these can cause drastic differences in gameplay (ex, the difference between pressing one button or different button).

Furthermore, sending 360 degree video will allow EML to use our completed product in the future without modification, whereas the second alternative would require EML developers to build separate client side applications for any future project, consuming valuable development time.

3.3. Video Encoding

3.3.1. Background

An important implication of choosing a live-streaming approach is that it requires the ability to extract or create a video from a running Unity project. Unfortunately, Unity does not currently expose a method to directly grab a 360 degree video from a project as it is running; instead, it exposes a method to grab 360 degree images. Therefore, the task of assembling 360 degree images into a video must be handled by the Unity plugin being created.

The individual image frames are too large to be sent over the network, and hence the frames must be encoded. Video encoding is a process where individual image frames are compressed into packets. These packets are much smaller in size and therefore are able to be transmitted faster over a network. To demonstrate why encoding is necessary, consider the fact that a 1080p video frame is 1920x1080 pixels, for a total of 2073600 pixels. If each pixel is 2 bytes large, not unreasonable in practice, a single unencoded video-frame will be ~31Mb large. Multiply this by 30 frames a second, and one client would require ~950Mbps of bandwidth from the network. Note that constraint **C2** constrains us to a network that has only 72 Mbps of available bandwidth.

Two important design decisions arise from the necessity of encoding the video. The first design decision is whether to use hardware or software encoding, while the second design decision is what video codec to use. Further background information for these decisions are given in their respective sections.

3.3.2. Hardware vs Software Encoding

3.3.2.1. Additional Background Information

There are two ways to encode video - hardware encoding and software encoding [1]. As such, a choice had to be made on which form of encoding is more suitable for this particular use case. For software encoding, the encoding would run on the instructor's computer in parallel to the Unity project. In contrast, hardware decoding would involve a separate piece of hardware that would be connected directly to an instructor's computer through a cable (generally HDMI). This piece of hardware would be exclusively responsible for the task of encoding video.

3.3.2.2. Evaluation Criteria

In addition to being able to meet the non-functional requirements, the evaluation criteria also includes the cost of the design, the ease of use by the instructor, scalability, maintainability, and ease of development.

3.3.2.3. Design Comparison

Hardware video encoders vary widely in price, so for this comparison, the price was taken for a video encoder that had specifications as close to the non-functional requirements as could be found [2]. The table below compares the differences between hardware and software encoding relating to the evaluation criteria.

	Hardware Encoding	**Software Encoding**
Cost	~$300 (tax included) [2]	Free (Assuming the software written uses free software libraries)
Setup Required	Instructors would have to power on and connect the encoder to the EML supplied laptop. This is easy in the case that it works, but adds another point of possible failure.	None

Scalability Concerns	If EML wishes to be able to have different demos for different classrooms, they would need to purchase multiple encoders.	None
Ease of Maintainability	Integrating a new encoder would require technical know-how with hardware.	Updating the encoding software would require technical know-how with software.
Ease of Development	Requires interfacing between Unity and the video encoder. This would be difficult, as a video encoder would likely expect a specific protocol and cable type to be used for input. Formatting video from Unity into this protocol and then interfacing with the laptop's IO would require some low-level and complicated software that our team does not have experience writing.	Requires interfacing between Unity and a video encoding library.
Framerate and Resolution	Resolution and framerate are specifications of the purchased encoder and an encoder could be purchased that could guarantee the ability to encode video at the resolution and framerate specified by the non-functional requirements,	Processing power required increases with higher resolution and framerate. Framerate and resolution may be limited to the minimum specified by the non-functional requirements.

Table 2. Hardware vs Software Encoding

3.3.2.4. Design Decision & Justification

We have gone forward with the decision to use software encoding. As can be seen from the table, hardware encoding has the disadvantage of being costly, while also requiring a bit of extra setup for the instructor. While the extra setup may seem simple, it does add an extra component that could fail or require troubleshooting. This extra setup is particularly important, as EML has indicated that they desire the solution to be easy to use. Any difficulty encountered setting up a hardware encoder may dissuade an instructor from using the solution entirely. Hardware also has scalability concerns if EML wishes to run multiple demos at one time, as they would need to purchase more encoders. In terms of development and maintainability, while not possible to quantify, it is likely that these designs would be of similar complexity to develop and

maintain. However, with hardware encoding, any developer who wishes to test the end-to-end solution would need direct access to an encoder, which may not always be available.

Lastly, while software encoding may limit the maximum frame-rate and resolution of the project, it was determined that it was still possible to meet the non-functional requirements. This determination was based on a quick prototype that included both frame extraction from Unity and software-based video-encoding. The framerate was tested using the Alienware laptop (minimum specified device) on a very simple machine and found to be 60fps, much greater than the required framerate of 30fps. It was then estimated that this would provide enough buffer room to account for more complex scenes.

3.3.3. Video Codec Design Decision

3.3.3.1. Additional Background Information

As video compression is a very common task, it is not surprising that there are many different compression standards available. Different standards are classified as a codec, which describes the format of the compressed video so that it can be encoded and decoded by different software and hardware. Many different codecs have been developed and have varying performance in terms of speed and compression rates. The codec with the most suitable performance for our application must therefore be chosen.

3.3.3.2. Evaluation Criteria

The first criteria for this decision is the encoding speed. This can be measured in terms of frames encoded per second. As software encoding has been chosen for this project, any codec without a software implementation capable of meeting the non-technical requirements will not be suitable for the project. The non-functional requirements and constraints to consider are framerate (**NFR1)**, resolution **(NRR3)**, and the minimum specified device (**C4).**

The second criteria for this decision is the compression rate of the encoder. A poor compression rate will lead to larger packets. Packets that are too large may require too much bandwidth to send over the network at 30 fps.

3.3.3.3. Codecs Considered

Four popular codecs are being considered for this decision. These codecs are:

1. **H.264**: The most common codec used for video compression.
2. **H.265**: The successor to H.264.
3. **AV1**: A free to use codec from the Alliance for Open Media, which is backed by large companies such as Amazon, Apple, Google, Microsoft, etc.

4. **VP9**: A free to use codec from Google.

While there are a vast array of codecs available, these are the most commonly used encoders that are available for free for the purposes of this project.

3.3.3.4. Codec Comparison

The encoding software being used can be configured to use different codecs. Because of this, it was decided that testing would be performed to determine the most suitable codec. For all codecs, the implementation was using CPU based encoding, as opposed to GPU based. The 360 frame extraction in Unity is entirely GPU based, so a CPU based encoding will help spread the processing load among the available hardware.

Testing was done in two parts. The first round of testing was done using EML's Alienware laptop and a simple Unity scene. In this case, simple refers to the textures in the scene being solid colors and the scene only containing a few objects consisting of a single texture each. During this test, the VP9 codec and H264 codecs were tested to examine the fps on the instructor side. The Alienware laptop was able to run the simple scene with H264 encoding at 60fps, while it could only run at 20fps when using VP9 encoding. As 20fps is less than 30fps (**NFR3)**, it was determined to not be feasible. Additionally, the encoding library's documentation indicates that AV1 is significantly slower than VP9, so we removed that option as well [8].

The two remaining possible codecs to use were then H264 or H265. Unfortunately, due to the Covid-19 crisis, access to the Alienware laptop was lost and this comparison had to be done using a significantly less powerful machine. Because of this, the framerate achieved for both codecs was much less than 30 when running these tests. Also important to note is these tests were run on a different machine than what was used for the validation document.

This test was run using a more complicated demo scene then the previous scene, with the intent of acquiring more realistic compressions rates, which gets worse with scene complexity. Note that this scene was approved by EML as a demo scene and more information can be found in the appendix of the Validation document. For each test, the instructor-side was run and the scene was moved around in a fixed path. A single Android phone was also connected through RTP to test how the codec might affect the client-side fps. The resolution set for this test was 1024x1024 pixels, which is less than the required resolution of 1080p (1920x1080) pixels. However, due to limitations in the Unity API, the next available resolution would be 2048x2048 pixels, and the test machine was not able to handle running the project at this resolution.

Both codecs are tunable so that it is possible to optimize for speed or compression rate. The library being used has presets to help tune these parameters. Each codec was tested using a slow, medium and fast preset. Also note that for these tests, the input frame size was 4Mb. Table 3 shows the results of the tests.

Codec	H264 Fast	H264 Medium	H264 Slow	H265 Fast	H265 Medium	H265 Slow
Average Compressed Video Frame Size (Mb)	0.12	0.03	0.03	0.01	0.01	0.02
Instructor-Side Framerate (fps	8	8	5	6	5	<1
Client-Side framerate FPS (fps)	8	8	5	5	5	<1

Table 3. H264 vs H265

Interestingly, the results of using the slow preset was worse both in terms of compression rates and speed for the H265 encoder compared to H265 medium or fast, though this may have been a result of the test laptop barely being able to run the encoding with this setting. Another interesting result was that the medium and fast settings appeared to result in the same frame rate for the H264 codec but a different compression rate, while for H265 the compression rate was similar but frame rate differed. In all cases, the client side was able to receive and decode at the same rate as the server, so no information was gained.

3.4.3.5. Design Decision & Justification

The decision was made to use the H264 codec with a medium setting. This gave an average packet size of 0.03Mb at an FPS of 8 (on a low-powered machine). This was 3 times worse compression than H265 fast encoding, but was an extra 33% faster. The decision to value speed over compression rate was made due to the fact that an average video frame size of 0.03Mb requires little bandwidth, and our solution is more likely to be constrained by processing power.

To demonstrate the network bandwidth, recall that constraint **C2** indicates the UBC network is likely to support 72Mbps of bandwidth. At 30fps, the required bandwidth is:

0.03MB * 30 frames / sec = 0.9Mbps
72Mbps / (0.9Mbps / client) = 80 clients

So with this average video-frame size, the UBC network would in theory be able to handle 80 clients, far more than the minimum of 3 clients specified by **NFR3.** This gives a large amount of leeway in the case that other projects result in worse compression rates, or if the resolution is increased. However, switching between presets, as well as switching between H264 and H265 codecs is quite simple. Because of this, both of these settings are adjustable in the main script, and there are instructions for how to change them. This way, an EML developer can change settings if they believe that their specific project would be better suited for a different preset or for H265 encoding.

One last note is that it is hard to predict how these tests may have differed if running on a more powerful machine; in particular, at a higher fps it is possible that the client side may have struggled to decode at the rate that the instructor-side was encoding.

3.4. Video Server

3.4.1. Background

The video server component is responsible for hosting the 360 video extracted from the Unity project and serving it to the students to view.

3.4.2. Evaluation Criteria

A number of criteria are important to consider for this design decision. The designs that are considered in the next section were evaluated based on their effect on the latency (NTR2) as well as their impact on the ability to run the solution on EML's provided device (C4). Also considered are the potential costs of the design option, the difficulty of maintenance for EML after project completion, and the complexity of the set up procedure for an instructor. As would be expected, the optimal solution would be free, require no maintenance, and would require no additional setup steps for the instructor.

3.4.3. Designs Considered

Three options (depicted in figure 4) have been considered for this component of the solution:

Option 1 - Instructor as host: The first option is to host the live stream directly on the instructor's device. In this solution, the instructor's device would continuously run a network server that would listen for and accept connection requests from each student device. The server would then stream the 360-degree video to each connected student device. The primary benefits of this option is that it reduces the complexity of the solution, as well as minimizes the cost - See section 3.3.4 for a more detailed explanation of the benefits. However, this approach adds additional strain on the instructor-side device.

Option 2 - Local device as host: The second option is similar to option 1, but offloads the burden of serving video to a separate but dedicated device that will act as the server. This dedicated device will be connected to the same network as the student's devices, and will

accept network connections from - and serve 360 degree video to - each student device. For this option, the instructor's device would communicate with the dedicated device either through networking protocols, or through a direct wired connection, such as USB or HDMI. This would reduce the load on the instructor's computer, as it would only have to handle one active connection, as opposed to a connection for each student device.

Option 3 - Cloud as host: The final option considered is to host the livestream on a 3rd party cloud service, such as Amazon Web Services (AWS). In this option, the instructor's device would send the 360-degree video over the public internet to a 3rd-party cloud service, which would then serve the video back to each student device, also through the public internet. This option, similar to option 2, would reduce the workload on the instructor's device, but would not require an instructor to set up an additional physical device.

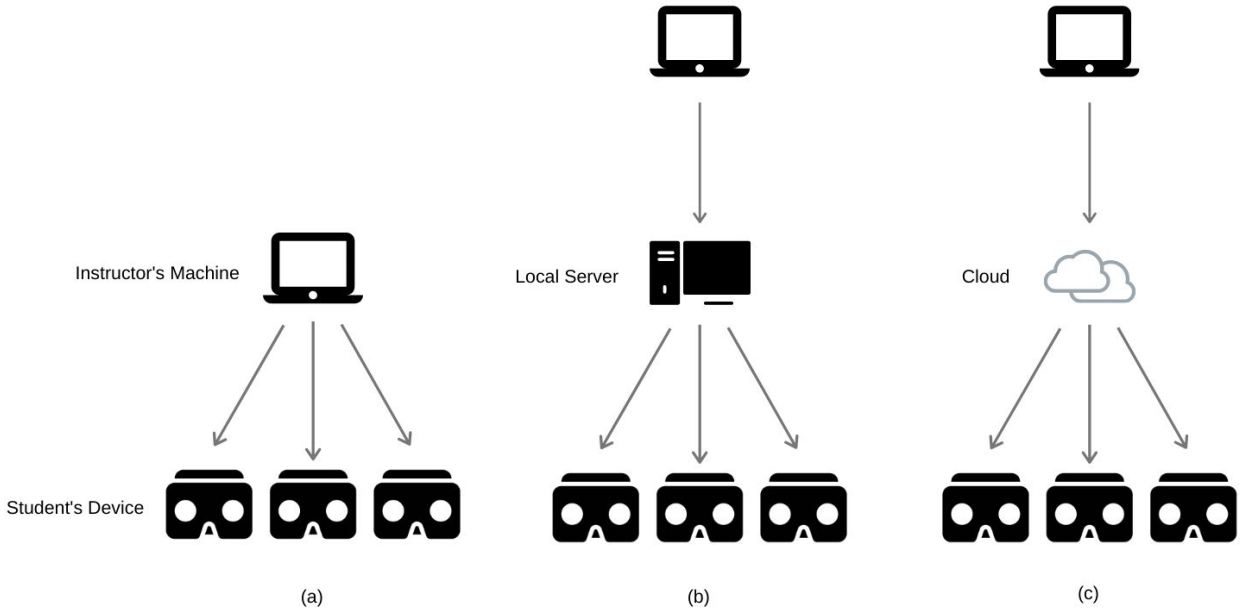

Figure 4. Three Video Server Options. (a) Instructor as host; (b) Local Device as host; (c) Cloud as host.

3.4.4. Design Comparison

	Latency	EML-Device Considerations	Cost	Difficulty of Maintenance	Difficulty of Setup
Option 1 - Instructor as Host	<3ms (latency of local wifi network)	Serving the video on the EML Laptop directly will increase the workload of the machine.	None	None - server is integrated into Unity plugin so no additional maintenance is needed	None - server is integrated into Unity plugin and runs as part of the plugin
Option 2 - Local Device as Host	<3ms (latency of local wifi network)	None	$1,629.99 for Alienware laptop with similar specs to instructor laptop	Medium - Must maintain a separate device	High - Separate device must be configured and connected to instructor laptop
Option 3 - Cloud as Host	20-60ms (latency to AWS server located in Oregon)	None	$122 per month for AWS c5.xlarge server	Low - maintenance handled by external company	High - Must ensure cloud server is running and connected

Table 4. Design Comparison for Video Server Implementations

3.4.5. Design Justification

The current design decision is to host the video directly on the instructor's computer. This design decision is similar to the encoding design decision, in that the benefits of this solution are that it is free and low in complexity. Also similar to the encoding design decision, is that hosting the video directly will add to the processing power required of the instructor's computer (the EML provided Alienware laptop).

3.5. Application-Level Streaming Protocol

3.5.1 Background

As was decided in the data transmission section (section 3.2), the instructor-side computer will transmit HD 360 video to client-devices over a computer network. The previous sections discussed the encoding and serving of the video on the instructor-side machine. However, another important decision is the streaming protocol that will be used to send the video over the network. A streaming protocol dictates how a video is broken down into multiple packets, sent over the network, and then reassembled by the client. In addition, streaming protocols often provide other services, such as jitter compensation, which prevents variable framerate on the client-side in the case that video frames arrive on the client-side in short bursts.

3.5.3. Evaluation Criteria

As can be seen in the following section, streaming protocols often add significant latency to the stream. This makes latency a major point of consideration in terms of the non-functional requirements of the project. Any effect on the achievable frame rate of the stream will also be considered.

Outside of the non-functional requirements, useful services offered by the streaming protocol will also be considered. For instance, jitter compensation, which was briefly described above, improves the client-side experience by smoothing out the frame rate of the video. We are considering this to be relevant and useful in terms of this project. However, other streaming protocol services are not relevant. For instance, many streaming protocols offer adaptive bitrate, which enables the stream to switch between different available bitrates depending on the available bandwidth of the network. For instance, if a video was available to be streamed in different resolutions, a client may start streaming at a high resolution, and then drop to a lower resolution if the network was unable to handle the required bitrate of the high resolution stream. As the stream for this project will only be available in a single fixed bitrate, this service is not considered relevant or useful for this project.

3.5.3. Protocols Considered

The company Wowza, has created a good comparison of many streaming protocols [4]. From examining this breakdown, it can be immediately seen that many streaming protocols introduce more than 5 seconds of latency into the stream which would fail the latency non-technical requirement. For instance, HLS, MPEG-DASH, and RTMP are all streaming protocols that do not meet latency requirements, and will therefore not be examined further.

Some other protocols on this list that did meet the latency requirement were also easy to remove from the list of potential protocols. For instance, on the list was a tuned version of RTMP

(RTMP-Tuned). However, this protocol is meant for use with Adobe Flash Player [5], which is meant for use in browsers and will be deprecated by the end of the year [6]. WebRTC is another protocol mentioned in the list, but it is also browser based and meant for browser-based use cases, such as video-calls [7].

Two protocols that were suitable for our application were the Real-Time-Protocol (RTP) and a protocol not in the list from the previous link called ZeroMQ Publisher/Subscriber, which will be compared in the following section.

3.5.4. Design Decision

The main benefit of the ZeroMQ Publisher/Subscriber protocol over RTP is its implementation simplicity. We were confident that we could get a demonstrable prototype quickly and easily with this protocol. We also believed that its simplicity would make it easy to maintain by future project maintainers. However, it is a general use protocol and not a dedicated streaming protocol. Because of this, it was missing certain features, such as jitter compensation. RTP on the other hand, does have jitter compensation.

Additionally, it runs over the Transmission Control Protocol (TCP) as opposed to RTP, which uses the Universal Datagram Protocol (UDP). UDP is more suited to video streaming applications, as TCP can add extra latency to the stream due to its reliable transmission design. Additionally, TCP has underlying logic that can throttle the sending rate of packets. This could lower the achievable fps of the stream.

Because of these reasons, we chose to use RTP as the streaming protocol for this application.

3.6. Client-Side Application Platform

3.6.1. Background

As discussed in the Design Overview section of this document, the client-side of the application is responsible for receiving and displaying the 360-video (**FR1)**, while allowing the student *3DOF (***FR2)***,* and must run separately on each student's device (**FR3)**. The major design decision pertaining to this component of the solution is to choose the most suitable platform to build the component around.

3.6.2. Evaluation Criteria

Before discussing the criteria used for this decision, it is important to note that it is assumed the student's device is capable of streaming and displaying a video at 1080p resolution (**NFR1)** at 30fps (**NFR3)**. Any device unable to meet these requirements will not be considered in our

design and will not be supported due to their fundamental inability to meet the non-functional requirements of this project regardless of the design of the final solution. Some common supported devices, based on Android are: Samsung Galaxy S10, Google Pixel 4, OnePlus 7T and others.

The non-functional requirement considered for this decision is the maximum latency of 5 seconds (**NFR2)**. The other main considerations are the ease of installation and use by a student, the difficulty of development, and the difficulty of maintenance for developers at EML. Note that this decision should have no impact on the amount of clients that can connect at one time (**NFR4)** or the amount of required bandwidth from the network. Additionally, no solution will add cost to the project. Lastly, while different solutions may have differences in battery usage, EML's VR demos are not generally longer than 10 minutes, and so the battery usage is being considered negligible and will not be an evaluation criteria.

3.6.3. Designs Considered

Two approaches to the client side video viewing platform are considered. The first approach is a dedicated cross-platform Unity application which will have to be installed by the student who is viewing the livestream. This application can be downloaded once, and used to view the livestream of all EML demos. The second approach is to host a website to display the livestream, and allow students to use their browser to connect to the website and view the livestream in VR. An important note for the first approach is that accounts are not needed for students to use Unity-based applications, and EML has the means to publish an app to the Google Play store so that it is a verified application.

3.6.4. Design Comparison

The following table gives an overview of the two considered design solutions.

	Unity-Based Application	Web-Based Application
Latency	A Unity-Based application is not constrained to any networking protocols. Any suitable networking protocol can therefore be chosen to meet this requirement, so latency is not a concern.	Constrained to web compatible streaming protocols which are http-based. Common protocols have a latency of higher than 5 seconds (**NFR2)** [3]. Some much newer protocols promise latency within our requirements, but relying on new and less common protocols would add technical risk to the project.
Maintenance	EML uses Unity for their own projects, so maintaining a Unity project should be within their capability. EML would need to maintain an active application on mobile app stores such as Google Play and Apple's App Store.	Not all EML developers will have web experience. May be relatively difficult to maintain if changes need to be made to the project.
Development	Can be sure that the data on instructor and client sides is compatible. For instance, it was found that video frames are stored upside-down in Unity compared to what is generally considered standard. Therefore, manual flipping of frames would be required for any non-Unity based solution. This would likely be either complicated or impact performance, so a Unity application significantly reduces technical risk compared to other solutions.	Possible compatibility issues with different browsers. Would require hosting a website to serve the video player. Potential data incompatibilities from using a different platform for the client (web) than the instructor (Unity).
Ease-of-use	Requires a user to download an application which requires time and space on their device.	Easy-to-use - A student simply needs to open a web-browser and navigate to a specified URL.

Table 5. Unity Application vs Web Application

3.6.5. Design Justification

The decision was made to use a Unity-based application. While the web-based application has the advantage of not requiring a user to install a separate application, a Unity application would be easier to develop and maintain by EML. Furthermore, relying on new http-based protocols would add significant technical risk to the project, and may lead to the solution not meeting **NFR2.** Possible data incompatibility may also lead to poor performance or additional development time and adds to the technical risk of a web-based solution.

4. Conclusion

The document provided a design overview as well as specifics on design decisions for each of the three components of our proposed solution to the client problem.

1. **Instructor side application:** The main VR application from which data would be streamed to student devices. This application extracts 360 degree video frames from Unity, encodes them, and then sends them over the network to student devices.

2. **Client side application:** The application each student would run on their devices to view the livestream. This has been implemented in Unity.

3. **Network component:** This component handles transmission of data from the instructor-side to the client side. The networking protocol chosen was RTP.

It is our hope that the reader now has a complete understanding of the system components and their interactions and that this will enable them to maintain or improve the final product if necessary.

5. Glossary

- **6DOF:** Six degrees of freedom. A user can look around a 360 degree field of view and move in all directions in the VR world (forward, backward, right, left, up, down, or any combination of the six)
- **3DOF:** Three degrees of freedom. A user can look around a 360 field of view ***but are locked in one position, and cannot move in the VR world.***

6. References

[1] N. Smalls, "Software vs. Hardware Encoders for Live Video Streams," *dacast.com,* Oct 30, 2019. [Online] Available: "https://www.dacast.com/blog/software-vs-hardware-encoders-for-live-video-streams/" [Accessed Feb. 9, 2020].

[2] "URayTech MPEG4 Full HD 1080P 1080i H.264 HDMI Video Encoder," *amazon.ca*, [Online]. Available: https://www.amazon.ca/URayTech-Encoder-Protocols-Facebook-Platforms/dp/B07D78L3SZ/ref=sr_1_1_sspa?keywords=hardware%2Bencoder&qid=1581297267&sr=8-1-spons&spLa=ZW5jcnlwdGVkUXVhbGlmaWVyPUExV1VJSFdUOVRHVzJOJmVuY3J5cHRlZElkPUEwMjU3MDUzMVZMVlQ0S1o3V1JDRiZlbmNyeXB0ZWRBZElkPUEwODk2OTM4MVhRU05WUlJLUEE2MSZ3aWRnZXROYW1lPXNwX2F0ZiZhY3Rpb249Y2xpY2tSZWRpcmVjdCZkb05vdExvZ0NsaWNrPXRydWU&th=1 [Accessed Feb, 9, 2020].

[3] "Low Latency Streaming". *Wowza.com,* [Online]. Available: https://www.wowza.com/low-latency [Accessed Feb. 9, 2020].

[4] "Streaming Protocols". *Wowza.com,* [Online]. Available [https://www.wowza.com/blog/streaming-protocols [Accessed Apr. 8, 2020].

[5] "Real-Time Messaging Protocol Specification" *adobe.com*, [Online]. Available: https://www.adobe.com/devnet/rtmp.html [Accessed Apr. 8, 2020]

[6] "Adobe Flash Player" *wikipedia.com* [Online]. Available https://en.wikipedia.org/wiki/Adobe_Flash_Player [Accessed Apr. 8, 2020]

[7] "WebRTC" *webrtc.org* [Online]. Available: https://webrtc.org/ [Accessed Apr. 8, 2020]

[8] "Encode/AV1" *ffmpeg.org* [Online]. Available: https://trac.ffmpeg.org/wiki/Encode/AV1 [Accessed Apr. 8, 2020].

[9]. "Netflix Help Center", *netflix.com* [Online]. Available: https://help.netflix.com/en/node/306 [Accessed Apr. 8, 2020].