3VTC: 360^o Video Teleconferencing

1st Harshal Soni
dept. of Computing Science
University of Alberta
Edmonton, Canada
hsoni@ualberta.ca

2nd Frincy Clement
dept. of Computing Science
University of Alberta
Edmonton, Canada
frincy@ualberta.ca

3rd Belqes Mohammed
dept. of Computing Science
University of Alberta
Edmonton, Canada
belqes@ualberta.ca

***Abstract*—The advancements in communication technologies will open new frontiers in the domain of teleconferencing and video conferencing. Current applications of video conferencing are solely based on streaming 2D videos over the network. However, the new research towards 360^o video transmission has the potential to revamp the video calling experience.**

We are the pioneers in this domain by creating a mobile application that coalesces the stereoscopic Virtual Reality (VR) video streaming and monoscopic video streaming on the same platform using the same 360^o video. The application is powered by Android on client-side and PANO [1] video streaming strategy on the server-side. We have empowered the entire application to support multiple clients by orchestrating the entire server on Kubernetes [2] and Docker [3] containers on our local machine.

Our evaluation has suggested that our application can run even in poor network conditions, just under 10 Mbps network link. Our optimizations to PANO streaming strategy for teleconferencing application saves bandwidth consumption upto 38% compared to the original 360^o videos and upto 46% compared to PANO baseline version.

***Index Terms*—360^o video,video teleconferencing, PANO, video streaming optimization**

I. Introduction

Teleconferencing has completely changed the way people interact with each other. The progress made in Virtual Reality (VR) has opened more possibilities with the development of virtual meeting rooms where people can see each other in either avatar form or in a theatre mode. However, there are no applications currently available that can playback synchronized 360^o video in video calling fashion. The only development which is remotely close to virtual teleconferencing is implemented by Oculus in their meeting rooms, but it uses animated avatars to symbolize every user.

In this project, we built a prototype of a 360^o video-conferencing application that can be played on full screen or head mounted devices (HMDs). We developed a client-server video streaming application using RTSP or Real-Time Streaming protocol over a secure TCP connection. The server is deployed in a scalable and highly efficient Kubernetes node which helps in creating a swarm of server workers, when multiple clients request for streaming. The optimized video is streamed from server to client. We built an Android application which can request the video to the server as well as select the type of player. If the user wishes to view in HMD, we have a stereoscopic player developed in Unity3D to support the request. If the user wishes to view in full-screen we have support for that as well using Clappr.js video player. Thus, our prototype encompasses a complete flow of video streaming starting from input video for optimizing to displaying the video on the user's favourite display. This application opens the opportunity for one to one virtual video call in 360^o VR, teleconferencing in VR as well as streaming services in VR which can provide a definitive sense of reality based on live 360^o video streaming rather than animated avatars.

We have utilized a unique video streaming strategy, PANO [1] as our baseline model which is unlike other viewport driven approaches. PANO operates on estimating and maximizing perceived quality modelled by PSPNR (Perceived Quality Noise to Signal Ratio) and not the traditional measure of PSNR (Peak Signal to Noise Ratio) for non-360^o videos. PANO has considered the subjective aspect of human attention span, while developing the 360^o video quality model. Furthermore, they added a unique tiling scheme different from equal tiling scheme used by other viewport driven approaches. Since PANO was able to derive large savings in bandwidth, it was selected as the baseline of our project.

In a video teleconference, there will be multiple users accessing the application at the same time. The motion of object (people) in viewport will be very less compared to a regular 360^o video which might have faster moving object in the viewport. Also, sometimes the video conferencing would not need the whole sphere of 360^o, but a 180^o would be enough [27] to cover the area of conference. By taking these two specifics into consideration, we have further optimized PANO for teleconferencing application.

Our proposed application, 3VTC, entails two major contributions: First, we built a multi-user mobile application (upto 8 users), which is the first ever to use 360^o videos streaming for teleconferencing. It has capabilities to play on full-screen monoscopic mode as well as stereoscopic mode to be used in HMDs. It can play videos in theatre-like mode and at the same time can be easily extended to a full-fledged 360^o VOIP system.

Second, we optimized PANO, our baseline video streaming strategy, for the application of teleconferencing. Both of our optimized versions proved to be working at lesser bandwidth, while maintaining higher quality than the baseline PANO version. Our optimized 180^o version saves 38% bandwidth compared to the original video and around 20-46% compared to the PANO baseline version. Our optimized 360^o version

saves 23-26% bandwidth compared to original video and upto 33% compared to PANO baseline version. This has been achieved while maintaining higher PSPNR values compared to the baseline PANO version.

The remainder of this paper is organized as follows. Section II explores the most recent works in the field of 360^o video streaming strategy followed by an overview of our baseline strategy - PANO in Section III. The project scope, detailed explanation of our development work, environment and novelty are provided in Section IV and V. The methodology and metrics employed for our evaluations and their results are given in Section VI. The remainder of the paper covers project status, project contributions, conclusion, future work, references and team roles and responsibilities.

II. Related Work

There has been ongoing research in the field of optimizing 360^o video streaming. Most of the work has been focused on viewport-driven approaches or on predicting factors such as user's movements that would affect the viewport. In this section, some of the recent work using those approaches are discussed.

In 2016, Sreedhar et. al [4] introduced a concept that discussed a new method using fusion of both equirectangular and cubemap projections from the traditional pyramidal approach. Later in the same year, Bao et. al introduced another method which considered user's movements to predict the current viewport. As per Bao et. al [5], in a 360^o video only 20% percent of the view will be actually displayed. 360^o videos are mostly displayed at 6K resolution rather than 4K for better clarity which increases its bandwidth consumption by 4 or 6 times. Hence, predicting the user viewpoint using regression techniques employing neural networks, would help in reducing bandwidth.

In 2017, Petrangeli et. al [6] presented a tile-based method that treats each image in the 360^o video as a tile. They used a server push method of HTTP/2, to push the required images, with a single push request from the client side. This approach also produced a reduction in bandwidth consumption up-to 35% compared to non-tiled VR streaming solutions.

In the same year, Fan et. al [7] introduced another concept that tried to predict the user's fixation point using neural networks. It is different from the previous concepts as it uses both sensor data and the content data concurrently. Tile-based method and orientation-based method were studied in this research. In the Tile based method, the images seen by the user is considered as a tile and an LSTM network gives out 87% accuracy for classification of the tiles . In the orientation-based method, the historical orientation of the user was studied using sensor data and it gave an accuracy of 89% for the same.

Another work by Corbillon et. al [8] introduced a new technique called Viewport-Adaptive 360^o video delivery. Normally, 360^o videos take a lot of bit rate and the VR head gears have to react in 10 ms for any head movement. Their proposed method reduces the required bit rate by finding out the area where the user is watching (called viewport). The quality of video for the viewport area is increased and other segments where users are not watching is reduced which helps in reducing the bit rate.

In 2018, 'ClusTile' introduced by Zhou et. al [9], used clusters of images tiles to improvise and optimize the time taken to send the images to the user. This clustering technique can save up to 76% bandwidth compared to the standard 360^o streaming , and reduces bandwidth consumption by 52% compared to fixed-tiling schemes.

Qian et. al [10] introduced 'Flare' in the same year, which discussed new algorithms to provide viewport adaptive streaming and have reduced the bandwidth up to 35% over Wi-Fi and enhanced the video quality by 4.9 times over LTE networks, compared to non-viewport-adaptive approaches.

In 2019, Yu et. al [1] introduced PANO, a 360^o video streaming strategy which is based on perceived quality unlike the other view-port driven approaches. We have chosen PANO as the baseline for our 360^o video teleconferencing application because of the huge bandwidth savings it has achieved without reducing the perceived quality of videos.

III. Our Baseline Model - PANO

PANO approaches the 360^o video streaming problem with an understanding that users want to perceive the 360^o video in the same quality as a non-360^o video, but this would mean a lot of additional bandwidth. PANO strategy is modelled on the perceived quality metric (PSPNR) which is based on three new quality determining factors, which considers the users limited span of attention. It proposes a variable-sized tiling scheme to capture the variation in user attention distributed throughout the viewport area, which is unlike the other fixed-tiling schemes. This strikes a balance between perceived quality and video encoding efficiency. It provides robust quality adaptation methods that help to deliver the encoded visual quality, readily deployable at the client system. Moreover, PANO saves 41-46% of bandwidth without reducing the perceived quality or it increases the perceived quality by 25-142% which is measured in terms of Mean Opinion Scores (MOS).

A. Quality-Determining Factors

PANO understands that the user perceived quality of 360^o video is uniquely affected by the user's viewpoint movements. The following are the three quality-determining factors which PANO considers as those which are affected by the viewpoint movements.

1) Relative moving speed between viewpoint and visual objects in the region.
2) Difference in Depth of Field (DoF) between region and viewpoint.
3) Change in illuminance of viewport in a timeframe

B. Quality Modelling in PANO

PANO models the video quality in a new metric known as Peak Signal-toPerceptible-Noise ratio (PSPNR), which is an improvisation on the classic metric Peak Signal-to-Noise ratio (PSNR). PSPNR filters out the distortions in quality that are

not perceptible by the users. It is modelled in terms of *JND* or *Just Noticeable Difference*, which is the minimal difference in pixel values noticeable by the users. PSPNR is defined in terms of JND as follows.

$$P(q) = 20 \times log_{10} \frac{255}{M(q)} \tag{1}$$

$$M(q) = \frac{1}{S} \sum_{i,j} \left[|p_{i,j} - \hat{p}_{i,j}| - JND_{i,j} \right]^2 \times \Delta(i,j) \tag{2}$$

$$\Delta_{i,j} = \begin{cases} 1, |p_{i,j} - \hat{p}_{i,j}| \geq JND_{i,j} \\ 0, |p_{i,j} - \hat{p}_{i,j}| < JND_{i,j} \end{cases} \tag{3}$$

where *P(q)*, *M(q)* is the PSPNR of image at quality level *q*, $JND_{i,j}$ denotes the JND at pixel *(i,j)*, $p_{i,j}$ and $\hat{p}_{i,j}$ are the pixel values at pixel *(i,j)* in original and encoded images, which is of size *S*. Equation (3) defines the condition for considering a pixel change in calculating the perceived quality. If only the changes in pixel values after encoding is greater than the JND at that location, then it will be attributed to the calculation in perceived quality.

To profile the JND for 360^o videos, the authors studied the quality determining factors individually as well as combined, two at a time. The results modelled JND in terms of *Content-dependent JND* $C_{i,j}$ and *Action-dependent ratio* $A(x_1, x_2, x_3)$ which is given by the below equation.

$$JND_{i,j} = C_{i,j} \cdot F_v(x_1) \cdot F_d(x_2) \cdot F_l(x_3) \triangleq C_{i,j} \cdot A(x_1, x_2, x_3) \tag{4}$$

Content-dependent JND is the JND when the viewpoint movement is zero and *Action-dependent ratio* is the product of three factor multipliers $F_v(x_1)$, $F_d(x_2)$ and $F_l(x_3)$. A factor multiplier is calculated as the ratio between the JND when the factor (e.g: relative velocity of object) is a constant *x* and the factor is zero.

C. Tiling Scheme in PANO

PANO uses a variable sized tiling scheme, which starts with splitting the video chunks into fine-grained tiles. The efficiency scores of each tile is calculated which is based on the PSPNR values. Tiles with similar efficiency scores are then grouped together to form a larger tile.

D. Video Quality Adaptation in PANO

In PANO the quality is adapted at chunk level as well as tile level. The quality level of the tile group is determined in such a way as to maximize the overall PSPNR without increasing the size of the tile more than the bit rate of the chunks. They also adopted a pruned search instead of an exhaustive search in assigning the quality levels to the tiles, thus preserving the computing power. PANO stores PSPNR values for all the different combinations of quality determining factors in a PSPNR lookup table, which is optimized and sent to the client as a manifest file. PANO has also provided methods in keeping the additional system overhead like pre-processing time for calculating PSPNR to a minimum.

At the client side, an estimator compares the predicted viewpoint movements with the information in the manifest file, for each specific tile location. This information is converted into new PSPNR quality levels for the tiles and are rendered at the client player.

IV. Project Scope

The proposed application has to be developed in two stages, theatre mode and live - teleconferencing application. It is built on a mobile application and can be viewed via HMDs or on a full-screen video player. We have completed the first stage which can be extended easily to the second stage to create a seamless multi-user 360^o teleconferencing application.

In theatre mode, the server pushes a pre-downloaded 360^o video to users synchronically. These are built on top of streaming server-client strategies. We covered network transmission, encoding, and GUI part in this stage.

In live mode, there will be two phases. In the first phase, only one user will be broadcasting its surrounding environment using a 360^o camera to the other user using implemented theatre transmission schemes. Furthermore, the key feature addition will be an underlying audio stream between two users that will be streamed along with the 360^o video. Hence, only one user will be transmitting the video stream, but they will be able to talk to each other in Voice-over-IP fashion. In the second phase, to-and-fro communication can be set up in a multi-cast teleconferencing fashion enabling both the users to share their surrounding feeds using their set of 360^o cameras and be able to see the other on stereoscopic screen. This shall be the complete 360^o teleconferencing experience to both the users.

V. Development Work and Novelty

In this project, we developed a prototype of an application for 360^o teleconferencing. Even though 360^o videos have been around for some time, there has never been a dedicated application for teleconferencing involving them. We captured that opportunity and created the first stage – "Theater Mode", which allows one-way transmission of 360^o videos from server to client.

We used PANO as our baseline for optimizing the videos before transmission, so as to save enormous amount of bandwidth without dropping perceived quality. In our novel application, we have further optimized PANO in two ways: 1) 360^o videos with constant velocity of object in motion as there is lesser motion in teleconferencing applications; 2) 180^o videos with constant velocity to accommodate video conferencing which needs lesser viewport requirements.

A. Development Environment

For developing our application, we have used a wide variety of tools, protocols and libraries.

1) Client Server System: A client server application was created for video transmission using RTP/RTSP protocol over secure TCP connection

- RTSP [11] or Real-Time Streaming Protocol is a protocol used for streaming multimedia in real time between two end points. It uses a TCP connection to establish and maintain connection between the end points. It allows two-way control messages between server and client, even though most of the communication is from client to server. To control the video transmission in real-time from server to client, the client application uses play, pause and other features. In our teleconferencing application, we have utilized the same to add play and pause functionality.
- RTP [12] and RTCP are two networks protocols used by RTSP for performing real-time transmission. RTP (Real-time Transport Protocol) helps in transporting audio and video whereas RTCP (RTP Control Protocol) helps in maintaining and monitoring statistics, quality and synchronization of the transmission.
- How does RTSP work [13] : To initiate a session, the client application sends an RTSP request to the server with the available options such as play or pause. The server receives the request and sends its response with list of available options it can accept. Clients learns about them and requests the description of multimedia to which server responds with the description. Clients then proceed with a setup request to server and once it is done, streaming session will start between server and client.

2) Full-Screen 360^o video player: A 360^o video player was created using Clappr.js for the client-side to play the streaming 360^o video from the server.
   - Clappr [14] is a video player for the web, which has a wide variety of capabilities including playing 360^o videos with control options. The video player itself can be customized to add different functionalities including chromecast options, thumbnail on seeking, markers and tooltips, streaming quality level selector and so on. The clappr package can be installed using Node Package Manager (npm) from its GitHub project link `git clone https://github.com/clappr/clappr.git`. It supports a variety of browsers like Google Chrome, Microsoft Edge, Firefox, Internet Explorer 11, Safari, recent versions of Chrome Android and Safari iOS.
   - To use Clappr for 360^o videos, ensure *clappr-video360* [15] is installed using `npm install clappr clappr-video360`. The libraries *clappr.min.js* and *clappr-video360.min.js* is added to the web page and a player is created using Clappr. Player object with a container-specific to 360^o videos and input from a 360 equirectangular video source.

3) Cardboard-style video player for Head-mounted Display(HMD): A 360^o VR video player for experiencing the content on head-mounted displays(HMDs) directly from the application was created using Unity3D.
   - Unity headjack SDK [16] [29] provides the support for building VR driven applications and also supports adding 360^o and 180^o videos as a texture. We developed a player by centring the main camera at the centre of the sphere and inverting the normal of the sphere to make them face towards the main camera. Next, we projected the videos as a material texture on the sphere so that the main camera is able to look at the sphere and see the texture of the sphere as the content of the video. In order to change the viewport of the camera, a script was attached to the main camera that changes the camera's orientation based on the movement of the device(values from gyroscope) or user's interaction in the video.
   - The novelty of this player is that lightweight in comparison to other pre-existing players for 360^os VR. On top of that, when the user pinches the screen to zoom in or zoom-out, the player automatically calculates the zoon and changes the camera's coordinates in the direction of normal. hence, the user can access the functionality of zooming. This makes the experience interactive and immersive. To create the video player in Unity, we used two tools from the Unity asset store: VRTK and Steam VR plugin.
   - VRTK [17] (Virtual Reality Toolkit) provides a suite of tools for developing VR solutions using Unity3D. We used it for our application as it supports StreamVR. It helps in simulating VR applications without using VR devices. Different types of interactions and functionalities like pointer interactions, body physics, object interactions, 2D and 3D controls etc. can be added by just using VRTK.
   - SteamVR Plugin [18] is used to interface SteamVR with Unity. SteamVR is a set of tools and services to create VR applications. It helps to create one application that can be used by multiple types of VR devices such as Oculus, HTV Vive, and Google Cardboard. To use this plugin, SteamVR runtime must be installed in the system. The plugin for Unity can be downloaded from the Unity asset store.

4) Threading and Queue (Python libraries): For our teleconferencing application, multiple clients were required to be connected to the server at the same time. To handle the processing power required for the same, we implemented our client-sever application to allow parallel computation using multiple threads. We used thread module from threading library and Queue modules from Queue library.
   - Thread [19] is created whenever a new client joins the system. When the client connects to server over

TCP, the Server initiates Server Worker as a new thread. This helps in maximizing the utilization of CPU when multiple parts of an application are running simultaneously. They are lightweight as they share the same pool of memory. It makes the application efficient, as it reduces the idle time of CPU.

- Queue [20] classes from the Queue library help to create FIFO (First in First Out) queue. We used Queue in our application to place the client information in a queue so that the threads can access as it gets created for the corresponding client application.

5) FFmpeg: Tool for processing video and audio after video optimization by PANO
   - FFmpeg [21] is an open-source suite of solutions to process video, audio and multimedia. It is basically developed for multimedia processing from the command line. It supports a wide variety of input formats and protocols. We have used FFmpeg for many applications: (i) To mix audio with the optimized video output of PANO, (ii) To crop 360 videos into 180^o video for optimizing our application for teleconferencing and (iii) To calculate PSNR and SSIM.
   - PANO system also uses FFmpeg to slide the video into chunks and use those chunks to assign a quality to them. Furthermore, it uses FFmpeg to prepare the manifest file of the video which is sent to the client for adaptive video playback.
6) Purplepill VR Unity SDK Tools: PurplePill [29] provides interactive Unity templates that we have used for making our android application backbone. It provides database services, application controlling, and client-server integration tools. We have used it for encapsulating our controllers, decoders, playback video services, and platform orchestration between full-screen mode and Google cardboard VR mode.
7) youtube-dl: To create equirectangular videos from stereoscopic 360^o videos, we used an open-source library called youtube-dl [22]
   - It is a command-line program that helps to download YouTube videos with several options for download, video selection, video formats, subtitles and so on. We used the command youtube-dl -f bestvideo[ext=mp4] –user-agent " URL to download YouTube 360^o videos into equirectangular videos which are the format we used as input to PANO and our 360^o players.
8) Pano: Optimization strategy for 360^o video transmission
   - As discussed in the related work, PANO [1] video streaming strategy was used for preprocessing our 360^o videos before streaming over RTSP. PANO takes into account of optimizing the video without reducing perceived quality by taking into account of the difference in attention span of humans with respect to the motion of an object, depth of field and brightness changes. It gives a quality model, tiling scheme and quality adaptation strategy at the client side for the seamless delivery of these optimized videos. We have utilized the source code-shared by the creators of PANO to use it as baseline and added our own contributions to optimize the strategy for the application of teleconferencing.
9) Docker: Docker [3] is an OS virtualization platform service which lets a software to be run inside any local machine, or cloud or both.
   - To improve the throughput of the server, we have developed the sever as a docker container. Our server is hosted on the local machine, specifically one is windows device and another is Linux device. Thus, the usage of OS virtualization was necessitated and docker enabled us to support our server in a heterogeneous environment.
10) Kubernetes: Kubernetes [2] is an open-source container used for scaling, deploying and managing applications
    - Since there are multiple docker containers in our application, we have used Kubernetes to balance the load between both the minion nodes of server and also orchestrate the entire process. Also, It creates a swarm cluster of server worker instances on demand and marshals the user request to the appropriate node which is required for our teleconferencing application as multiple clients access the app at the same time.
11) Android Studio: The Teleconferencing Android app was created in Android studio platform [23]by importing Unity Engine and purplepillvr Unity Plugin [29], which is one of the best IDE available for creating Android apps.
12) Unity 3D: The 360^o video player for the HMDs was created in Unity 3D platform [16], which provides assets and tools to create, simulate and integrate VR applications.
13) Visual Studio: The custom codes for Client-Server application, full-screen 360^o video player and evaluation metrics were developed on Visual Studio platform.
14) MATLAB: Majority of scripts in our baseline model, PANO, has been developed in MATLAB. It includes calculation of Quality model, Tiling Scheme and creation of manifest file. We modified the velocity parameter in the Quality module to optimize the input video for teleconferencing application by setting the velocity threshold as constant equal to 10.f, where f is the velocity with which optical flow of the content inside the video moves.
15) Latency calculator: Latency of each and every video and app component is measured using Android's in-built clock.
16) VideoFrame.js and Math.js: For evaluating the quality of video streaming using optimized PANO for teleconferencing compared to the original video and PANO,

we calculated the standard deviation in frame rate of the video streaming. VideoFrame.js [24], an opensource javascript library was used to extract the video frames for frame rate calculation. Math.js [25] another opensource javascript library was used to perform calculations for standard deviation in real-time on the web 360^o player using clappr.

B. System Parameters

1) Network Conditions: We have tested our system in a normal home network to ensure that it works in slower network conditions as well. The network on which the system is tested can be defined as follows. Network link: 100 Mbps, ping: 13ms, jitter: 4ms, download speed: 79.1 Mbps and upload speed: 10.5 Mbps.

2) Codec Details: The videos are encoded at Advanced Video Codec h.265 which is a high efficiency codec into mp4 containers. CRF or constant rate factor ensures the quality setting for the encoder. Lower the CRF, higher the quality. For h.265 default is 28, and we used 18 for our application. In short, these are the codec details: Key frame interval: 2 sec, AVC h.265/HEVC @ CRF level 18 [28] and maximum resolution: 4K.

3) Hardware: The hardware on which our system was developed are as follows: Intel Xeon Gold 6234 (3.30GHz, up to 4.00GHz with Turbo Boost, 8 Cores, 25MB Cache), RTX Quadro 5000 16 GB GDDR6 and RAM: 32GB + 12 GB OS. The faster processor allowed us to host the server in the local machine, which helped us learn more about network programming. The high power graphics card also enabled us to test the multi-used 360^o video rendering at client side, before deploying it into Kubernetes.

C. Implementation Pipeline

The objective of our project is to create a 360^o teleconferencing application. We proposed to implement the same in two stages: Theatre mode and live mode of teleconferencing. For this semester, we were able to implement a working prototype of theatre mode. In Theatre mode, the server pushes a predownloaded 360^o video to multiple users whereas in live mode there will be live two-way 360^o teleconferencing between multiple users with the ability to talk to each other.

1) Input Dataset: The pipeline of theatre mode starts with input data of a pre-downloaded 360^o video in the equirectangular format. It can be downloaded from YouTube 360^o videos which are equirectangular using a python command-line program youtube-dl. The downloaded video will be in mp4 format and is equirectangular with the same resolution as it is available online. For our prototype and evaluation, we have used two sample videos, a self-shot video at University of Alberta and a 360^o video downloaded from vimeo [26].

2) Video Optimization Strategy: For Optimizing the videos before transmission, we used PANO, one of the recent 360^o video optimization strategies. It is different from the other viewport driven approaches as it focuses on perceived quality and optimizes the video based on its content. Their strategy is driven by three factors: (i) Velocity of moving object in the viewport, (ii) Depth of Field and (iii) change in brightness levels. Based on this, they modelled PSPNR, a quality measure for 360^o videos which are used in optimizing the videos. Furthermore, they have provided a novel tiling scheme and a quality adaptation model at the client.

Proposed Optimization for 360^o Teleconferencing

Our proposed theatre mode of teleconferencing application was built with PANO as the baseline optimization strategy. Since it is a teleconferencing application, we have further optimized PANO based on the following assumptions:

1) In Teleconferencing application, the motion of an object in the viewport (the person on another end), will be zero or constant in most part of the video. In this case, we can optimize PANO preprocessing by providing a constant velocity value of 10.f where f is the maximum of the velocity at which the user moves the viewport and the optical flow of the content of the video. This would take away the need for continuous tracking the moving object in viewport and calculating its velocity.
2) As per Liu et al [27], video teleconferencing can be done with camera positions in two ways. The camera can be placed either in the centre of the table with people around it or it can be placed at the end of the table with people sitting in front of it. In the first case, a velocity optimized 360^o video streaming would give an immersive teleconferencing experience. In the second case, 360^o video is not required but a 180^o viewport of the video where the people are in focus with optimized velocity is enough to cover the meeting area.

Based on these assumptions, we have optimized PANO in two stages to provide two outputs. The first output is 360^o videos with optimized velocity and second is a 180^o video with optimized velocity. They are saved on our local machine into a file-based database.

3) Containers: The RTSP server is developed in python which reads the output of PANO and it streams the videos to the client on demand. To streamline the process and improve the efficiency of the server, we have hosted our server on two local machines, one is running Windows 10 Pro operating system and another is running Ubuntu 18.01 LTS operating system. Both the server nodes can accept the request and respond on a real-time basis.

To deal with the challenge of running the same server instance of multiple OS environments, we have wrapped the server code with all the required files and dependencies into a docker container and deployed on these local machines.

To manage the load between both the server instance and improve the scalability by maximizing the utilization of processor cores of each server machine, we have used Kubernetes on top of docker containers. The Master kublet server is hosted on Windows 10 OS machine and minion kublet server is on Ubuntu OS machine. Thus, by setting proxies to these nodes, we improved the load handling and scalability of the server.

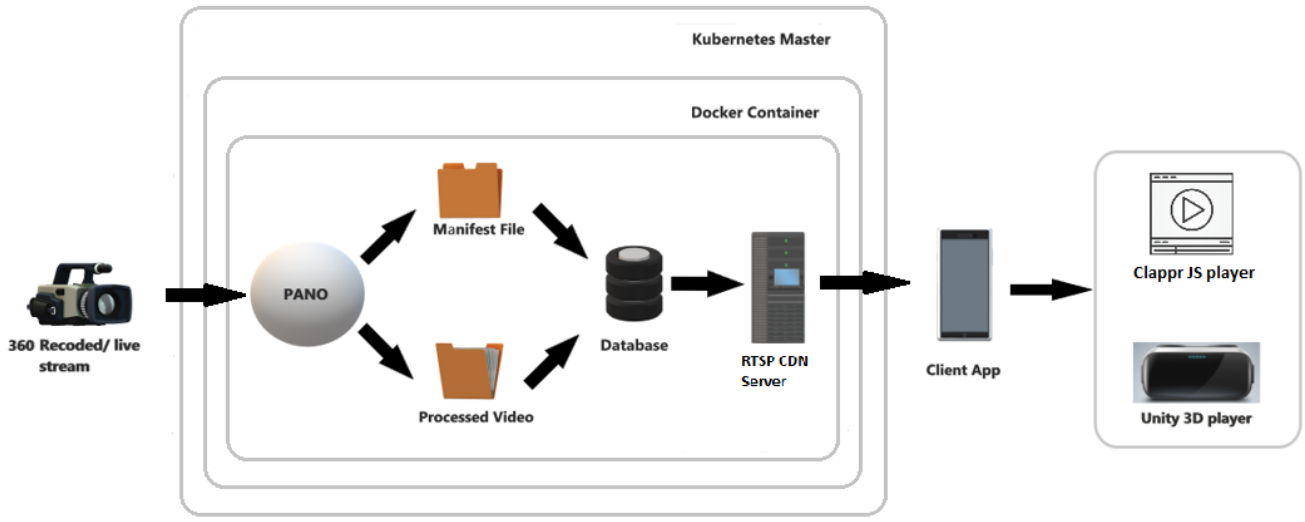

Fig. 1. Implementation pipeline of Theatre Mode of 360^o Video Teleconferencing Prototype

4) Teleconferencing Android Application: We have developed an Android based application based on Android SDK version 3.6.2. The application provides a unified view for the users to select the stereoscopic VR player and a monoscopic fullscreen player just with the touch of a button. The application is designed to establish a connection to server during the loading time of the app on the mobile phone. The app has the following functionalities:

- It establishes a TCP/IP based connection secured by SSL to the server.
- It requests the server the name of the video, the type in which the user wants to see the video, the option which the user selects to stream the video or download it, and then pause and terminate options which users requests during the video play.
- While the video is being played, the app reads the gyroscopic data collected from the mobile hardware and calculates the viewport location inside the video. Based on the viewport movements, it requests the server for the update in the quality of the video.
- The application also adjusts the initial video quality based on the manifest file sent by the server along with the video.
- The application is also capable of splitting the video screen into stereoscopic VR screen so that the video play is compatible with google cardboard. It performs the same play, pause, and terminate functionality in the VR mode after reading the selections and inputs made by head-mounted display's controllers.

5) Video Player for Head-Mounted Display: The VR player developed in the Unity3D is rendered as an Android file using Android development tools and embedded in the mobile application. Before the app loads the player, it splits the video file to be compatible with the player. The player uses VRTK and SteamVR to read the inputs from head-mounted display's controllers before passing them to the main application, which is used to adjust the resolution of the video. The video player can be accessed by clicking the "Cardboard" option after selecting your favourite video to play.

6) Video Player for full-screen monoscopic display: Our application allows the user to stream the video on a full-screen mobile display using video player by Clappr.js. The player has options for play, pause, volume control and full screen. It is embedded in the application using Android HTML viewer. This video player works well on the mobile application as well as web applications. The video player can be accessed by clicking the "Full-Screen" option after selecting your favourite video to play.

D. Application Workflow

The overall processes involved from opening our application to playing a video is given below. The screen captures of the different options available in our Android application is shown in "Fig. 2"

- The Client requests 360^o video-on-demand using our teleconferencing Android application. There are two options available, either to stream from server or download in local machine for playing the video.
- The client selects the type of video player for the 360^o video to be played from the two available options - full-screen monoscopic or VR stereoscopic . Client also selects the size of the video to be viewed, either 180^o or 360^o
- The server receives the request and creates an instance of server worker for the client.
- The video quality assessment and tiling component of PANO is performed inside the server for streaming the 180^o/360^o video, which includes our added optimization for teleconferencing.

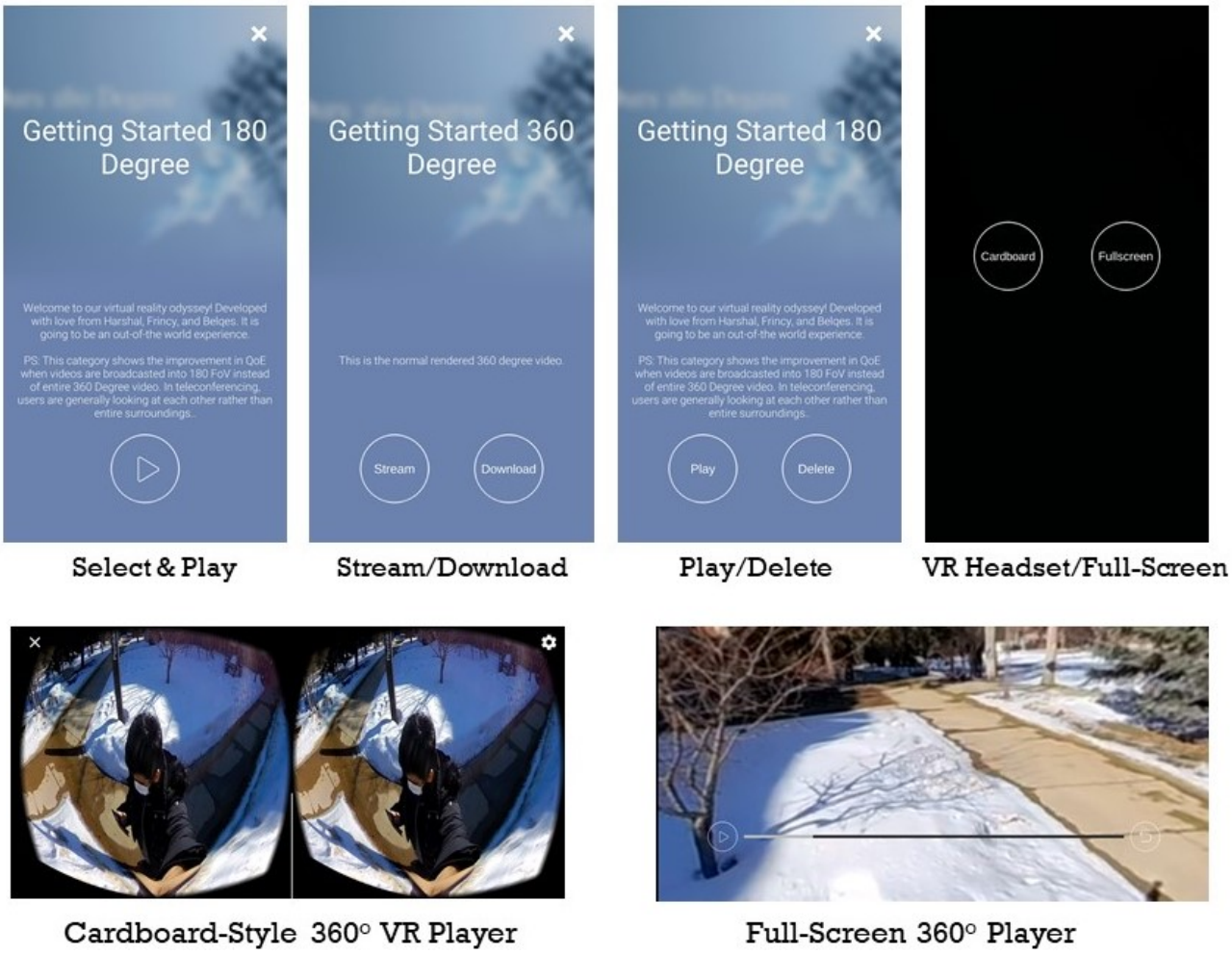

Fig. 2. Screen captures of the Android application prototype for 360^o teleconferencing

- The server starts streaming the video to the client using RTSP protocol after embedding the video in h.265 codec with CRF level 18.
- The streaming video is then rendered and played on the selected 360^o media player at client side in real-time.

E. Novelty

There are no applications currently available that can playback synchronized 360^o VR video in video calling fashion. The only development which is remotely close to virtual teleconferencing is implemented by Oculus in their meeting rooms, but it uses animated avatars to symbolize every user.

In our project, we implemented the stage-one which is multi-user "theatre mode" of the "360^o Teleconferencing Application" which can be easily extended to multi-user live 360^o video calling application. Furthermore, we optimized the baseline video streaming strategy, PANO, by considering two factors which are specific to the application of teleconferencing.

Furthermore, it is the first-ever in built-app to switch between Fullscreen 2D mode to 360^os stereoscopic mode. Its built-in player allows custom zooming functionality for better QoE. More importantly, It works on poor network conditions (under 10 MBPS network bandwidth availability) while maintaining high quality.

VI. Evaluation

A. Methodology

To evaluate our proposed application, we chose two sample 360^o videos, a self-shot at University of Alberta using a 360^o camera and another downloaded from vimeo [26]. To evaluate the selected metrics, we used several tools such as Android built-in tools, Gamebench [30], FFmpeg, custom codes in player and PANO source code for evaluation. We performed comparison across original version, baseline PANO output without our optimization, PANO optimized for 360^o and PANO optimized for 180^o.

B. Metrics

To compare the results of our optimization against our baseline streaming strategy PANO, we have selected few metrics. Some of the metrics are same as those used to evaluate

PANO by its authors and the remaining metrics are those normally used for assessing video streaming strategies. The initial evaluation was performed using two metrics which compares the most important factors perceived quality using PSPNR and bandwidth consumption (Mbps). We also assessed the video streaming quality using frame per second (FPS), standard deviation from FPS, latency(secs) and pre-processing time (secs).

C. Results and Inferences

1) Perceived Quality: PSPNR: It is an improvised version of traditional quality metric PSNR, introduced in the baseline strategy, PANO. We compared the baseline PANO output with our optimized 360^o and 180^o versions at different levels of noises. The graphs were traced using the evaluation codes available in PANO source code. The results as seen on "Fig. 3" shows that both of our optimized versions have better perceived quality (higher PSPNR) than the baseline PANO output. Among the two optimized versions, 180^o had better perceived quality than 360^o which is clearly visible when playing them in our mobile application as well.

2) Maximum Bandwidth Consumed: One of the major challenges faced by 360^o video streaming is the amount of bandwidth required to stream the video without reducing the perceived quality. Hence, using Android built-in tools (Gamebench [30]) we traced the maximum bandwidth consumed for all the versions of two sample videos. "Fig. 4" shows that our optimized versions of PANO performs much better than baseline PANO version as well as original 360^o version. PANO optimized 180^o version saves 38% bandwidth compared to original 360^o version and 20-46% compared to PANO baseline version. PANO optimized 360^o version saves 23-26% bandwidth compared to original version and upto 33% compared to PANO baseline version.

When all PANO versions are compared on PSPNR against bandwidth consumption as seen on "Fig. 5", our PANO optimized 180^o version outperforms the others by a huge margin followed by our PANO optimized 360^o version. This means, both of our optimized versions, especially 180^o require much lesser bandwidth to stream the videos at higher perceived quality compared to the baseline PANO version.

3) Latency: Latency in video streaming is measured by calculating the delay in displaying the first frame of the video, using Android built-in clock. Both of our optimized versions have lesser latency compared to that of PANO versions as seen on "Fig. 6" (left). The optimized 180^o version loads much faster than other two versions and optimized 360 version loads faster than the baseline PANO version.

4) PANO Pre-processing Time: To understand the additional overhead required for PANO to pre-process the videos, we calculated the time required for the same using Android built in tools. This includes the time required for PANO to perform tiling, assessing the quality, and push it out for streaming. The results in "Fig. 6" (right) shows that pre-processing time is slightly higher for our optimized versions compared to PANO original version. However, our optimized 180^o version takes 39% less time for video 1 (duration 57 sec) and 69% less time for video 2 (duration 2 mins) for pre-processing compared to the optimized 360^o version. Hence, when we do not need the full 360^o, especially in the application of teleconferencing, we could choose 180^o because of its superior benefits, as we have that option available in our application.

5) Average Frame Rate: The pattern of frame rate throughout the playback time, which could be measured using its standard deviation, gives an understanding of video streaming quality. We compared all versions of videos on average frame rate which is measured as frames per second (FPS) using custom codes in player and traced using Gamebench. As seen on "Fig. 7" (left) 180^o optimized version seem to have the highest FPS, even better than the original 360^o video. Our optimized 360^o version also has better average frame rate than the baseline PANO version.

When compared on standard deviation of frame rate, the PANO baseline version seemed to have more variations in their frame rate, compared to any others, as seen on "Fig. 7" (right). Lesser standard deviation would mean better video streaming quality. Our optimized 180^o version performs equivalent to the original video. The standard deviation presented by our optimized 360^o video is not significantly high to produce reduction in video streaming quality. It is worth using our optimized versions over original video or the baseline PANO version, as it provides huge savings in bandwidth while providing better perceived quality.

D. Summary of Results

To evaluate our proposed optimization, we compared them with the baseline PANO version and original video, across multiple metrics which gives insights into the video streaming quality. The highlight of the evaluation result is that our optimized 180^o version stands out from the rest whether it be bandwidth savings, perceived quality, latency or standard deviation in frame rate. Our optimized 360^o version performs much better than PANO baseline version for the same as well. When it comes to bandwidth , PANO optimized 180^o version saves 38% compared to the original video and around 20-46% compared to the PANO baseline version. Our optimized 360^o version saves 23-26% bandwidth compared to original video and upto 33% compared to PANO baseline version. This has been achieved while maintaining higher PSPNR values compared to the baseline PANO version.

VII. Project Contributions

We have ventured to innovate the video conferencing experience of all the users who are dependent on video conferencing for their personal or professional affairs. Our application can be used on the day-to-day scenarios across a variety of domains in all the network conditions and gaining a more immersive experience of video conferencing.

Our primary contribution is in the video streaming strategy for 360- degree videos. We have optimized the strategy to function on lower bandwidth. The content creator and streaming platforms is a huge platform in itself and content

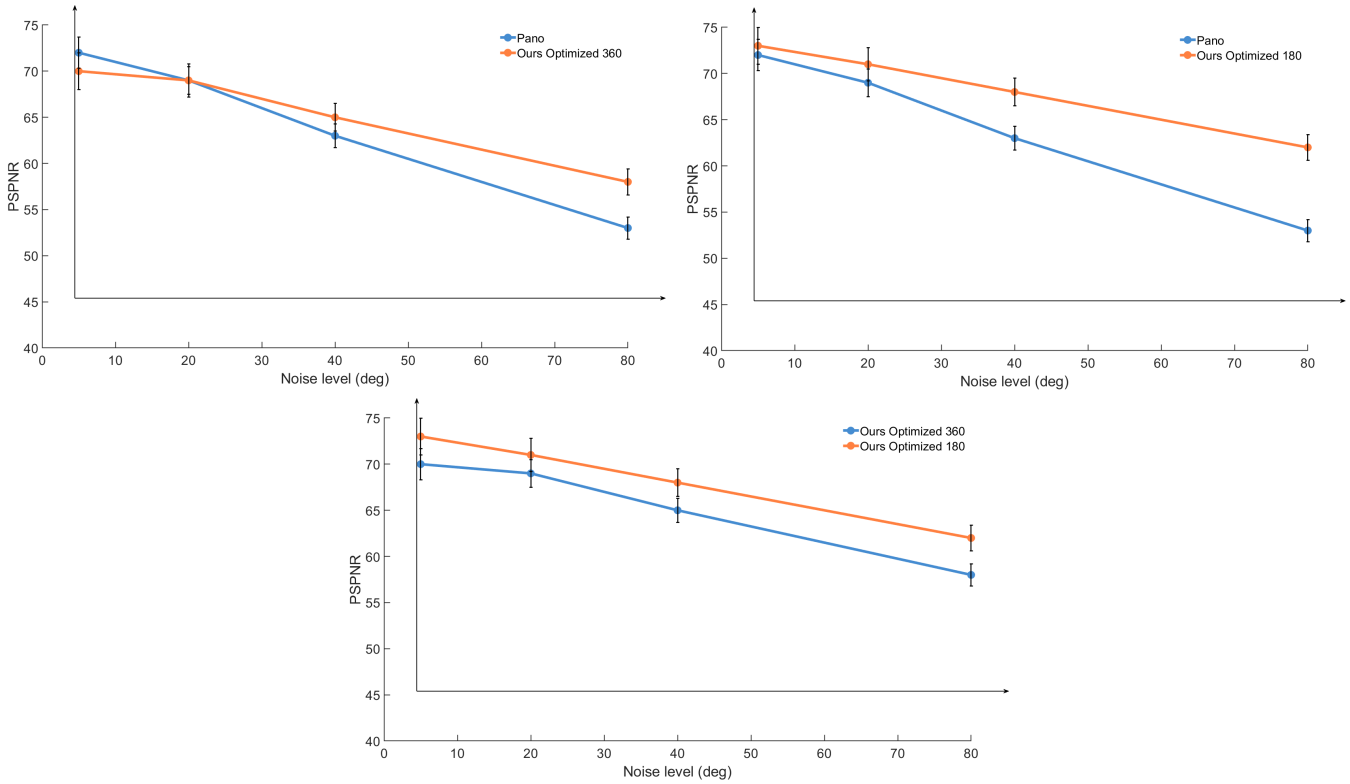

Fig. 3. PSPNR at different noise levels for (1) PANO vs Our Optimized PANO 360^o (Top-left), (2) PANO vs Our Optimized PANO 180^o (Top-right) and (3) PANO Optimized 360^o Vs 180^o (Bottom)

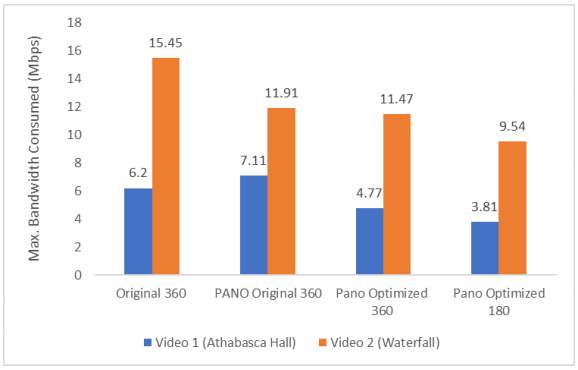

Fig. 4. Comparison of Maximum Bandwidth consumption across 4 versions of video

developers can upload or live stream their videos using the app without worrying about the format of video or display.

Our application opens a brand new area of Virtual reality video conference which provides a sense of realism and immersion. Using our application, an astronaut at International Space Station can video call her/his family on earth and experience the homely feeling by streaming content to any head-mounted display using our application. Likewise, there are endless applications and areas, our application can revamp the manner of our multimedia content streaming in 3D.

VIII. Project Status

Our proposed method for the application of 360^o teleconferencing consists of two stages: (i) Theatre mode (ii) Live-Mode. We have successfully completed the first stage i.e. theatre mode in this project as expected with the current project scope and duration in mind. The theatre mode can be extended to a live video conferencing by adding two-way communication (both audio and video) between two users.

Due to current unprecedented situation, we were not able to procure 360^o Streaming camera before the deadline and test our streaming live-mode but testing the streaming using pre-downloaded videos indicated the capability of our application for the same.

IX. Conclusion and Future Work

360^o Video conferencing is an untapped opportunity, which is the reason for us to endeavour into this project. Using one of the recent 360^o video streaming strategy, PANO as baseline, we developed a mobile application to experience 360^o multimedia content. Our application supports video conferencing as well as theatre-like video support. Combining both the VR platform as well as monoscopic stereo screens, our app provides larger flexibility than any of the pre-existing application for 360^o. Furthermore, we added two optimizations in video streaming, specifically in the application of teleconferencing, which provides bandwidth savings upto 38% compared the original videos and upto 46% compared to PANO baseline

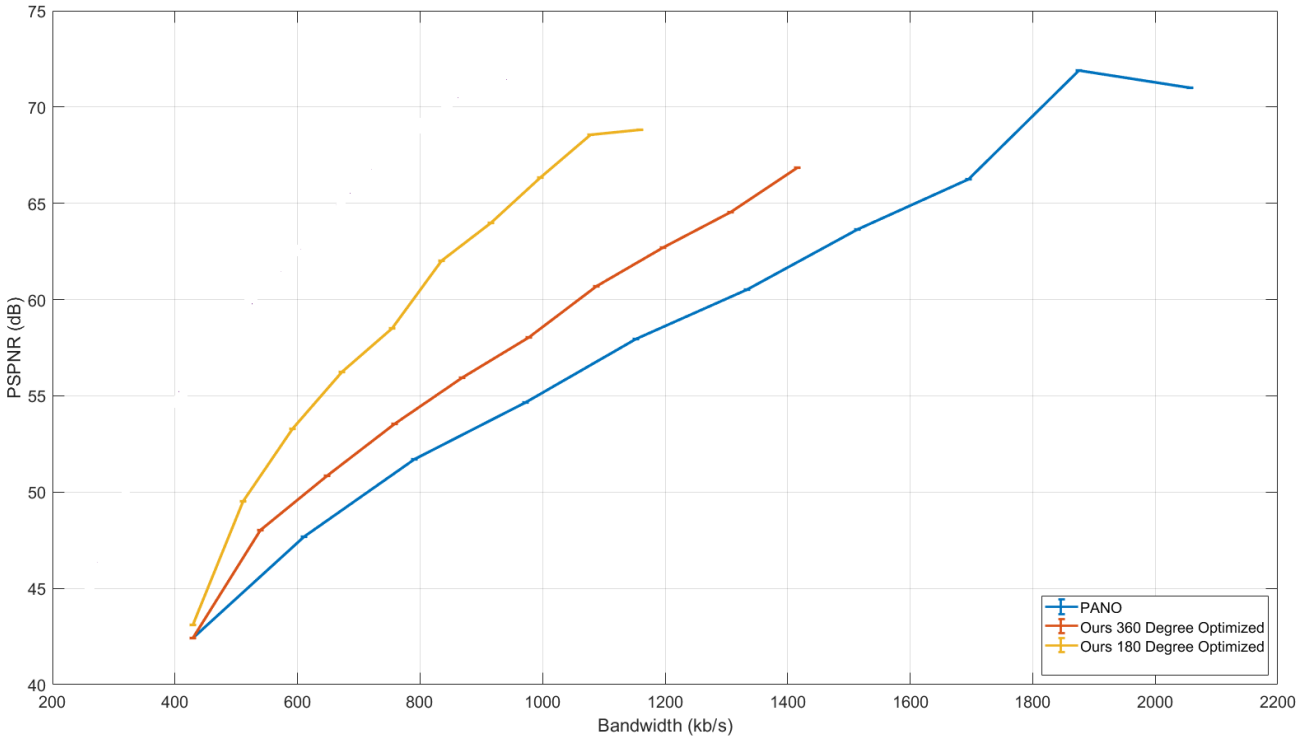

Fig. 5. Comparison of PANO versions on PSPNR against Bandwidth consumption (Kbps)

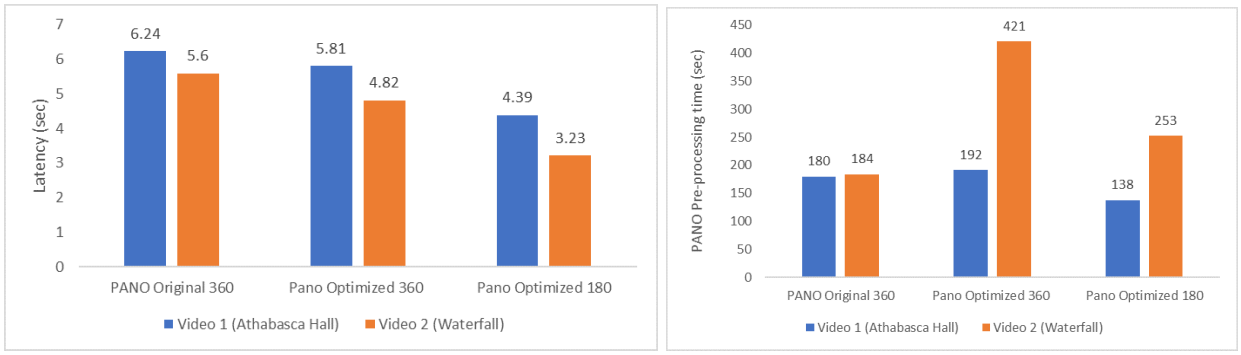

Fig. 6. Left: Comparison of Latency across 3 PANO versions; Right: Comparison of pre-processing time taken by PANO across 3 PANO versions

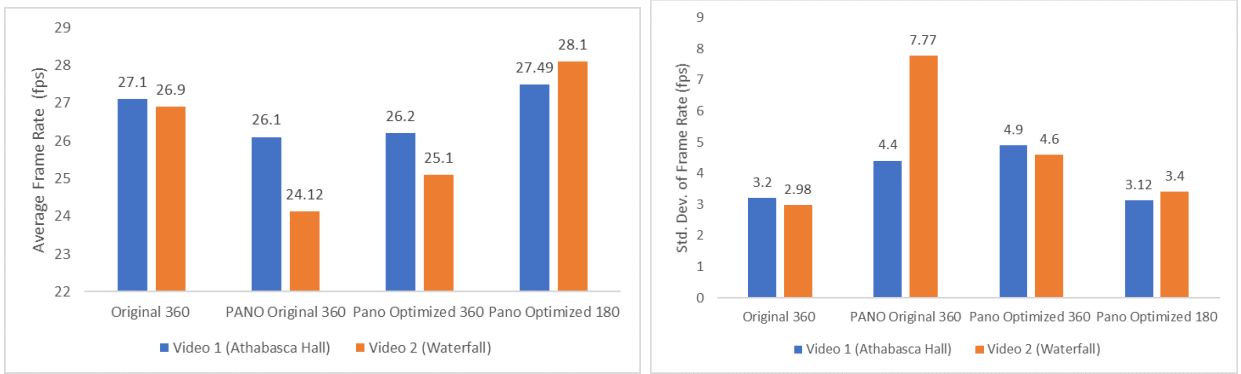

Fig. 7. Left: Comparison of Average Frame Rate across all 4 versions of video; Right: Comparison of Standard Deviation of Frame Rate across all 4 versions of video

version. Our optimized versions can work on lesser bandwidth while maintaining higher perceived quality defined in terms of PSPNR.

For our future work, we plan to procure multiple 360^o cameras and update our application to connect the camera and stream the content directly from the camera over the network. We will invest our time to further optimize the performance of our streaming strategy in order to make it more robust to varying upload and download speeds. Also, we aim to extend the video support up to 8K stereoscopic 360^o videos and reduce the pre-processing time of the same to reduce server-side processing time.

Team Roles and Responsibilities

The roles and responsibilities for the projects were divided as follows:

Roles
Harshal • System architecture & design • Developed Stereoscopic Player • App development - video player calibration and hardware interaction • Adaptive playback support (client-side) • PANO configuration - Kubernetes nodes • PANO optimization - 180-degree videos + constant velocity • Setting up and hosting servers • Port forwarding • Client-server integration • Report writing, Presentation slides and Demo
Princy • Workflow management • Developed Full-screen player • App development - client and server connection • Adaptive playback support (server-side) • PANO configuration - docker containers • PANO optimization - 360-degree videos + constant velocity • Setup (.yaml)configurations files for the server • Client-server integration • Evaluating the performance • Report writing and Presentation slides
Belqes • Literature review and testing • Developed Full-screen player • App Development- GUI design • Video pre-processing and tweaking • PANO optimization - original videos • Testing the reliability of the server • Evaluating the performance • Report writing and Presentation slides

Acknowledgment

We would like to thank Dr. Mohammed Elmorsy, Department of Computing Science, University of Alberta in guiding us throughout the project and the authors of PANO video optimization strategy which became the baseline of our application.

X. Resources

- Source Code: https://bit.ly/3e0AgP9
- Input Dataset and our optimized PANO versions: https://bit.ly/3dZuddG
- Presentation and Demo: https://youtu.be/2TbuDQvJq-U

References

[1] Guan, Yu, Chengyuan Zheng, Xinggong Zhang, Zongming Guo, and Junchen Jiang. "Pano: Optimizing 360 video streaming with a better understanding of quality perception." In Proceedings of the ACM Special Interest Group on Data Communication, pp. 394-407. 2019.

[2] "Production-Grade Container Orchestration." Kubernetes. Accessed April 5, 2020. https://kubernetes.io/.

[3] "Empowering App Development for Developers." Docker. Accessed April 5, 2020. https://www.docker.com/.

[4] Sreedhar, Kashyap Kammachi, Alireza Aminlou, Miska M. Hannuksela, and Moncef Gabbouj. "Viewport-adaptive encoding and streaming of 360-degree video for virtual reality applications." In 2016 IEEE International Symposium on Multimedia (ISM), pp. 583-586. IEEE, 2016.

[5] Bao, Yanan, Huasen Wu, Tianxiao Zhang, Albara Ah Ramli, and Xin Liu. "Shooting a moving target: Motion-prediction-based transmission for 360-degree videos." In 2016 IEEE International Conference on Big Data (Big Data), pp. 1161-1170. IEEE, 2016.

[6] Petrangeli, Stefano, Viswanathan Swaminathan, Mohammad Hosseini, and Filip De Turck. "An http/2-based adaptive streaming framework for 360 virtual reality videos." In Proceedings of the 25th ACM international conference on Multimedia, pp. 306-314. 2017.

[7] Fan, Ching-Ling, Jean Lee, Wen-Chih Lo, Chun-Ying Huang, Kuan-Ta Chen, and Cheng-Hsin Hsu. "Fixation prediction for 360 video streaming in head-mounted virtual reality." In Proceedings of the 27th Workshop on Network and Operating Systems Support for Digital Audio and Video, pp. 67-72. 2017.

[8] Corbillon, Xavier, Gwendal Simon, Alisa Devlic, and Jacob Chakareski. "Viewport-adaptive navigable 360-degree video delivery." In 2017 IEEE international conference on communications (ICC), pp. 1-7. IEEE, 2017.

[9] Zhou, Chao, Mengbai Xiao, and Yao Liu. "Clustile: Toward minimizing bandwidth in 360-degree video streaming." In IEEE INFOCOM 2018-IEEE Conference on Computer Communications, pp. 962-970. IEEE, 2018.

[10] Qian, Feng, Bo Han, Qingyang Xiao, and Vijay Gopalakrishnan. "Flare: Practical viewport-adaptive 360-degree video streaming for mobile devices." In Proceedings of the 24th Annual International Conference on Mobile Computing and Networking, pp. 99-114. 2018.

[11] "Real Time Streaming Protocol." Wikipedia. Wikimedia Foundation, February 9, 2020. https://en.wikipedia.org/wiki/Real_Time_Streaming_Protocol.

[12] "Real-Time Transport Protocol." Wikipedia. Wikimedia Foundation, March 25, 2020. https://en.wikipedia.org/wiki/Real-time_Transport_Protocol.

[13] Rouse, Margaret. "What Is Real Time Streaming Protocol (RTSP)? ." SearchVirtualDesktop. TechTarget, March 12, 2018. https://searchvirtualdesktop.techtarget.com/definition/Real-Time-Streaming-Protocol-RTSP.

[14] Clappr. "Clappr/Clappr." GitHub, December 3, 2019. https://github.com/clappr/clappr.

[15] Thiagopnts. "Thiagopnts/Clappr-video360." GitHub, December 27, 2019. https://github.com/thiagopnts/clappr-video360.

[16] Technologies, Unity. "Unity." Unity. Accessed April 5, 2020. https://unity.com/.

[17] "VRTK - Virtual Reality Toolkit - [VR Toolkit]: Integration: Unity Asset Store." Integration — Unity Asset Store. Accessed April 5, 2020. https://assetstore.unity.com/packages/tools/integration/vrtk-virtual-reality-toolkit-vr-toolkit-64131.

[18] “SteamVR Plugin: Integration: Unity Asset Store.” Integration — Unity Asset Store. Accessed April 5, 2020. https://assetstore.unity.com/packages/tools/integration/steamvr-plugin-32647.
[19] “16.3. Thread - Multiple Threads of Control.” 16.3. thread - Multiple threads of control - Python 2.7.17 documentation. Accessed April 5, 2020. https://docs.python.org/2/library/thread.html.
[20] “Queue - A Synchronized Queue Class.” queue - A synchronized queue class - Python 3.8.2 documentation. Accessed April 5, 2020. https://docs.python.org/3/library/queue.html.
[21] “FFmpeg.” FFmpeg. Accessed April 5, 2020. https://www.ffmpeg.org/.
[22] Ytdl-Org. “Ytdl-Org/Youtube-Dl.” GitHub, April 5, 2020. https://github.com/ytdl-org/youtube-dl.
[23] “Download Android Studio and SDK Tools : Android Developers.” Android Developers. Accessed April 5, 2020. https://developer.android.com/studio.
[24] Allensarkisyan. “Allensarkisyan/VideoFrame.” GitHub, October 9, 2014. https://github.com/allensarkisyan/VideoFrame.
[25] Jong, Jos de. “Math.js.” math.js. Accessed April 5, 2020. https://mathjs.org/.
[26] vijay_sharathy. “Sample Input 360 Degree Video - Waterfall.” Vimeo, March 31, 2020. https://vimeo.com/257105938.
[27] Liu, Zicheng, Michael Cohen, Deepti Bhatnagar, Ross Cutler, and Zhengyou Zhang. ”Head-size equalization for improved visual perception in video conferencing.” IEEE Transactions on Multimedia 9, no. 7 (2007): 1520-1527.
[28] “High Efficiency Video Coding.” Wikipedia. Wikimedia Foundation, March 1, 2020. https://en.wikipedia.org/wiki/High_Efficiency_Video_Coding.
[29] “Introduction to Headjack.” Headjack. Accessed April 6, 2020. https://headjack.io/docs/.
[30] Staff, GameBench. “Measuring Frame Rates In Android And IOS Games.” GameBench. Accessed April 6, 2020. https://blog.gamebench.net/measuring-frame-rates-in-android-and-ios-games.