mm-Wave WiFi: Frame-By-Frame Encoded VR Application Layer Performance

Pieter Hendriks
Department of Computer Science
University of Antwerp
Antwerp, Belgium
pieter.hendriks@student.uantwerpen.be

***Abstract*—We present an analysis of mm-Wave WiFi network performance, between two TP-Link AD7200 Talon routers, controlling for distance, movement and a human body as obstacle between them. These factors are the most relevant complicating factors for a VR use-case. We implement a frame-by-frame encoded video transmitter to benchmark the system's performance, with a built-in NTP clock synchronization. We enable router mobility by mounting it on top of a robot, for which we implement a controller emulation layer to allow pre-programmed paths to be followed rather than requiring manual control. We evaluate the received video with respect to packet loss (relative to frame display time - late packets are dropped), frame latency, image quality and video freezes. Our results show that the network layer is insufficient to achieve optimal performance in this application - but given that it uses frame-by-frame encoding, rather than a video encoding, that is not surprising. Additionally, we show that the network tends to maintain a reasonable level of performance when only one of the complicating factors is present, but that performance quickly drops to unacceptable levels once two (and especially three) of the disruptive factors we controlled for are introduced to the experiments.**

I. Introduction

Over the past few years, following the release of the Oculus Rift, virtual reality (VR) content has been becoming more and more popular. Generally, the content is displayed to the user using a head-mounted display (HMD). Some are standalone devices (e.g. Oculus Quest 2), others are connected to a computer, which renders the frames and then transmits them (e.g. Oculus Rift). In this work, we are most interested in the connected devices. Most of the devices currently on the market are tethered (especially high quality ones); they use a multi-gigabit HDMI cable to transmit data between the computer and the HMD. One notable exception is the HTC Vive, for which a wireless adapter is available as a separate purchase. The cable connecting the user (wearing the HMD) to the computer creates three problems:

- The user's range of movement is limited
- Touching the cable breaks immersion
- Tripping hazard

The obvious solution, to avoid these problems, is to transmit the VR content from the rendering PC to the HMD wirelessly. While a wireless signal will still limit the user's movement (to within the range of the wireless network), this will generally be less restrictive than a cable. The other two problems are avoided entirely.

One of the benefits a VR system offers over more traditional display panels is improved immersion. Due to the head-mounted display, the user only sees the virtual world. With traditional displays, the virtual world is only a part of what the user sees. The virtual world covering the user's entire visual field makes the experience more immersive - i.e. the user will feel more like they are the character they're controlling.

VR content displayed in this way has stringent data rate and latency requirements, as even slight hiccups in frame display can be unpleasant for the user and break immersion. In most cases, the target frame-rate (as specified by the device manufacturers) for these HMDs is 90 frames per second (fps) or more, as those frame-rates have been found to mostly eliminate motion sickness symptoms that have been reported as a result of the use of VR systems [1]. If the frame-rate frequently drops below this value (or occasionally far below this value), the user's experience (UX) will be negatively affected.

Similarly, the latency experienced by the user must be constrained. This is especially important interactive VR systems. These are systems where the user performs actions that cause effects in the virtual world (for example, pressing a button on a controller fires a weapon in a video game). If the latency between the user's action and the result being displayed is high, the system will feel unresponsive and unpleasant to use, disrupting immersion. The range of acceptable delay between user's action and display of the result is found to be between 7 and 20 milliseconds, depending on the user [2].

When a user performs an action in an interactive VR system, it is important that the result of this action is shown to the user in a timely manner (a delay of around 7-20ms is acceptable, depending on the user) [2]. Non-responsive controls break the user's immersion, which is undesirable.

mm-Wave WiFi's advantages nearly exactly coincide with the requirements for wireless VR video transmission, which makes it seem like an ideal candidate transmission medium. It comes with a few downsides. Notably, signals transmitted over these frequencies (mm-Wave spans roughly from 30 GHz to 300 GHz) experience a much higher propagation loss than what is typical in traditionally used wireless networks (below 6 GHz, commonly 2.4 GHz and 5 GHz). So much so, in fact, that a brick, windowless, wall between two devices could fully prevent them from communicating [3]. Even when in line of

sight, a distance of roughly 10 meters is typically sufficient for the transmission to experience significant performance drop-off [4].

In this work, we analyze the application layer performance of IEEE 802.11ad millimeter wave (mm-Wave) WiFi as a transmission medium for a VR video stream. mm-Wave WiFi utilizes beam forming, which is a technique to create a highly directional signal, to improve the signal strength at the receiver. In our work, we use TP-Link AD7200 Talon routers [5]. They implement a sector-based beam forming, where each sector covers a range of directions. Whenever the current sector is no longer sufficient, a new sector is selected through a sector sweep [6]. Due to the directionality, movement of one (or both) of the antennas in a mm-wave WiFi network significantly degrades the performance [7], by causing the beams to become misaligned. This results in a significant decrease in signal-to-noise ratio (SNR) and an immediate associated drop in network throughput, and often also the selection of a new sector.

We present an application that encodes a video into separate frames and transmits them, via a mm-Wave WiFi link, to a receiver where the video can be reconstructed. We study the impact mobility and obstacles have on the quality of this video transmission, with a focus on application layer performance (i.e. visual analysis of the images and frame latency/freezing). To enable consistent, reproducible mobility, we present modifications made to the open source control system developed by Rover Robotics [8], which allows the robot to be driven using a Bluetooth controller. We then run experiments in various scenario's; we start with a baseline evaluation case, where both routers are static and within line of sight. We further consider linear motion (in the assumed direction of the beam - i.e. along the line defined by the two routers' midpoints), circular motion (with one router being the circle's center) and the human body as an obstacle (no other obstacles tend to be between transmitter and receiver, for safety reasons). We conclude with a comparison of the results we saw in each of these situations.

II. Related work

Utilizing mm-Wave to get rid of the cable connecting the HMD to the PC is definitely not a new idea, it has been considered in the literature. IEEE 802.11ad is a good fit for the transmission of latency-sensitive VR data. Struye *et al.* introduce a theoretical framework for the computation of attainable VR datarates, for different IEEE 802.11ad channel access settings [2]. They validate their framework using the ns-3 simulator, showing that in an ideal case, a single AP can support 8 VR headsets receiving 4k video feeds.

Most work considering mm-Wave WiFi focuses on a single link at a time, even if up to 8 VR HMDs can be supported by an AP at a time. This work considers the impact of up to 8 connected devices in a dense mm-Wave environment [9]. Their testbed allows for control over lower-layer parameters to quantify their impact.

Abari *et al.* present MoVR, a technology to avoid blockage issues [7]. They use (at least one, potentially more) mirrors to reflect the transmitted signal around blockages, using the HMD's built-in location and orientation tracking.

coVRage [10] is a beam-forming solution tailored to VR HMDs, it predicts the change in the angle of arrival (AoA) for the video signal based on the current and past positions of the HMD. Using this prediction, it is capable of covering the AoA with a dynamically shaped beam resulting in improved performance.

A programmable HMD, allowing extensive modifications in software to optimise the performance, is introduced by Zhong *et al.* [11]. Using their newly introduced programmable HMD, they investigate methods to optimize rendering. Further studies exploring these optimizations have been done [12] [13].

Using mm-Wave to enable VR applications is actively being studied in the context of 5G networks as well [14]. They discuss the challenges faced and a number of proposed solutions for low latency, reliable VR over mm-Wave WiFi.

To deal with other traffic being transmitted over the network (e.g. controller inputs), multi-beam multiple access techniques are proposed for uplink traffic in VR applications [15].

COTS hardware tends to be designed to save manufacturing costs, where possible. This results in mm-Wave routers with highly irregular beam shapes, which requires special consideration when dealing with these routers [16].

The impact of the beam steering algorithm on the performance of the mm-Wave WiFi network is significant. COTS devices tend to use sector-based approaches, while more recent advancements offer much better performance [17]. These, however, have not yet been integrated into over-the-shelf devices. This work adapts compressive path tracking for sector selection in COTS devices.

III. Methodology

We attempt to emulate a typical VR use-case in our experiments, so that our results reflect (as closely as feasible) a real-world scenario. We employ a static device and a mobile device, as is typical in VR use-cases, where the static device transmits a video stream to the mobile device. On the mobile device, we record the arrived data and reconstruct the frames composing the video stream. These reconstructed frames can then be compared to the originals to visually compare the source to the displayed video.

A. Network

The network is set up as shown in figure 1. We have two end-point devices, each connected to one router. Between the routers, there is a wireless 60GHz WiFi link. One of the routers is static, the other is mobile and we will refer to them as such throughout the paper. The static router vertically located approximately 10 centimeters below the mobile router.

Both the mobile and the static router are TP-Link AD7200 Talon routers [5]. The static router is configured as a wireless access point, the mobile one is connected to that network. Both routers are configured to route traffic received through

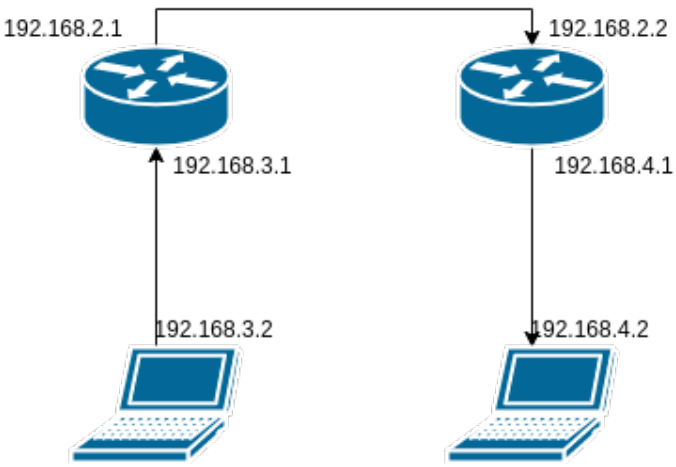

Fig. 1. Schematic of the network

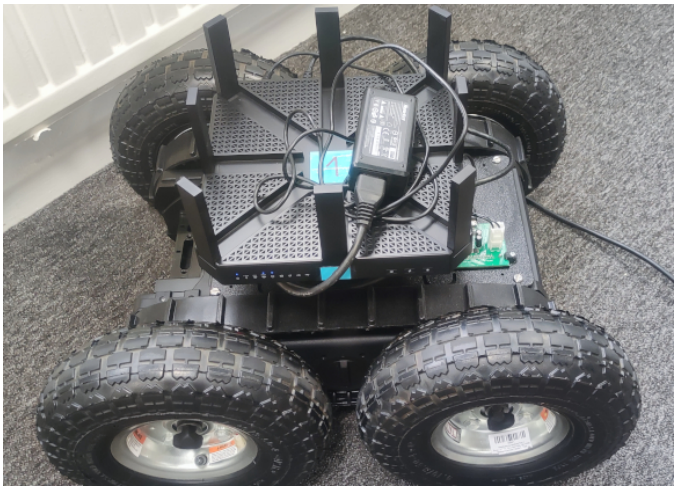

Fig. 2. The TP-Link AD7200 Talon mounted on top of the Rover Robotics Pro

the ethernet interface over the wireless network and vice versa, creating a connection between the two endpoints in both directions. The static endpoint will then transmit video frames to the mobile endpoint.

B. Robot

To facilitate mobility within our experiments, one of the routers is mounted on top of a robot, which will drive around to simulate how a user might move around when using a VR system. The robot we are using is a Rover Robotics Pro, in 4-wheel drive configuration [18].

The flat upper part of the robot provided an ideal surface to mount the moving router, as shown in figure 2, using Velcro. The robot is equipped with a small form factor mini PC [19]. We connect the router to this robot-mounted PC and use it as the receiver for our application.

The robot uses the Robot Operating System (ROS). The input, out of the box, is done using a Bluetooth controller. Due to experiment reproducibility concerns, we decided against using this input method. Instead, we implemented a controller emulation layer, allowing the robot to follow a pre-programmed path. This ensures that any experiments we perform can be repeated as desired.

Controller value	**Measured speed** (m/s)
0.1	0.15
0.2	0.4
0.3	0.7
0.4	1.1
0.5	1.5
0.6	2
0.7	2.2

TABLE I
TABLE FOR CONVERSIONS OF ROBOT SPEED TO METERS PER SECOND. THESE VALUES ARE INEXACT, AS THEY ARE THE RESULT OF MANUAL MEASUREMENT.

The implementation of the controller emulator is based on the software stack that is provided by Rover Robotics [8] [20], we replace the section of code handling the controller input with a simple fake controller. The default implementation uses the left joystick for forward and backward movement, the right joystick for turning left and right. The joystick positions are represented using a number between -1 and 1, the speed to move/turn at is determined based on this value.

The fake controller is configured with a certain path, which is made up of a series of checkpoints. These checkpoints are a triplet: (relative time stamp (relative to ROS node start time), linear speed, turning speed). The time stamp is a positive floating point number, in seconds. The linear speed and turning speed are both floating point numbers in the range [-1, 1], to mimic the controller's implementation. These values are then regularly published to the relevant ROS topics. Other parts of the system subscribe to the topics, to retrieve the values, and move the robot as directed. These parts of the system have remained wholly untouched.

Due to the unconventional speed specification used by our control system, there is no simple conversion to meters per second available. We performed some experiments and measured the time it took for the robot to travel a certain distance. However, since these measurements were done by manually starting and stopping a stopwatch, some error is expected to be present in these numbers. They serve as an indication for the reader, we use the values published by the controller emulator to denote the robot's movements in the paper.

C. Application layer

To study the performance of a network on the application layer, one first requires an application layer. We developed a sample application that transmits a frame-by-frame JPEG encoded 360° video stream. The application, additionally, performs time synchronization between the devices, in order to allow accurate reporting of performance statistics.

1) Time Synchronization: Both endpoints of the network are synchronize their clocks to an internet server. This synchronization ensures that the initial clock values are close together.

However, internet synchronization is insufficient to obtain the precision we need.

To improve upon this, we perform an additional time synchronization step to obtain relative offsets between the two devices. This time synchronization is implemented using the Network Time Protocol (NTP) [21]. Alternatively, Precision Time Protocol (PTP) [22] was considered as an alternative, but over a local network the accuracy obtained through NTP proved sufficient. To draw meaningful conclusions from our data, we require a measure of how far apart the clocks are, since the clock measurements are taken on separate devices. Without a measure on the relative drift, these are impossible to compare.

2) Video Stream: Each of our experiments uses the same 8k (7680x3840) 360° source video. We pre-processed the video, to extract each of the frames as individual images, creating a frame-by-frame JPEG encoding. These JPEG images are then transmitted as raw binary data, in segments of (up to) 1280 bytes (last segment often contains less data). Each segment contains an additional 20 bytes of data that is used on the receiver's end to correctly label each segment so we can perform accurate processing of the results. This adds up to 1300 bytes, which is comfortably lower than the 1500 byte path MTU in our network setup (the MTU for each ethernet connection is 1500 bytes), leaving sufficient room for headers.

Frame transmission happens at a constant rate of 10 frames per second (frame-time = 100ms). With a frame size of (on average) a little over 5 megabytes, that translates to approximately 400 megabits per second. The frame rate we're using is significantly lower than the value we mentioned before for an optimal experience (at least 90 fps). We're using an 8k video stream, which at 90 fps in this frame-by-frame encoding would translate to roughly 3.6 gigabits per second, which is comfortably below the Talon's advertised 4.6 gigabit per second data rate (over the 60 GHz band) [5]. However, part of our network are ethernet connections, where the speed is limited at a far lower value. Testing revealed that a frame rate of around 10 fps was the highest we could go where the baseline evaluations did not drop significant amounts of data. At this point, we'd like to note that frame-by-frame streaming significantly increases the data requirements for the network. In our case, the 8k source video (mp4) contained 4410 frames (at 30 fps), totaling a file size of 1.0 GB (1007358614 bytes). After frame-by-frame encoding (JPEG), the size of the folder containing all 4410 frames is approximately 23 GB. An over 20-fold increase in size, means the data rate used to transmit 10 fps in this application could instead be used to transmit 70 fps of a video encoding. This is much closer to an acceptable value and, most likely, in case of applications with limited high-speed movement, a threshold of acceptable performance is already reached.

The Talon's 60 GHz interface is advertised to be capable of data rates up to 4600 Mbps [5]. We do not get anywhere close to these speeds; we're off by approximately factor 10. This can be, in part, due to the ethernet connection used between the routers and the devices on each end. However, for gigabit ethernet, a data rate of 400 Mbps is also on the low end - we would expect something closer to 800 Mbps.

The video stream is sent over a UDP channel, we decided on UDP over TCP because when transmitting data that is immediately displayed, the delay incurred through re-transmissions could be worse for the UX than simply missing some data. Any missing data would only have an effect on the single frame that is missing data - requiring re-transmissions of missed data would impact many frames. This is particularly relevant in the case of interactive VR; when a TCP stream misses data, it must be re-transmitted and the transmission resumes where it left off. This causes a delay to accumulate, increasing whenever a re-transmission occurs, causing the user's experience to be ever more out of sync with the virtual world. In case of interactive VR, this is clearly undesirable. An implementation could improve upon this by discarding packets that will no longer be relevant (e.g. discard all re-transmissions if a new frame has been rendered) but implementing this is non-trivial and has not been considered in this work.

Similar statements hold for Web Real Time Communication (WebRTC) and QUIC, which are layers over UDP that provide improved performance while including some measure of reliability. WebRTC is used by e.g. Google Stadia as a technology for their video transmissions. The video data in Google Stadia is transmitted as a video file (without frame-by-frame encoding), this drastically reduces the data requirements by allowing the use of temporal information to further reduce file size (for example, transmit a key frame every 5 seconds, for all other frames, only transmit differences). They further improve the performance through the use of a frame-by-frame feedback loop, allowing dynamic downscaling of video quality when the quality of the network connection is poor. Stadia performs relatively well in wired environments, but the bursty losses that tend to occur in wireless networks are problematic and cause a severe degradation of the video quality [23].

UDP is an unreliable protocol, which means we have to deal with missing datagrams. Simply ignoring them creates incredibly jarring artifacts in the output images, when even a single segment goes missing. A similar, though slightly less extreme, thing happens when we fill the space with zeroes. Both options quickly result in a fully unrecognizable picture, when just a few segments are missing. A slightly more robust way to fill these gaps is by using the previous frame's data. Handling missing datagrams this way slightly increases the visual fidelity of images with missing data, though the effect is still very prominent and easily sufficient to ruin user experience. This leads to a phenomenon analogous to screen tearing (though it presents differently, due to the JPEG encoding).

Screen tearing is the appearance of horizontal lines in a frame when, during the monitor's refresh cycle, the frame buffer is modified, usually to the following frame. The slight difference between the two frames causes the appearance of a horizontal line, where the updated frame is noticeably different from the previous. It can be clearly seen in figure 3, where two instances of screen tearing are visible.

Fig. 3. Image showing 2 instances of screen tearing in a single frame [24]

Fig. 4. Example of JPEG corruption when some data (1280 bytes) is re-used from the previous frame

The JPEG equivalent we expect to see in our results is shown in figure 4. Since JPEG data is not simply a list of pixel color values, the effect can be a lot more pronounced. The image in figure 4 re-used a single segment (index 2943, ranging from byte 3767040 to byte 3768320) from the preceding frame. While the data used to replace the missing segment is clearly incorrect for this image, it results in significantly better fidelity than both other simple options (simply leaving the data out entirely or writing the appropriate amount of zeroes). Perhaps more elaborate methods of determining optimal filler data may exist, but these are out of scope for this study. The results displayed in the above image are extreme for a single segment being incorrect, but they are indicative for the type of corruption that can be experienced.

IV. Experiment Design

We consider a few different cases in our experimentation. As a baseline evaluation, we evaluate the performance at various distances, without any motion or obstacles (i.e. the routers maintain line of sight). The goal of these is to obtain a baseline performance level for us to compare to. We then add in complicating factors (motion and obstacles) and evaluate how much they impact the performance.

For our baseline experiments, where both routers are static and within line of sight, we consider four distances: 1 meter, 3 meters, 5 meters and 10 meters. We previously noted that around 10 meters should be sufficient for the network to experience performance degradation, the goal is to quantify that effect and determine how that impacts UX.

A. Motion

The directionality, clearly, is critical. We consider both best- and worst-case motion (w.r.t. beam-forming) in our experiments. The ideal direction for the beam between two devices, at any one point in time, is exactly the line connecting the two devices. It is trivial, then, that the best-case movement is travel along this line.

The worst case is circular motion, where the static router is the circle's center. At any point on the circle, the direction of the robot's movement (assuming it drives a perfect circle) is perpendicular to the line connecting the robot to the static router. This type of motion is thus the quickest to break the beams' alignment.

1) Linear motion: To quantify the effects of linear motion (along the ideal beam direction, as described above in section IV-A on the network's performance, we attempt to isolate the movement from the distance. This is achieved by having the robot repeatedly drive, back and forth, over a line segment of a set length (depending on the experiment). The robot's distance from the router will thus not vary between iterations of these experiments, whereas the speed will. This isolates the effect the robot's speed of movement has on the network's performance, controlling for the distance between the two devices.

We consider 2 sets of distances, 2 meters to 5 meters separation and 6 meters to 8 meters separation. Each of these is repeated at two speeds, 0.35 and 0.6 (conversions are available in table I - in this case, the values correspond to approximately 0.9 m/s and 2 m/s respectively).

2) Circular Motion: Due to space constraints, we used semi-circles (approximately 180°) rather than full circles. For all circular motion experiments, the robot drives along one of the semi circles at a constant speed until it reaches one of the end points, then it reverses until it reaches the other end point (repeat until end of the experiment, similar to movement repetition in linear movement experiments). A very rough illustration of what that looks like is included in figure 5.

Stopping the robot's movement while it is turning in order to turn around results in a slight drift in the robot's direction. We attempted to compensate for this by introducing an intermediate checkpoint in the programmed path, to offset the angular drift. This compensation is definitely not perfect, though, so the distance between the robot and the static router is slightly variable between repeats of the experiments.

We perform the experiments for two radii, 3 meters and 5 meters. Each circle is traversed at two different speeds, (0.3, [0.34,0.2]) and (0.6, [0.7,0.4]). The second item in the tuple contains both values that are used; the first is used for the

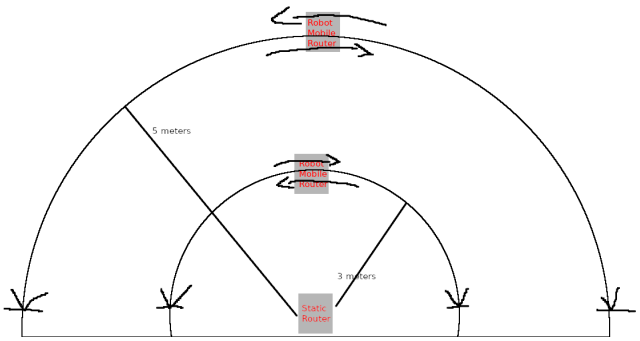

Fig. 5. Illustration of the robot's movements around the static router in our circular motion experiments

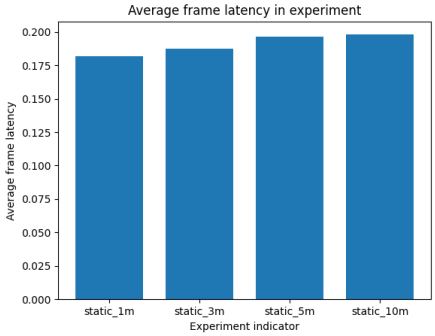

Fig. 6. Average frame latency for static experiments

smaller circle, the second for the bigger one (lower turning rate results in a larger circle).

B. Obstacles

Typically, when a VR system is in use, there are no obstacles in the immediate vicinity. As the user of the system is not able to see his surroundings, these would pose a safety issue. The only obstacle we consider, thus, is the human body.

For our testing, we employ essentially a worst-case scenario. The receiver is held up to the chest, while the back is turned to the transmitter, placing as much bulk of the body between the two. For the static experiments, this pose is held for the duration of the experiment. Unfortunately, this implies that the results we obtain in this section are not consistently reproducible. Someone with a slighter frame reproducing the exact scenario would most likely see better performance, for example.

Two linear motion experiments are also performed with a human body as an obstacle. In this case, the researcher holding the device moves back and forth between two marks on the floor, similar to the robot's movements described in section IV-A. The researcher, using a timer, attempts to mimic the robot's speed as closely as possible, though the results are not expected to be reproducible in this case. They serve as an indication of the expected performance when motion and obstacles are combined, more reliable testing of this case is required (further mentioned in section VI-A).

C. Evaluation criteria

Our report details the mean packet loss experienced in each experiment, which we use as a first indicative performance metric.

We add to this by visually comparing the images received by the robot's mounted mini PC and the transmitted originals. Which frames we compared was randomly selected (excluding all bit-wise identical frames from the pool) - visually comparing every received frame is not feasible, so this seemed like a fair solution. We've included only a single case in the paper, as it is more than sufficient to indicate that the experienced corruption is more than sufficient to make the images near worthless for display in a VR application.

In section V-A we note that the application might freeze the video when significant data loss is present, in order to avoid displaying entirely corrupted frames (slight hiccups in display would be preferred). As a naive implementation of this behaviour, we consider each frame with more than 10% dropped data to be too corrupt for display. This is computable at display time, as frame segments contain both their index as well as the total amount of segments expected this frame. As long as any frame data is received, the computation is possible (and if not, the video will freeze anyway). When we mention the amount of freezes in this evaluation, we count each frame that freezes the video, even if it immediately follows a frame that also froze the video (and thus is, in effect, part of the same freeze). We report on amount of frames causing a freeze in the video and a superficial analysis on chains of sequential freeze-causing frames to evaluate the performance. This metric is preferred over visual analysis in later sections of our results discussion, after we'd shown that the corruption is sufficient to make images unusable.

V. Results

The naming scheme for the experiments is as follows:

- static_x: Experiments without movement, distance between transmitter and receiver = x.
- straight_x-y_z: Linear movement, from x to y and back (repeat until experiment over) at speed z.
- circle_x_y,z: Circular movement, circle radius x, with speed y and turn rate z.
- x_obs: Identical (or as close as feasible) to any experiment x, but with a human body as obstacle added.

These names are mostly not used in the text (where we prefer more descriptive language), but may appear, for example, as labels on an axis in graphs.

A. Static experiments

In figure 6, we show the average frame display latency experienced in our experiments. While the consistency over various distances seems promising, the base latency experienced when the routers are close together is unacceptable for interactive VR systems. An action taken by the user not being displayed on the HMD for almost 200 ms is always unacceptable, as the

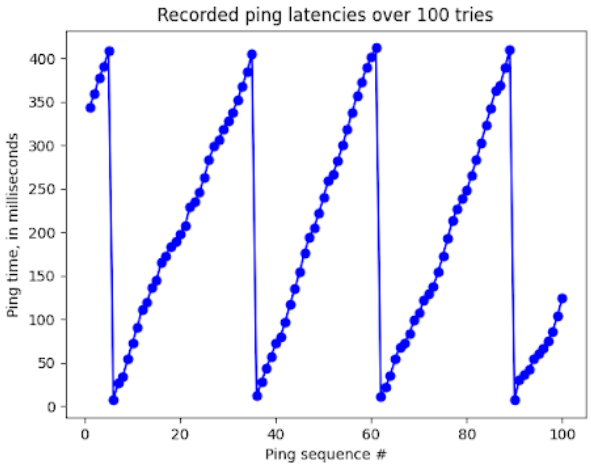

Fig. 7. Results from 100 ICMP pings from one router to the other (static, 2m apart, no obstacles)

acceptable period ranges from around 7 to 20 milliseconds depending on the user [2].

In early experiments, we noted a strange behaviour in the AD7200 Talon's latencies. When simply transmitting a sequence of pings, 1 every second, from one talon to the other, the latency was extremely variable, though in a very regular pattern, as seen in figure 7. We hypothesise that the high latency we see in our frame transmissions and the ping latencies are most likely related phenomena, with the same root cause. The cause of these observations is not identified in this work.

The packet loss (adjusted for frame display time - packets received after the frame is rendered are not included, as they cannot be used by the application) for these experiments are essentially identical; in each of the static experiments, the packet loss is approximately 2.5% ($\pm 1\%$).

While this number is high, it is, unfortunately, not sufficient to adequately carry the information required by our application. As seen before (section III-C2 and figure 4), even small amounts of information being lost can result in significant image corruption. If the packet loss rate were constant at 2.5%, most frames would be close to unrecognizable. The packet loss rate we experienced is very bursty, resulting in a sequence of bit-wise identical frames, followed by a few corrupt images. The brief flashing colors the user of a HMD receiving these images would see would be incredibly bothersome and clearly undesirable. To illustrate, we compare the 93^{rd} frame received in the static 10m separation experiment with that same frame received in the 10 meter with obstacle experiment.

In figure 8, 24 out of 4075 segments had been lost. The corruption appears in the bottom of the image, creating a tearing effect and, comparatively subtly, changing the color of the vehicle. In figure 9, it is immediately clear that the corruption is far more extreme. In this case, that is to be expected - approximately 25% of the image data was lost (1025 out of 4075 segments). While the corruption in the image is more than obvious, it is not completely unrecognizable. While that is fun to note, if this image were displayed to a user on a HMD, it is clear that the UX would be quite terrible so the image being recognizable isn't relevant. For comparison, the original image is included in figure 10.

An application transmitting video in this way may, when not receiving such a significant amount of data, prefer to freeze the video stream. Especially when such extensive corruption (dropped data) is present, the UX impact of a temporarily frozen video is lower than the impact from such distorted frames. When the following frame is received, display would resume, simply dropping the corrupted frame.

1) Obstacle: The effect of an obstacle appears greater when the routers are further apart. While the static experiment at 3 meter separation also included a slight increase in packet loss (around 2 percentage points), the difference is far more significant in the 10 meter case, where the packet loss percentage increases to 17.9%. Intuitively, this makes sense. When the signal has already travelled a longer path, it has experienced more propagation loss and the SNR is lower when the signal encounters the obstacle. Trivially, a weaker signal will be less likely to be recognizable once it has passed an obstacle.

B. Motion

In all motion experiments, frame delays are similar to what we observe in the static case (at most around a 10% increase). This is not further discussed as the initial value we observed was already so high as to be debilitating to any interactive application and so is of limited interest.

We first consider linear motion. As described, we test four cases without any obstacles present:

- 2 to 6 meters, at speed 0.35
- 2 to 6 meters, at speed 0.6
- 6 to 8 meters, at speed 0.35
- 6 to 8 meters, at speed 0.6

The packet loss statistics (in time for frame display, as before) are in line with what one might expect; the slower speeds perform slightly better than the faster ones. In the case of the 2 to 6 meter interval, the loss is 4.2% and 6% respectively. In both cases, the experienced loss is very bursty, some images missing hundreds of sequential data segments. In the 6 to 8 meter interval, the percentage of packets being lost is 4.7% and 9.3%. Speed of the robot's motion thus clearly impacts the amount of packets lost and, similarly to the obstacle effect observed in the static case, the effect is more pronounced when signal quality is already low (when distance is greater).

The video stream in the 6 to 8 meter interval, at speed 0.35 would have frozen for 33 frames, the longest chain of sequential frames was 3. 3 sequential frames being dropped would be a very noticeable bump in the display, especially if the following frame is not entirely complete as the filler data used would be more outdated than usual.

This worsens, significantly, at speed 0.6 over the same interval. The video feed would freeze 57 times. The longest freeze is 6 sequential frames, but there would be 3 of these

Fig. 8. Frame 93 from a 10 meter separation without obstacle transmission

Fig. 9. Frame 93 from a 10 meter separation with obstacle transmission

Fig. 10. Original 93^{rd} frame of the video

6-frame freezes present. The stuttering that would cause in the display is unacceptable. With the frame rate at 10 fps, these are freezes lasting over half a second.

1) Obstacle: For our linear motion case, we include a case where the router is being carried, with the carrier's body serving as an obstacle for the network. The receiver is being held up at chest level, as close to the chest as feasible, while keeping the back turned directly toward the transmitter to block as much of the signal as possible. We performed the experiment on the interval 6 to 8 meters, with travel duration for this distance measured at robot speed 0.6 and on the interval 2 to 6 meters, based on robot speed 0.35.

Using a timer beeping at a regular interval (the time it took for the robot to travel between the markers), the researcher carrying the device went back and forth between two markers on the floor, attempting to reach them as close to the beep as possible every time (variance is definitely unavoidable in this case, so these results will not be consistently reproducible). Due to the limited reproducibility, these experiments were only performed once.

The combination of motion and obstacles is clearly killer for the performance; in the slow, close distance experiment, packet loss is 20%. In the faster, long distance (i.e. 6-8 meter interval) experiment, we see 47.5%(!) packet loss.

Both of these values are incredibly high and likely to make any application heavily reliant on the network nigh on unusable. It is possible these results are inflated, and would be reduced by averaging over multiple iterations of the experiment, but a significant increase in packet loss through the combination of motion and obstacles is expected.

2) Circular motion: In these experiments, the packet loss is significantly worse than both linear motion and static with obstacle. As we chose this evaluation metric to represent a worst-case situation for beam-forming, this is not entirely unexpected.

Frames become wholly unrecognizable, and some even are dropped completely. The packet loss, in the easiest case of this experiment set (3 meter radius, speed=(0.3,0.34)), is 9%. This is comparable to the results we see in the long distance, fast-moving linear motion case. Clearly, the impact of circular motion is significantly greater than that of linear motion, as expected.

It only gets worse from there. At a 3 meter radius, with speed (0.6, 0.7), we see 11.2% packet loss.

For the circle with a radius of 5 meters, the results are 16.2% and 22% packet arrival rates for slow and fast movement respectively. In both of these cases, over a third of the frames lose more than 10% of their data and would lead to freezes in the application. The longest chain of sequential frames with more than 10% loss is 15, leading to a 1.5 second stutter in the display. There's also a number of shorter sequences of freeze frames, where only 1 or 2 frames are being displayed in between. This, too, isn't great for the network's performance.

The packet loss in the faster case is notably higher, but the amount of frames that causes the application to freeze does not notably increase, indicating that a lot of the loss was bursty, within frames. Bursty losses are definitely expected, e.g. when beam forming sector selection has to be done (because of misalignment due to movement).

To finalize our results, we display the frame freezes over all the experiments in figure 11. We chose a single iteration of the experiment to display in figure 11, with the median number of freeze frames per experiment. The lines drawn across the screen are the duration of a single experiment, each line is built up of line segments, colored green and red (green for no freeze, red for freeze) based on the packet loss of the preceding frame (i.e. if frame index 0 didn't lose any data, the line segment going from index 0 to index 1 is colored green. If it lost more than 10% of data, that line segment would be colored red).

We identified 3 influential factors on network performance that we attempted to control for; distance, motion and obstacles. Based on this image, it is obvious that the presence of one slightly disrupts the network's performance (e.g. static_3m_obs or straight_2-6_[0.35,0.6]), it is mostly acceptable. A few freezes are apparent, but they are relatively few and far between. In this exact configuration, the performance would be unacceptable (10 fps display, which still freezes would be quite unpleasant if any significant motion were present in the scene), but we note that the data rate we used is the highest we could push the baseline without starting to drop traffic - so any complicating factor is expected to cause performance issues. When an implementation on top of this network were to transmit less data (by e.g. lower resolution video or using video encoding rather than frame-by-frame), the resulting performance would be much greater and the effects we observe now would be diminished.

When more than one of the factors is present, though, performance drops significantly. Many more red segments start to appear in the cases where two complicating factors

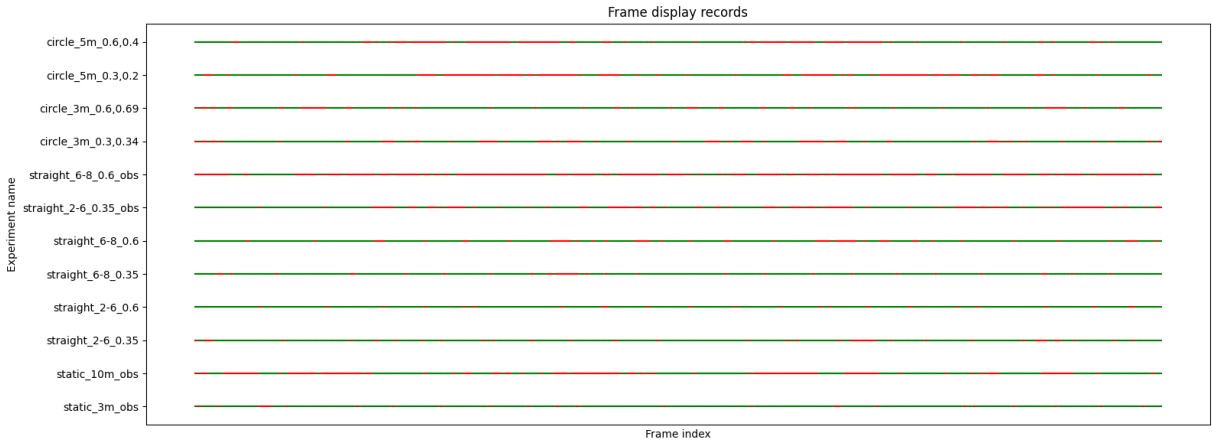

Fig. 11. Graph showing the frames where the video freezes in red (cut-off at 10%)

occur. In some of these cases, we expect that utilizing video compression and a feedback channel (to enable e.g. variable frame rate or dynamic resolution scaling) would allow a part of the problem to be mitigated (though some problems will remain - data loss during a sector sweep is hard to avoid).

When all three are present, performance becomes truly horrendous. As seen in the straight_6-8_0.6_obs case, the video spends more time frozen than it does actually displaying frames. Of the frames that do get displayed in that case, most are still missing a few segments of data, so there's no guarantee that the video the user would get to see was even recognizable.

VI. Future work

The simple router to router communication we studied in this work is insufficient to guarantee good performance for an interactive VR application using frame-by-frame encoded video. Luckily, many improvements are possible. These require further study to identify which ones (or combinations thereof) have the best resulting performance and lead to acceptable conditions.

We note four significant improvements that must be studied further.

A. Obstacle movement

Our experiment with a moving obstacle (close to the receiver, as would be typical with a HMD) use a non-reproducible method (research carrying the receiver, walking between floor markers). These experiments were performed a limited number of times and so the results may not be representative. A future work should find a way to consistently move an obstacle around with a receiver nearby and block the signal in a more realistic, intermittent way (a user of a VR system won't generally block as much as possible at all times). Additionally, we performed no testing to determine the impact of obstacles in the non-linear movement case. We expect a result showing a similar impact to the linear case, but this is not confirmed in our work.

B. Video encoding

An analysis of the performance of a video encoded stream is a next step. We identified that the frame-by-frame encoding used in this work significantly reduces the achievable performance on the application layer (through a 23-fold increase in data rate requirement to transmit at the same frame rate as the source video). A video encoding is more loss tolerant (losses tend to only corrupt parts of the frame, rather than the entire image) and takes less resources to transmit, which should allow significantly improved performance.

C. On-device antennas

As discussed in section III-C2, the ethernet connection between the two endpoints and the routers may be a bottleneck in the system if the data rate is pushed to the wireless interface's limit. A future work should remove this bottleneck from the system and verify that results remain similar.

D. Application of literature proposals

As noted before in section II, there are relevant works in the literate proposing solutions that promise to significantly improve the performance of mm-Wave wireless networks in the context of VR applications. Specifically, MoVR [7] and coVRage [10] are promising. The combination of both should provide an ideal circumstance. The implementation of coVRage would improve the system's resilience to motion, moVR improves the resilience to blockages. The combination of the two is promising to reduce the impact most of the limitations experienced mm-wave WiFi networks.

VII. Conclusion

We've shown that the VR frame-by-frame streaming application presented in this paper does not achieve acceptable performance for a VR HMD. Frame-by-frame encoding is definitely not a great candidate for such an application, though, as its loss tolerance is limited and the data rate requirement

is, compared to video encodings, massive (see section III-C2 for more details). A video encoding being transmitted at a comparable data rate could achieve acceptable performance for environments with no high-speed movement.

While our application is unsuitable for display in a VR HMD, the transmission data is fully suitable to benchmark the network, as we've done. We note that the network's performance is mostly resilient to any one of distance, movement and obstacles (within reason). We considered distances up to 10 meters in a static environment, where the network performance degradation was limited. Similarly, introduction of a human body obstacle while the routers are close together and fully static or the addition of movement over a relatively close-by interval does not cause the network performance to drop dramatically (small, consistent changes are noted, but they are not debilitating).

This changes when multiple factors coincide - in experiments where we consider two of the three factors we controlled for, the network performance degrades significantly. Packet loss starts spiking up to near 20% in the more extreme cases, images become unrecognizable and a video streaming that freezes rather than display corrupt images would suffer significant disruptive stuttering.

This worsens even more when the all three factors are combined. The network drops close to half its packets in the case where we introduce an obstacle, movement and a significant distance between the routers simultaneously. The packet loss rate of near 50% mostly prevents any network-reliant applications from performing useful work. When these three combine, as can happen occasionally during use of a VR system (e.g. far corner of the allocated space, back turned to the transmitter, head tilted down), performance is expected to degrade significantly. The effect might be less extreme than what we observed (as we attempted a worst-case scenario for the obstacle, which is unlikely to occur), but even a less extreme case where, e.g. 40% of packets are dropped, will be extremely disruptive for the VR application.

REFERENCES

[1] I. VR. (2021) The importance of frame rates. [Online]. Available: https://help.irisvr.com/hc/en-us/articles/215884547-The-Importance-of-Frame-Rates

[2] J. Struye, F. Lemic, and J. Famaey, "Towards ultra-low-latency mmWave Wi-Fi for multi-user interactive virtual reality," in *GLOBECOM 2020 - 2020 IEEE Global Communications Conference*, 2020, pp. 1–6.

[3] J. Lu, D. Steinbach, P. Cabrol, P. Pietraski, and R. V. Pragada, "Propagation characterization of an office building in the 60 GHz band," in *The 8th European Conference on Antennas and Propagation (EuCAP 2014)*, 2014, pp. 809–813.

[4] T. Nitsche, C. Cordeiro, A. B. Flores, E. W. Knightly, E. Perahia, and J. C. Widmer, "IEEE 802.11ad: directional 60 GHz communication for multi-gigabit-per-second Wi-Fi [invited paper]," *IEEE Communications Magazine*, vol. 52, no. 12, pp. 132–141, 2014.

[5] TP-Link. (2016) Talon ad7200 multi-band Wi-Fi router. [Online]. Available: www.tp-link.com/us/home-networking/wifi-router/ad7200/

[6] C. Cai, Z. Chen, and J. Luo, "Hackman: hacking commodity millimeter-wave hardware for a measurement study," in *Wireless Netw 26*, 2020, pp. 5411–5425.

[7] O. Abari, D. Bharadia, A. Duffield, and D. Katabi, "Enabling high-quality untethered virtual reality," in *14th {USENIX} Symposium on Networked Systems Design and Implementation ({NSDI} 17)*, 2017, pp. 531–544.

[8] R. Robotics. (2021) Rover robotics software stack. [Online]. Available: https://github.com/RoverRobotics/roverrobotics_stack

[9] H. Assasa, S. Kumar Saha, A. Loch, D. Koutsonikolas, and J. Widmer, "Medium access and transport protocol aspects in practical 802.11 ad networks," in *2018 IEEE 19th International Symposium on "A World of Wireless, Mobile and Multimedia Networks" (WoWMoM)*, 2018, pp. 1–11.

[10] J. Struye, F. Lemic, and J. Famaey, "Millimeter-wave beamforming with continuous coverage for mobile interactive virtual reality," *CoRR*, vol. abs/2105.11793, 2021. [Online]. Available: https://arxiv.org/abs/2105.11793

[11] R. Zhong, M. Wang, Z. Chen, L. Liu, Y. Liu, J. Zhang, L. Zhang, and T. Moscibroda, "On building a programmable wireless high-quality virtual reality system using commodity hardware," in *Proceedings of the 8th Asia-Pacific Workshop on Systems*, ser. APSys '17. New York, NY, USA: Association for Computing Machinery, 2017. [Online]. Available: https://doi.org/10.1145/3124680.3124723

[12] L. Liu, R. Zhong, W. Zhang, Y. Liu, J. Zhang, L. Zhang, and M. Gruteser, "Cutting the cord: Designing a high-quality untethered VR system with low latency remote rendering," in *Proceedings of the 16th Annual International Conference on Mobile Systems, Applications, and Services*, ser. MobiSys '18. New York, NY, USA: Association for Computing Machinery, 2018, p. 68–80. [Online]. Available: https://doi.org/10.1145/3210240.3210313

[13] T. T. Le, D. V. Nguyen, and E.-S. Ryu, "Computing offloading over mmWave for mobile VR: Make 360 video streaming alive," *IEEE Access*, vol. 6, pp. 66 576–66 589, 2018.

[14] M. S. Elbamby, C. Perfecto, M. Bennis, and K. Doppler, "Toward low-latency and ultra-reliable virtual reality," *IEEE Network*, vol. 32, no. 2, pp. 78–84, 2018.

[15] M. Huang, J. Wang, and X. Zhang, "Multi-beam multiple access scheme for uplink traffic of wireless virtual reality with millimeter-wave analog beamforming," in *2018 IEEE International Conference on Communications Workshops (ICC Workshops)*, 2018, pp. 1–6.

[16] G. Bielsa, A. Loch, I. Tejado, T. Nitsche, and J. Widmer, "60 ghz networking: Mobility, beamforming, and frame level operation from theory to practice," *IEEE Transactions on Mobile Computing*, vol. 18, no. 10, pp. 2217–2230, 2019.

[17] D. Steinmetzer, D. Wegemer, M. Schulz, J. Widmer, and M. Hollick, "Compressive millimeter-wave sector selection in off-the-shelf IEEE 802.11ad devices," in *Proceedings of the 13th International Conference on Emerging Networking EXperiments and Technologies*, ser. CoNEXT '17. New York, NY, USA: Association for Computing Machinery, 2017, p. 414–425. [Online]. Available: https://doi.org/10.1145/3143361.3143384

[18] R. Robotics. (2020) 4WD rover pro bundle. [Online]. Available: https://www.robotshop.com/en/rover-robotics-4wd-bundle.html

[19] adlinktech. (2020) Adlink Vizi-AI development starter kit. [Online]. Available: https://www.adlinktech.com/en/News_20040601215272582

[20] N. Mizuno. (2021) ROS driver for the DualShock 4 controller. [Online]. Available: https://github.com/naoki-mizuno/ds4_driver

[21] D. Mills, "Internet time synchronization: the network time protocol," *IEEE Transactions on Communications*, vol. 39, no. 10, pp. 1482–1493, 1991.

[22] IEEE, "IEEE standard for a precision clock synchronization protocol for networked measurement and control systems," *IEEE Std 1588-2019 (Revision of IEEE Std 1588-2008)*, pp. 1–499, 2020.

[23] M. Carrascosa and B. Bellalta, "Cloud-gaming:analysis of Google Stadia traffic," 2020.

[24] V. Ezekowitz. (2009) Screentear. [Online]. Available: https://commons.wikimedia.org/wiki/File:Tearing_(simulated).jpg