Collection and Analysis of Virtual Reality Workload Traces

Radu Apşan
Vrije Universiteit Amsterdam
The Netherlands
r.apsan@student.vu.nl

Damla Ural
Vrije Universiteit Amsterdam
The Netherlands
d.ural@student.vu.nl

Paul Daniëlse
Vrije Universiteit Amsterdam
The Netherlands
p.e.g.danielse@student.vu.nl

Vlad-Andrei Cursaru
Vrije Universiteit Amsterdam
The Netherlands
v.cursaru@student.vu.nl

Eames Trinh
Vrije Universiteit Amsterdam
The Netherlands
e.v.t.trinh@student.vu.nl

Joshua Offermans
Vrije Universiteit Amsterdam
The Netherlands
j.j.a.offermans@student.vu.nl

ABSTRACT

While research on the Metaverse and virtual reality (VR) has been ongoing in the computer science community for over two decades, these topics have recently gained the attention of both the community and society. However, there is an apparent lack of tools to generate traces, benchmark, and test the performance and energy efficiency of VR systems in a reproducible and transparent manner. This study investigates the replicability and behaviour of VR traces collected using Meta's leading VR headsets, the Meta Quest Pro and Meta Quest 2, while analyzing their network and energy footprints. We design, develop, and validate an experiment orchestrator for VR experiments, and extend and validate a tool to aid in the replicability of VR experiments. Using these two tools, we run network and energy efficiency-related experiments and find that (i) the Meta Quest Pro requires more GPU resources to run compared to the Meta Quest 2, (ii) the Meta Quest Pro consumes more energy under low-quality network conditions compared to ideal network conditions. In the interest of FAIR data, and free and open-source software (FOSS) principles, all of the data generated over the course of this study is available through public GitHub repositories at https://github.com/Raiduy/librnr-experiments.

KEYWORDS

virtual reality, metaverse, gaming, computer networks, sustainability

1 INTRODUCTION

The Metaverse and virtual reality (VR) have been explored in the computer science community ever since the coining of the *metaverse* term. These topics have recently gained the attention of both the community and society thanks to Meta's $36 billion cumulative investments in their VR/AR applications department (Reality Labs) [10], Apple's entry in the XR market with their new Apple Vision Pro [3], and the release of the Meta Quest 3 [11].

As of 2022, the VR industry is forecasted to continue growing[1], with VR gaming currently holding the biggest share of the total VR market revenue at 76.5%. Along with the surge in profitability, literature over the past three years found VR to be a valuable asset to the medicine [4, 15], mental health [7], construction [1], and law enforcement [12] domains by improving professional trainings in immersive virtual environments.

[1]https://www.pwc.com/gx/en/industries/tmt/media/outlook/segment-findings.html

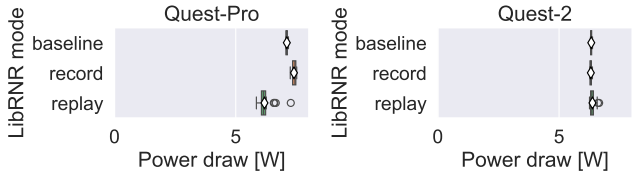

Figure 1: Validation of `librnr` energy overhead on the two VR devices.

Despite its massive growth, academic research into virtual reality remains underdeveloped with regard to VR device-specific performance, and energy efficiency, which threatens the longevity of VR systems. The lack of tools to generate traces, benchmark, and test the performance and energy efficiency of VR systems in a reproducible and transparent manner becomes more and more apparent. Traces are crucial to the reliable testing and evolution of an IT domain as they provide real-world utilization characteristics that allow researchers to replicate the behaviour of systems in simulation environments. Moreover, VR experiments are very difficult to replicate, as the systems under test require the user/experimenter to utilize the headset. Due to this, it is impossible for any person to replicate the exact movements with the VR controllers as the original experimenter, therefore rendering most VR experiments irreproducible, and thus, not fully transparent.

The VR domain needs to catch up to the rest of the computer science fields to ensure its longevity. Over the past years, databases of traces have been created and made publicly available in the interest of boosting research rates in their respective fields, such as datacenter workflow traces for the datacenter industry [14], and gaming traces [9] for the gaming industry. These traces are lacking completely for VR systems. Moreover, few peer-reviewed articles tackle the performance of VR systems under different network loads [5], and no research in terms of VR energy efficiency.

In this work, we address the lack of software solutions in replicable VR experiments by proposing an open-source tool that improves the quality and speed of running VR experiments. With this, we hope to boost the rate at which VR research is done and improve the transparency in the domain.

Addressing the knowledge gap, we study the effects of different less-than-ideal network connection types on the performance and energy efficiency of the VR headset (see Figure 1), and the performance of the host machine. We analyse the behaviour of

two of Meta's VR headsets: the flagship Meta Quest Pro, and the budget-friendlier Meta Quest 2.

Our contributions are as follows:

- **C1** We introduce an experiment orchestrator - `VR RUNNER` (in Section 3). VR Runner automatically executes experiments on Meta VR headsets and tracks numerous parameters relevant to the performance and energy efficiency of the VR device and host computer. The orchestrator provides a plug-and-play solution by launching and closing the specified applications and controlling network parameters through `clumsy` (**Bonus Feature 4 - own idea**).
  We extend Record 'n Replay (for VR) (`librnr`) tool to capture joystick inputs, an important source of user input in many applications. We further add the ability to record haptic feedback events, this might allow the synchronisation of a trace with the actions happening in the game (**Bonus Feature 3 - extend librnr**).
- **C2** We conduct validation of the experiments for tools and environment setup that are used for the hands-on part of this study in Sections 5.1, 5.2, and 5.3. The tools are analysed from (i) an overhead perspective, and (ii) a timing perspective.
- **C3** We design and conduct real-world experiments in Sections 5.4, 5.5, and 5.6. As part of our experiments, we validate our experiment tools including `librnr` itself, and `clumsy`, an open-source tool that allows manipulating network characteristics on Windows platforms. We analyse the experimental results and find that `librnr` and `clumsy` are tools that can be used in an experimental environment thanks to their low overhead over the systems under stress. Moreover, we find that VR systems are sensitive to bandwidth and packet loss, both of which have a high negative impact on framerate which heavily affects the user experience (**Project Requirement 1, 2, 3 - select representative applications, collect workloads from apps, analyse data collected**).
- **C4** We analyse the recorded haptic feedback events within the game Beat Saber. In section 3.5 we show the behaviour of these events and their usefulness better support replay functionality within `librnr`.
- **C5** We implement a generative model for input traces in Section 3.4.5 (**Bonus Feature 1 - trace generation**).

2 BACKGROUND

In this section, we attempt to cover the main topics and concepts that are crucial to the understanding of this project. We first address the way a VR setup works and the parts that are relevant to our study in Section 2.1. We describe the concept of traces and their relevance to the IT field in Section 2.2 and discuss the metrics that we collect throughout our experiments in Section 2.3.

2.1 System Model

The primary pieces of hardware are a virtual reality system that includes the headset and two controllers, and a computing device that runs the VR-compatible game. This can be viewed as part of a cloud computing model, where inputs are streamed to a cloud instance over the internet [5], which adds local architecture complexity (associated with networking over a Wide Area Network) but relieves the stress on the local network.

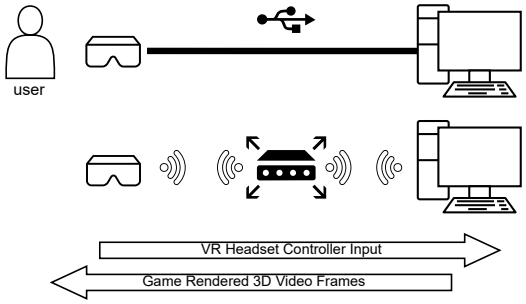

Figure 2: Two common methods of linking a desktop PC to a VR device, used for computational offloading.

Table 1: `librnr` Trace record types. "*" indicates record types added in this work.

DataType	Explanation
Space	An object's position in the virtual world
View	The user's orientation and direction in the in the virtual world
Float	Pulling the trigger button
Boolean	Pressing buttons
* Vector2f	Moving the joystick on the controllers
* ApplyHaptic	Haptic feedback when it is turned on

An overview of how the system works can be seen in Figure 2. We describe the way the system works from left to right in the figure. For the sake of this explanation, we assume that the VR-compatible and OpenXR-compatible application is already running on the Host PC. Increasingly, new and popular applications support the OpenXR standard. For example, BeatSaber, one of the most successful VR applications [13], was recently ported to include support for OpenXR [8]. The Player wears the headset and is holding the two controllers. When the Player performs an action, the VR device sends a signal to the Host PC through the OpenXR Device Plugin Layer that is part of the OpenXR Runtime. The signal is then forwarded to the engine running the application, in our case, Unity Game Engine, which then renders the result of the action that the player performed. When the rendered frame is ready, it is sent back to the VR device to be displayed to the player.

2.2 Traces

Traces are data objects used to encapsulate the key characteristics of the evolution of a workflow. Versluis *et al.* explain workflow traces as "[...] a recording of useful, relevant information during the processing of the workflow. Traces can be used to create models with, or used in emulations and simulations to replay the execution of a workflow in a controlled environment" [14].

In the context of VR experiments, traces are collected using `librnr` and stored as `.txt` files. Each trace includes the information shown in Table 1.

2.3 Metrics

We collect the average current battery of the VR headset in microamperes using the BatteryManager API[2]. Then we use the power to calculate the energy consumption in Joules (J). Given that every trace recording has a different length the energy consumption is normalized to power draw in Watts (W). Furthermore, we monitor the CPU and GPU usage of the PC and analyze the memory utilization difference during the process of simply playing the game, and during recording and replaying a trace. Lastly, we collect system and network statistics such as the FPS and the received data to analyze the performance of the VR when it is connected to the PC via WiFi.

3 IMPLEMENTATION

In this section, we describe `librnr` in detail and discuss the implementation of additions to it. We discuss the implementation of `VR RUNNER`, the integration of clumsy for use in experiments, and the LSTM model to generate traces. Furthermore, we analyse the haptic feedback events produced by Beat Saber.

3.1 Record 'n Replay

To recreate the same test conditions and to allow easy benchmarking we make use of the functionality of the ecosystem that supports the connection between the computer and the VR headset. This system is a collection of programs that adhere to the OpenXR standard. Shown in Figure 3 a VR headset sends its sensor data through the OpenXR Device Plugin Layer to an OpenXR Runtime Application. All controller inputs (eg. buttons or the position of the headset in space) from the VR system have been registered at the Runtime Application during the setup of the connection. In the registration step, a path string is added that maps the actual input to its intended use inside a game. For example, the thumbstick of the left controller is mapped as "/user/hand/right/input/thumbstick". The VR headset continually sends updates to the Runtime Application where the state of all inputs is captured. Any application can asynchronously retrieve this information and use it.

When the recording is enabled, each I/O request made by a game to the Runtime Application invokes a function in which the retrieved data is serialized and written to the `trace.txt` file together with a timestamp of the current frame time. When the replay feature is enabled the I/O requests return the data written to the file hereby bypassing the actual VR headset controller input.

The OpenXR Specification describes an API layer loader. An API layer allows developers to develop and integrate functionality into the I/O of the Runtime Application. `librnr` is implemented as an OpenXR API Layer.

3.2 Non-Functional and Functional Requirements

To further analyse `librnr` we introduce some non-functional and functional requirements for such a tool.

NFR1 Should run on various VR headset systems such as Meta and HTC Vive that use OpenXR - Ideally, traces should be runnable and reproducible in multiple systems to increase the overall availability of the software.

NFR2 Should exist as a supplementary layer that does not noticeably interfere with system performance - If user experience is significantly affected, this will also impact the quality of the traces. This will be validated by measuring CPU and GPU utilization on the host PC and by measuring frame rate on the headset. The frame rate should not fall below a refresh rate of 48ms [6].

NFR3 Should have minimal energy consumption - As simulations are computationally intensive (and will run multiple trials on multiple devices), the system should not add additional energy consumption *during replay mode* (recording can be overlooked as it only occurs once). Energy consumption is measured in power draw [W].

In Section 5.1 we explicitly validate and thereby meet NFR2 and NFR3. No investigation has been done into NFR1. We have however used both the Meta Quest 2 and Meta Quest Pro to record and replay traces.

FR1 Should be able to be sideloaded into the VR's OpenXR runtime API - The system should not "replace" anything to obtain data. To achieve *passive measurement*, it should simply hook into existing libraries and "snoop" on its communication.

FR2 Should be able to replay specific traces by editing a configuration file to point to the recorded trace .txt file - To switch between traces, the system needs somewhere to select the desired trace.

By implementing `librnr` as an OpenXR API Layer we meet FR1. As shown in Figure 3 there is a configuration file. This is used to tell `librnr` what path to use for the trace and thereby we meet FR2.

3.3 Implementation Requirements

We introduce some requirements for our implementation process and the results of the implementation.

IR1 Any software produced should be open source and freely available on GitHub.

IR2 Any data or datasets required to run the software should be freely available and included with the software

IR3 Any change to an existing project should be validated by doing a pull request to add the changes to the project.

The implementation requirements are not strict. for example we chose to not include requirements for software testing because of the limited scope of this paper.

3.4 Implementation

In this section, we cover the tools that were implemented throughout this study.

3.4.1 Additional `librnr` *Features.* We integrate logging of the *Vector2f* and the *ApplyHaptic* types to allow for joystick replay and analysis respectively. The data types are serialized and written to the trace file for replaying. During replays, the API layer reads from the trace file until the times stamp matches the current frame time of the application, then the row is read and the controller inputs that are sent from the VR headset are replaced with the read values.

[2]https://developer.android.com/reference/android/os/BatteryManager

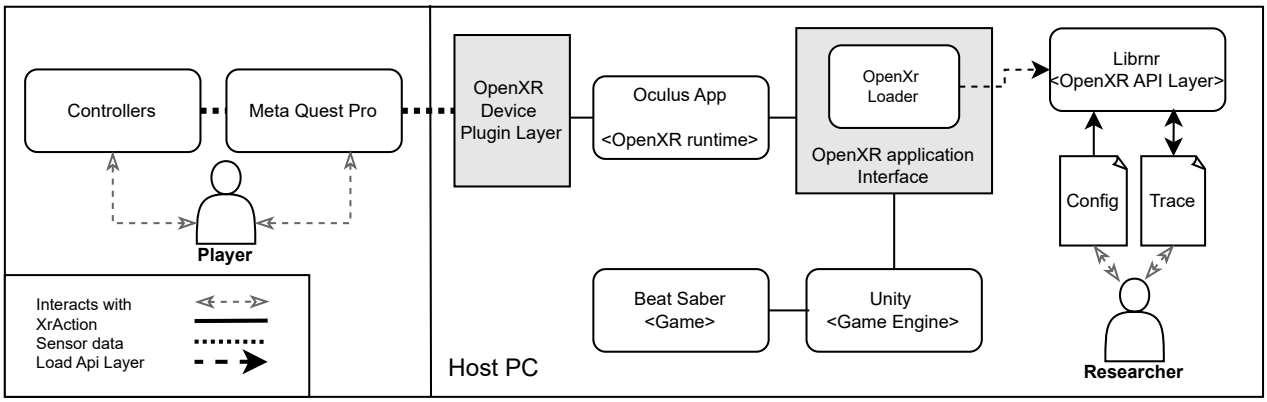

Figure 3: Diagram detailing communication between the host PC and the VR headset including `librnr`.

Currently, replaying traces is not consistent. This is mainly due to variations in loading time. This produces faulty behaviour when using different hardware/software. *ApplyHaptic* adds support to analyse for a method of obtaining more agile replay functionality. In section 3.5 we analyse and outline an approach to enhance the replay functionality. The `librnr` API Layer and code can be found at: https://github.com/atlarge-research/librnr

3.4.2 Trace Analysis Tools. To aid the visualization of traces we create a set of python tools that read and transform the collected traces to a Pandas Dataframe. Functionality was also added to write the dataframe back into a trace file. This for example enables editing traces but also graphing them. The python scripts including the collected traces can be found at https://github.com/peg-danielse/librnr-trace-repository

3.4.3 `clumsy` *Integration.* To aid with the automation of the benchmarking process, we created a wrapper script around the `clumsy` network traffic shaper. The wrapper simplifies the complex argument structure of clumsy to a small number of options for ease of use. The script is accompanied by PowerShell scripts which stop `clumsy`, as well as scripts that use Windows Quality of Service Policies to cap bandwidth more accurately than `clumsy`. The scripts can be found at https://github.com/Vlad2000Andrei/DistributedSystems-Traffic-Shaper

3.4.4 VR Runner. The system introduces an experiment orchestrator: VR Runner. Given the trace path, output directory, and the number of repetitions, VR Runner automatically launches the specified application, replays the trace, stores the performance metrics in the output directory and terminates the application once the replay process is complete. This is repeated as many times as specified. The inputs must be specified in a `config.json` file. Furthermore, the VR Runner can also integrate the `clumsy` scripts described in Section 3.4.3. The desired network modifications such as the `delay, bandwidth` and `drop_chance` must also be specified in the `config.json` file. VR Runner can be found at https://github.com/Raiduy/librnr-experiments

3.4.5 Generative Trace Model. In our research, we explore the feasibility of constructing a generative AI model to simulate VR traces.

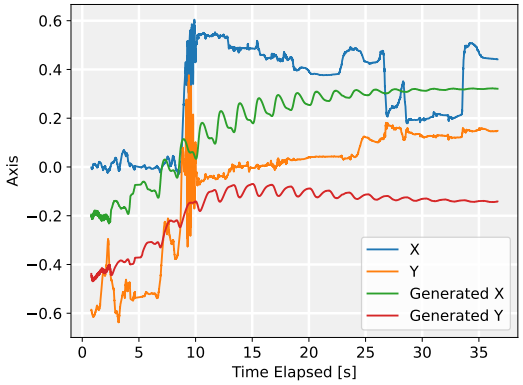

Figure 4: Original vs Generated Traces

The primary objective is to accurately replicate these traces, transforming them into replayable game interactions. This approach aims to facilitate automated simulation testing using the generated traces. The initial phase of our study involves data exploration and preprocessing of the dataset of the generated traces. They are predominantly comprised of space traces (constituting 77% of the data). Notably, haptic feedback traces are the least represented in the dataset, accounting for only 0.27%. Addressing the complex imbalance in the data and developing a model capable of generating realistic traces presented significant challenges. Consequently, we narrow the scope of our model to focus on the most prevalent type of action, specifically the Space trace associated with the action '/user/hand/left/input/grip/pose'. For the model architecture, we opt for a configuration employing multiple stacked Bidirectional Long Short-Term Memory (BiLSTM) layers. We selected a BiLSTM due to its superior performance in comparison to other Recurrent Neural Network (RNN) architectures, as evidenced by relevant studies [2]. Our architecture comprises three layers, each with 17 neurons, interspersed with a dropout layer set at 1%. Through experimentation with various dropout rates, we started with a higher dropout rate of 20% but switched to a lower dropout rate as it worked better

because of the high level of complexity of the model. We found that a 1% rate yielded the most favourable results in terms of loss reduction. Furthermore, we make the traces playable by taking an existing trace file and replacing all instances of the specific type '/user/hand/left/input/grip/pose' with the output from our model.

However, it is important to note that the resulting game movements appeared somewhat random and nonsensical. This outcome highlights the complexity of accurately simulating human interactions. You can also see this in the trace plots; as shown in the figure, the attempt is made to capture some of the complexity, but it only works to a small extent. The traces model is available at the earlier mentioned `librnr` on the traces model branch. https://github.com/atlarge-research/librnr

3.5 Trace Synchronization Analysis

If the loading time is shorter or longer, the replay of a trace might becomes desynchronized from the game. To analysis how to effectively synchronize actions taken in the game with the actions being performed by a replay we made recordings of the haptic feedback applied to the controllers during a replay. As a test case, we use Beat Saber. In Beat Saber, the sabers of the user input need to be in the right place at the right time. A possible way to ensure that this always happens could be to wait for haptic feedback to be generated by the game before continuing with the replay (haptic feedback is generated when a block is touched by a saber). If the replay has to wait for the haptic feedback to come in before it can continue with other movement we need to ensure that the replay is never ahead of Beat Saber. To do this the trace will have to be adapted to remove loading screens from the recording. A possible way to do this would be to use the fps drop that happens during a loading screen. To collect fine-grained FPS statistics we make use of MSI afterburner together with the Rivia Tuner Statistics Server. This allows us to collect the required data at a frequency of 4Hz. To collect the haptic feedback event data `librnr` was adapted to record the events during a replay.

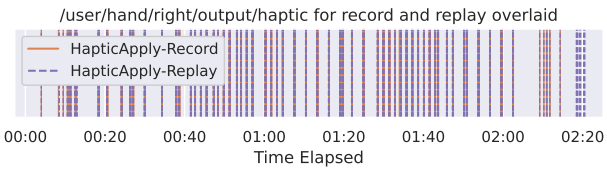

Figure 5: The haptic events overlaid and synchronized on the first haptic event.

In Figure 5 a recording and a replay are shown for the level *Beat Saber-Easy*. The recording consists of 3 phases. 1) Loading Beat Saber and choosing a song. 2) Loading and then playing the song. 3) Exiting the game. The first barrier where traces desync is at the border between phase 1 and 2. As seen in the figure around the 20 second mark we see a drop in fps as a result of this. In the bottom graph of Figure 5 we can see the recording and the replay haptic feedback events overlaid. This shows the predictability of the events. However, as is shown in Figure 6 the HapticApply events come in bursts. Hitting a block in game actually triggers multiple events. From some analysis of the level *Beat Saber-Extreme* these events follow each other so quickly that no stop in Haptic Apply events is detectable. Furthermore, Figure 6 shows that although the events are roughly synchronisable some small differences still occur. Lastly, The HapticStop event was analysed and as we can see this event happens almost every frame and is therefore fully unsuitable to synchronise traces on.

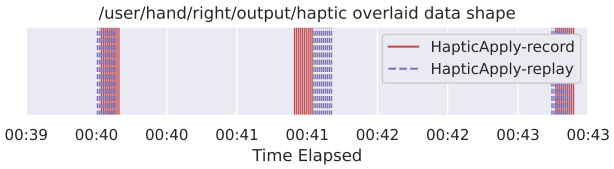

Figure 6: A view of the overlaid data of a record and replay.

4 EXPERIMENT DESIGN

Input traces were recorded mainly on two applications: Beat Saber and Phasmophobia. Beat Saber is a relatively simple game requiring little performance of the platforms. It has been optimized to run natively on VR headsets. Moreover, it is stable on most VR-PC pairs, making it a suitable application for experimenting with factors that might impact game performance without the game interfering with the measurement. Phasmophobia, on the other hand, requires more computing power from both platforms due to its more complex nature in terms of graphics and logical elements. We chose Phasmophobia as a real-world example of a game that is currently in development and might not be optimized to the full extent. If VR is going to be a real part of the gaming industry, the platforms should perform well in situations where applications run on the system don't fully use the capabilities of the system. For the lab, we are interested in running an experiment using `librnr` to experiment. In particular, we are interested in the behaviour of the VR system when it is run wireless. Here the graphics output of the game is streamed to the user as a 3D video. To test this we use a selected trace for the game we are measuring and we vary the network properties.

All measurements are done only a limited amount of times. For most tests, this will be 3-10 repeated measurements. This was chosen in the interest of time. As doing the statistically recommended 30 measurements would take around 2.5 hours per variable in the experiment space.

4.1 Power Consumption

To measure the load of gaming on battery life, we look at the game from the perspective of load on the system. In Beat Saber, we run this experiment by playing the same song of the game under different difficulty settings. This would result in a gradient of low user engagement to high user engagement and low amounts of game elements rendered on the screen to high amounts of game elements rendered on screen.

Table 2: Experiment and Validation Summary.

Section	Focus	Recording	Replay	Devices
§5.1	Validation: `librnr`	3 recordings with the layer off 3 recordings with the layer on	10 replays for each recording	PC-R tests on Quest 2 PC-D tests on Quest Pro
§5.2	Validation: Test setup	1 recording	10 replays	PC-R tests on Quest Pro PC-D tests on Quest 2
§5.3	Validation: `clumsy` accuracy	1 recording for each network modification	3 replays for each recording	PC-R tests on Quest 2 PC-D tests on Quest Pro
§5.4	Experiment: Limiting bandwidth effects on VR	1 recording each for 30, 50, 80, and 100 Mbps	3 replays for each recording	PC-R tests on Quest 2 PC-D tests on Quest Pro
§5.5	Experiment: Modifying packet drop chance effects on VR	1 recording each for 0.1%, 1%, 2.5% and 5%	3 replays for each recording	PC-R tests on Quest 2 PC-D tests on Quest Pro
§5.6	Experiment: Energy consumption	3 recordings for Beat Saber (wired) 3 recordings for Beat Saber (wireless) 1 recording for Phasmophobia (wired) 1 recording for Phasmophobia (wireless)	3 replays for each recording	PC-R tests on Quest 2 PC-D tests on Quest Pro

4.2 Network Measurements

The experiments done on the network are composed to test the headset under conditions that mimic current wireless networks. We test the performance of the headset for use in the following situations:

(1) A baseline Home network setup.
(2) A mobile network setup to mimic outdoor situations. (5g/6g) characterized by more chance for network congestion and low bandwidth
(3) A cloud-gaming setup. Characterized by higher latency and the potential for network congestion.

We test the headset performance by controlling the packet congestion, latency, and bandwidth. We use `clumsy` an open-source tool for simulating network conditions on Windows. In table 2 the experiments are listed including the parameters and test platforms.

4.3 Environment

The experimental environment consists of two different host machine and VR device combinations. Both computers in the environment are connected to the router via Wi-Fi. Moreover, both routers can communicate on 5 GHz at a speed of 1200 Mbps.

PC-D Specifications. The first setup, which will be referred to as PC-D is a Windows 11 system with a water-cooled AMD Ryzen 5 7600X CPU, a GeForce RTX 3080 GPU, and a motherboard that supports Wi-Fi 6E (802.11ax).

PC-R Specifications. The second setup, referred to as PC-R is a Windows 10 system with an air-cooled AMD Ryzen 5 7600X CPU, a GeForce RTX 4070 GPU, and a motherboard that supports Wi-Fi 6E (802.11ax).

The experiments are conducted on two VR devices: Meta Quest Pro and Meta Quest 2.

Meta Quest Pro. The Meta Quest Pro has a Snapdragon XR2+ processor, supports Wi-Fi 6E (802.11ax), and holds a 5,348 mAh rechargeable lithium-ion battery pack with a reported expected battery life of 2.6 hours under gaming load.

Meta Quest 2. The Meta Quest 2 has a Qualcomm Snapdragon XR2 processor, supports Wi-Fi 6, and holds a 3,640 mAh rechargeable lithium-ion battery pack with around 2 hours of battery life under gaming load.

5 RESULTS

In this section, we analyse the experiment data produced by the experiments proposed in Section 4. First, we validate `librnr` (Section 7), the testing environments (Section 5.2) that are used for gathering the experiment data, and `clumsy` (Section 3.4.3). Upon validation, we investigate how different network characteristics (bandwidth, and packet drop chance) impact the performance and energy consumption of the VR devices and host machines.

From these results, we have the following validation findings (VF):

VF1 `librnr` does not have noticeable overhead on the systems under stress, thus making it a viable solution for replicable VR experiments (see Section 5.1).
VF2 When `librnr` is in replay mode, the Meta Quest Pro uses less energy than in record mode (see Section 5.1).
VF3 `clumsy` is not a fully reliable tool in terms of manipulating bandwidth and packet drop characteristics, however, it works on a best-effort basis which makes it a decent solution for our purposes (see Section 5.3).

From these results, we have the following main findings (MF):

MF1 The host PC GPU has to work harder when connected to the Meta Quest Pro compared to when connected to the Meta Quest 2 (see Section 5.2).
MF2 Limiting the bandwidth on the host machine causes significant frame rate drops that impact the Meta Quest 2 device more than the Meta Quest Pro, rendering the VR experience unusable (see Section 5.4).
MF3 Increasing the drop chance on the host machine causes significant frame rate drops that impact the Meta Quest 2 device more than the Meta Quest Pro, rendering the VR experience unusable for drop chance higher than 1% on the Quest Pro, and 0.1% on the Quest 2 (see Section 5.5).
MF4 Bandwidth limitation causes contrasting behaviours for the two VR devices, the higher the bandwidth limit is for the Quest Pro, the lower the energy consumption, and vice-versa for the Quest 2 (see Section 5.6).

5.1 `librnr` Validation

Validating the overhead that `librnr` introduces over the system is crucial to derive meaningful results from the experiments that are

Table 3: An overview of all recorded VR traces.

Trace Folder Name	Game / App	Duration of trace	PC	VR	Network Conditions	Additional Info
clumsy-bandwidth-100Mbps/Quest-2	Beat Saber	229.161	PC-R	Quest 2	100 Mbps bandwidth limitation	1 recording, 3 replays
clumsy-bandwidth-100Mbps/Quest-Pro	Beat Saber	219.424	PC-D	Quest Pro	100 Mbps bandwidth limitation	1 recording, 3 replays
clumsy-bandwidth-80Mbps/Quest-2	Beat Saber	229.161	PC-R	Quest 2	80 Mbps bandwidth limitation	1 recording, 3 replays
clumsy-bandwidth-80Mbps/Quest-Pro	Beat Saber	219.424	PC-D	Quest Pro	80 Mbps bandwidth limitation	1 recording, 3 replays
clumsy-bandwidth-50Mbps/Quest-2	Beat Saber	229.161	PC-R	Quest 2	50 Mbps bandwidth limitation	1 recording, 3 replays
clumsy-bandwidth-50Mbps/Quest-Pro	Beat Saber	219.424	PC-D	Quest Pro	50 Mbps bandwith limitation	1 recording, 3 replays
clumsy-bandwidth-30Mbps/Quest-2	Beat Saber	229.161	PC-R	Quest 2	30 Mbps bandwith limitation	1 recording, 3 replays
clumsy-bandwidth-30Mbps/Quest-Pro	Beat Saber	219.424	PC-D	Quest Pro	30 Mbps bandwith limitation	1 recording, 3 replays
clumsy-dropchance-0.05/Quest-2	Beat Saber	229.161	PC-R	Quest 2	5% drop chance	1 recording, 3 replays
clumsy-dropchance-0.05/Quest-Pro	Beat Saber	219.424	PC-D	Quest Pro	5% drop chance	1 recording, 3 replays
clumsy-dropchance-0.025/Quest-2	Beat Saber	229.161	PC-R	Quest 2	2.5% drop chance	1 recording, 3 replays
clumsy-dropchance-0.025/Quest-Pro	Beat Saber	219.424	PC-D	Quest Pro	2.5% drop chance	1 recording, 3 replays
clumsy-dropchance-0.001/Quest-2	Beat Saber	229.161	PC-R	Quest 2	0.1% drop chance	1 recording, 3 replays
clumsy-dropchance-0.001/Quest-Pro	Beat Saber	219.424	PC-D	Quest Pro	0.1% drop chance	1 recording, 3 replays
clumsy-dropchance-0.01/Quest-2	Beat Saber	229.161	PC-R	Quest 2	1% drop chance	1 recording, 3 replays
clumsy-dropchance-0.01/Quest-Pro	Beat Saber	219.424	PC-D	Quest Pro	1% drop chance	1 recording, 3 replays
overhead-validation/Quest-2	Beat Saber	224.746, 229.161, 228.258	PC-R	Quest 2	-	3 baseline, 3 recordings, 10 replays per recording
overhead-validation/Quest-Pro	Beat Saber	222.230, 219.424, 217.662	PC-D	Quest Pro	-	3 baseline, 3 recordings, 10 replays per recording
swap-vr/Quest-2	Beat Saber	222.146	PC-D	Quest 2	-	1 recording, 10 replays
swap-vr/Quest-Pro	Beat Saber	233.927	PC-R	Quest Pro	-	1 recording, 10 replays

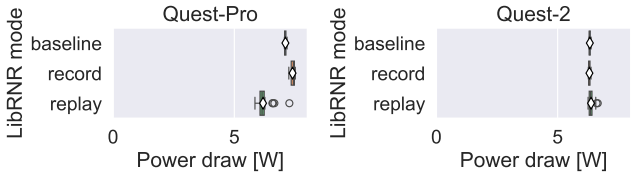

Figure 7: Validation of `librnr` energy overhead on the two VR devices.

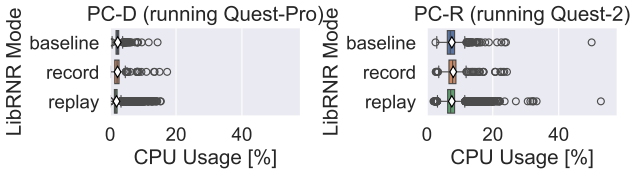

Figure 8: Validation of `librnr` CPU overhead on the two PCs. PC-D is the setup running Windows 11, PC-R is the setup running Windows 10. See §4.3 for exact specifications.

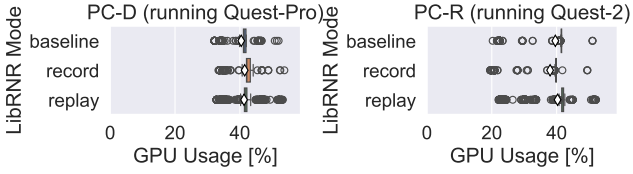

Figure 9: Validation of `librnr` GPU overhead on the two PCs. PC-D is the setup running Windows 11, PC-R is the setup running Windows 10. See §4.3 for exact specifications.

following. When talking about overhead, we refer to both overhead on the VR devices, as well as overhead on the components of the host machines.

In Figure 7, we find that from an energy standpoint `librnr` has a significant impact in replay mode on the Meta Quest Pro, consuming less power, but no impact on the power draw of the Meta Quest 2. `librnr` has no significant impact on the host-machine CPU, and `librnr` is slightly more demanding on the host machine GPU when attached to the Quest Pro while recording, whereas the opposite happens for the host-machine attached to the Quest 2. The figure shows the differences in VR device energy consumption for the three modes `librnr` can run in. The modes are *baseline*, when `librnr` is completely turned off, *record*, when `librnr` is recording the user inputs, and *replay*, when `librnr` is replaying the previously recorded user inputs. These modes are presented, in this order, from top to bottom on the vertical axis, and the horizontal axis contains the power draw information from the VR devices in Watts. The power draw metric is chosen because not all runs of the experiment have the same length. The *baseline* and *record* boxplots contain data from three different experiment runs. The *replay* boxplots summarize the replay data from 30 experiment runs (i.e., 10 runs for each of the three recordings). In the figure, we can see that the Quest Pro VR uses slightly more energy for recording compared to the baseline, and less energy for replay compared to the baseline, whereas the Quest 2 VR headset has very little variation in terms of energy consumption across the `librnr` modes. The difference in baseline and replay on the Quest Pro can be attributed to the fact that the physical controllers on the Quest Pro are not moving during replays, but rather sitting stationary, thus the Quest Pro spends less resources on tracking the controllers. Another noteworthy observation is that the baseline vs record behaviour does not occur on the Quest 2 device, confirming that the Quest 2 is using a less resource-intensive tracking solution for the physical controllers.

In Figure 8, we see almost identical behaviours in the baseline, record, and replay boxplots for each computer. The baseline and record boxplots are very similar across the runs, but the replay runs have more outliers compared to the other two `librnr` modes. This can be explained by the fact that there is about 10 times more data for the replays compared to the recordings and baseline runs. We can also see that the CPU utilisation is lower on PC-D compared to PC-R. This difference could be caused by several factors including Operating System (Windows 11 on PC-D and Windows 10 on

PC-R), different services running in the background on the two setups, peripherals connected to the PC, and even the VR devices being connected. We develop the possible explanations for this phenomenon in Section 5.2.

In Figure 9, we can see the same trends occurring, with similar GPU usage across the three `librnr` modes, and noticeably more outliers in the replay mode boxplot. The key difference that can be observed is that the GPU usage for record mode is slightly higher than baseline and replay on PC-D, whereas the opposite happens for PC-R.

These results lead us to conclude that, for the applications tested, librnr shows low overheads consistently, suggesting that it is a good tool for running replicable VR experiments.

5.2 `librnr` Experiment Setup Validation

We validate our experiment setups by swapping the VR headsets between the two PCs. As such, for this section, PC-R is connected to the Quest Pro, and PC-D is connected to the Quest 2. Through this experiment, we hope to highlight any possible erroneous data that might occur as a result of the host machines having different specifications and operating systems. Due to time constraints, this experiment has only one run in record mode and 10 runs in replay mode, and we acknowledge that the very limited recording data has a big chance of being unrepresentative of the behaviour of the host machine. To mitigate this shortcoming, we explore the findings from the replay data, as it is more reliable.

In this experiment, we find that the type of VR device does not have an impact on the CPU usage of the host machine, but it does influence the GPU usage. Figure 10 shows the CPU usage for the two PCs in record and replay mode running different VR devices. We notice that the CPU usage on both PC-R and PC-D stays consistent across recording and replaying. The main difference that can be observed in the figure is the CPU utilisation between the two PCs. We see that the lower utilisation on PC-D stays consistently lower, and the higher utilisation on PC-R stays consistently higher, just like in Figure 8. Thus, we can conclude that the VR devices do not contribute to the difference in CPU usage of the two machines.

Figure 11 shows the GPU usage for the two PCs in record and replay mode. We can see that the GPU usage stays consistent within each of the PCs that are selected. From this, we can deduce that the VR device does not impact the GPU usage consistency across record and replay. However, when comparing this figure to Figure 9, we notice big discrepancies in the GPU usage for each PC - PC-D uses less GPU resources when running the Quest 2, compared to running the Quest Pro, whereas the exact opposite behaviour appears with PC-R. This could be an indication that the Quest Pro requires more GPU resources to run a VR game compared to the Quest 2.

These results lead us to believe that the type of VR device used for experiments influences the utilisation patterns of the host machine GPU, but it does not change the CPU usage patterns.

5.3 `clumsy` Validation

We validate the capabilities of the `clumsy` software through a set of bandwidth experiments and a set of packet drop experiments. This validation will determine whether `clumsy` is a reliable tool for

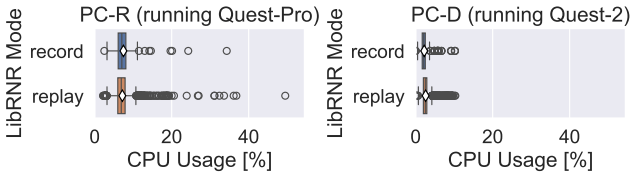

Figure 10: Setup validation of `librnr` CPU overhead on the two PCs. PC-D is the setup running Windows 11, PC-R is the setup running Windows 10. See §4.3 for exact specifications.

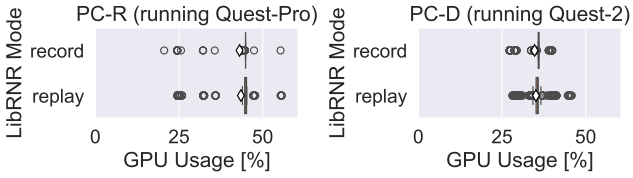

Figure 11: Setup validation of `librnr` GPU overhead on the two PCs. PC-D is the setup running Windows 11, PC-R is the setup running Windows 10. See §4.3 for exact specifications.

manipulating network characteristics, and how reliable the data that we are going to obtain from different network experiments is.

We find that `clumsy` can limit the bandwidth very reliably for the 100Mbps limit, but as we decrease the limit, more and more spikes start exceeding the limit. Manipulating the packet drop chance yielded unsatisfactory results, with none of the measured drop chance results aligning with the expected values.

Starting with the bandwidth experiments, we test four different bandwidth limit values that can be found in Figure 12. The plots show the amount of data that was received throughout the experiment runs for bandwidth limits 100Mbps, 80Mbps, 50Mbps, and 30Mbps. For each plot, the vertical axis shows the quantity of data received in Mbps, the horizontal axis shows the timestamp, the green, the dotted line shows the bandwidth limit, and the orange, dashed line shows the average Mbps received for the run. In the 100Mbps limit graph, we can see that the data received never crosses the 100Mbps limit, however, this behaviour does not translate to any of the other bandwidth limit results. For all three experiments, the limit is exceeded at least three times over the 4 minutes of runtime. This leads us to believe that even though clumsy manages to cap the bandwidth to the preferred value, it can not do this consistently throughout the entire run of the experiments.

Figure 13 shows the result of the drop chance experiment. In this experiment, we increase the chance for sent and received packets to be dropped. From 0.1% to 2.5%, the measured drop chance increases proportionally to the specified drop chance, but it decreases at 5%. Moreover, the measured drop-chance does not align with the desired drop-chance set via clumsy. We can again conclude that even though clumsy offers a best-effort solution rather than a consistent one.

In general, limiting bandwidth and adjusting packet drop-chance resulted in at least three concrete system warnings, as indicated by the network logs:

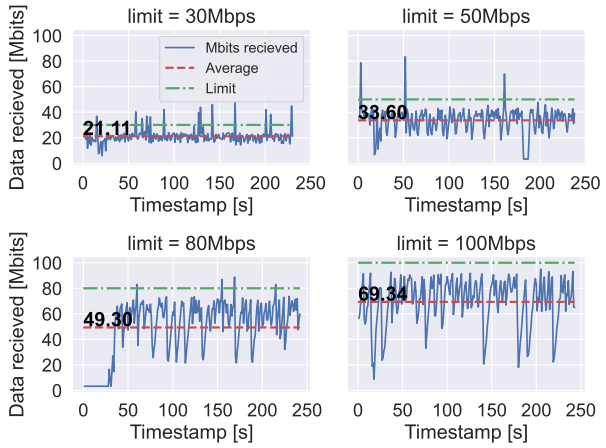

Figure 12: Clumsy bandwidth limit validation on the received bytes per second of the VR System.

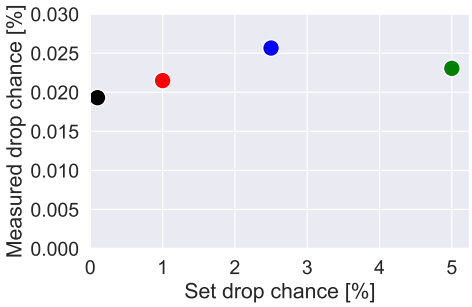

Figure 13: Clumsy packet drop chance validation on outgoing VR System packets.

- Max Future Display Vsync Exceeded - Vertical synchronization (vsync) is a bound used to synchronize the frame rate of a game or application with the refresh rate of the VR display. The warning indicates that the application is attempting to schedule frames for display at times/rates that are beyond what the system can handle.
- FrameIndex Discontinuity - This error indicates that there is a mismatch between the expected and received frames, meaning that they are likely "tripping over themselves" in the network.
- Clamping of Vsync Values - This warning suggests that the system is intervening to limit the vsync values submitted by the application to stay within an acceptable range.

We conclude from these results that `clumsy` is very unreliable concerning manipulating the drop chance of the Windows operating system, but usable for bandwidth limitation experiments.

5.4 Bandwidth Limit Effects

Limiting the bandwidth on the host machine causes significant frame rate drops that impact the Meta Quest 2 device more than the Meta Quest Pro, rendering the VR experience unusable.

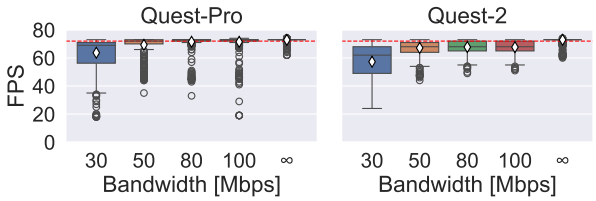

Figure 14: Effect of bandwidth limits on VR frames per second (FPS). Higher is better, the red dotted line marks the minimum recommended frame rate (i.e., 72 frames per second).

Figure 14 shows the distribution of the frame rate for the different bandwidth limits for the two VR devices. The vertical axis shows the frames per second displayed by the VR device, and the horizontal axis shows the different bandwidth limits chosen for the experiments. The white diamonds show the mean frame rate for the respective bandwidth, and the red dotted line marks the minimum frame rate for a good user experience. We notice that as the bandwidth values decrease, so do the mean FPS values, the lower the bandwidth limit, the more difficult it becomes for the user to play the game. The FPS distribution for the unbounded experiment has outliers below the 72 FPS mark, but these outliers most likely occur during the loading screens of the game, thus not impacting the user experience. In all other cases, the outliers reach very low FPS values which leads us to believe that it is not a case of the loading screen event. Comparing the behaviour of the two devices, we notice that for the 100Mbps, 80Mbps, and 50Mbps limits, the Quest Pro can handle the bandwidth limits much better than the Quest 2, with the average FPS for the Quest Pro being closer to the ideal 72 FPS mark, but with a larger spread in terms of outliers. The Quest 2 stays very consistent across the upper limits, with almost identical boxplots, but despite this consistency, the VR experience is unplayable due to the low average FPS. The 30Mbits bandwidth limit causes both devices to struggle in terms of frame rate consistency. This phenomenon occurs due to the host machine not being able to send enough information to the VR headset. As such, the VR device cannot fill in the frames that are not received, causing the playback to be stuttering and unpleasant for the user. A possible explanation for the better-performing Quest Pro might be a higher-quality network card on the VR device, or a piece of software that can do frame prediction, and can fill in some of the missing frames.

In light of these results, we conclude that bandwidth limits have strong negative effects on the frame rate of the VR device that renders the VR experience unplayable for users.

5.5 Packet Drop Chance Effects

Increasing the drop chance on the host machine causes significant frame rate drops that impact the Meta Quest 2 device more than the Meta Quest Pro, rendering the VR experience unusable for a drop chance higher than 1% on the Quest Pro, and 0.1% on the Quest 2.

Figure 15 shows the distribution of the frame rate for the different drop chance values for the two VR devices. The vertical axis

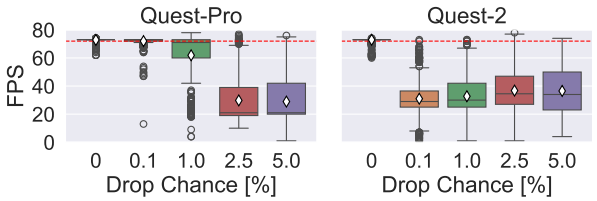

Figure 15: Effect of packet drop chance on VR frames per second (FPS). Higher is better, the red dotted line marks the minimum recommended frame rate (i.e., 72 frames per second).

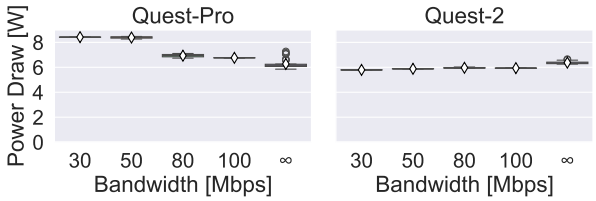

Figure 16: Effect of bandwidth limitation on the power consumption of the VR devices.

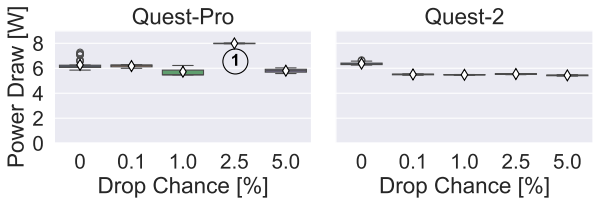

Figure 17: Effect of packet drop chance on the power consumption of the VR devices.

shows the frames per second displayed by the VR device, and the horizontal axis shows the different drop chance values chosen for the experiments. The white diamonds show the mean frame rate for the respective bandwidth, and the red dotted line marks the minimum frame rate for a good user experience.

The distributions for the Quest Pro show a decrease in frame rate as the drop chance probability increases. The Quest Pro can handle a drop chance of 0.1% with a significant amount of outliers below the ideal 72 FPS threshold, but the same cannot be said for the Quest 2. The Quest 2 frame rate decreases dramatically at 0.1% drop chance and increases slightly the higher the drop chance values. This unexpected behaviour is most likely caused by `clumsy` not being consistent with the rate at which packets are dropped on the PC-VR communication pipeline. Even with this limitation, we can confidently conclude that altering the packet drop chance on the host machine negatively impacts the wireless VR headsets to the point of rendering the VR experience unplayable.

5.6 Energy Consumption

Bandwidth limitation causes contrasting behaviours for the two VR devices, the higher the bandwidth limit is for the Quest Pro, the lower the energy consumption, and vice-versa for the Quest 2.

Figure 16 shows the energy consumption for the two VR devices under the different bandwidth limitations. The vertical axis shows the power draw of the device in watts, the horizontal axis shows the different bandwidth limits chosen for the experiments, and the white diamonds mark the mean power draw for each corresponding bandwidth. The power draw of the Quest Pro stays around 8.2W for the 30Mbps and 50Mbps limits and then decreases to around 6W as the bandwidth limit increases. We can see the opposite of this behaviour exhibited by the Quest 2 device, consuming the least average power at the 30Mbps limit, with the average power increasing with the bandwidth limit. The behaviour of the Quest Pro device could be explained by the CPU working harder to predict the frames that have not arrived yet from the host machine. The Quest 2 behaviour could be due to the frame predict functionality not being available on the headset, thus less energy is being used when fewer frames are received. It can be concluded that for a good and energy-efficient VR experience, the bandwidth from the host machine to the VR headset should be at least 100Mbps.

In terms of energy efficiency, manipulating the packet drop chance probability yields similar results to limiting the bandwidth on the connection between the VR and the host machine.

Figure 17 shows the energy consumption for the two VR devices under the different packet drop chance probabilities. The vertical axis shows the power draw of the device in watts, the horizontal axis shows the different packet drop chance probabilities chosen for the experiments, and the white diamonds mark the mean power draw for each corresponding drop chance. The Quest Pro device appears to have a chaotic pattern in terms of energy consumption, especially when looking at the 2.5% drop chance (see ①). But if we consider the `clumsy` drop chance patterns from Section 5.3 (drop chance 2.5% has the highest measured drop rate), and the possible explanation for the high Quest Pro energy consumption in the bandwidth limitation experiment (the lower the bandwidth, the higher the energy consumption), we can argue that a packet drop chance of 2.5% might have the same effect of energy consumption as limiting the bandwidth. As a result, it is very likely that for a drop chance equal to 2.5%, the Quest Pro does not get enough frames from the host machine, and starts using the energy-intensive frame prediction program. The results for the Meta Quest 2 are in line with the bandwidth results - the energy use increases along with the quality of the network. Thus, we can conclude that, in terms of energy efficiency, packet drop chance probability affects the VR devices similarly to bandwidth limitation.

6 LIMITATIONS

This study aims to analyze collected VR traces. In this section, we present the limitations that our study poses.

Types of Applications. This experiment only studies the traces collected from one VR application: Beat Saber. Therefore, the generalizability of the traces is limited to gaming applications.
Hardware. The VR devices that are used in this experiment are the Meta Quest 2 and Meta Quest Pro. Therefore, conducting the same experiment on different VR headsets connected to different setups could yield biased results regarding reliability and accuracy.
Number of Repetitions. The experiments are conducted a limited number of times which may impact the accuracy of our results. The number of recordings and replays can be seen in Table 2.
Record 'n Replay (for VR) (`librnr`*).* `librnr` has some functional shortcomings. In particular, VR-hybrid applications have unpredictable behaviour. If the VR connection and rendering are activated during the runtime of a game this will have an impact on the usefulness of replaying and recording as soon as the `librnr` API layer gets loaded. For a more consistent experience for the researcher, more extensive record and replay functionality could be added to the tool. This should include a way to start, stop, and restart a replay while the API layer is activated. These functionalities should also include starting and stopping a recording while the game is running. Lastly, having more control over the timestamp used for replaying would be useful to researchers as currently the raw frame time is used.
`clumsy` *tool.* In this experiment, we use `clumsy` to control the network conditions. However, the tool presents certain limitations that in turn affect the results of our research. These include inconsistencies in drop chance rate and specific bandwidth limitations.

7 CONCLUSION AND ONGOING WORK

Recent advancements in the computer industry have made the concept of the metaverse more feasible than ever. Nevertheless, there remains a prominent research gap regarding the performance and energy efficiency of VR devices under different network conditions. In this work, we aim to investigate the replicability of virtual reality (VR) traces and analyze their behaviour under various network characteristics and their energy consumption. We conduct this study on two of Meta's VR headsets: Meta Quest Pro and Meta Quest 2. First, our results show that the tools developed during this study improve the replicability and efficiency of VR experiments. Second, through our experiments, we present some of the first results in the field regarding VR performance, and the first results that tackle energy efficiency of VR headsets running over wireless connection.

In future work, we aim to improve our research results by using more reliable tools and addressing the limitations in Section 6, as well as further perfecting the tools developed throughout this study to integrate with more research-oriented operating systems (i.e., Linux), and to improve the capabilities of Record 'n Replay (for VR) (`librnr`) to allow for synchronization.

8 TIME SHEET

Time log can be found here: https://docs.google.com/spreadsheets/d/1G-r5rLODq__YRgmBpehdPgJzc-rCKiEk0l18ixEVxWo/edit?usp=sharing.

REFERENCES

[1] Pooya Adami, Patrick B. Rodrigues, Peter J. Woods, Burcin Becerik-Gerber, Lucio Soibelman, Yasemin Copur-Gencturk, and Gale M. Lucas. 2021. Effectiveness of VR-based training on improving construction workers' knowledge, skills, and safety behavior in robotic teleoperation. *Adv. Eng. Informatics* 50 (2021), 101431. https://doi.org/10.1016/J.AEI.2021.101431

[2] Halit Apaydin, Hajar Feizi, Mohammad Taghi Sattari, Muslume Sevba Colak, Shahaboddin Shamshirband, and Kwok-Wing Chau. 2020. Comparative Analysis of Recurrent Neural Network Architectures for Reservoir Inflow Forecasting. *Water* 12, 5 (2020). https://doi.org/10.3390/w12051500

[3] Apple. 2023. Introducing Apple Vision Pro: Apple's first spatial computer. https://www.apple.com/newsroom/2023/06/introducing-apple-vision-pro-apples-first-spatial-computer Accessed 2024-01-04.

[4] Patrick Carnahan, John Moore, Daniel Bainbridge, Gavin Wheeler, Shujie Deng, Kuberan Pushparajah, Elvis C. S. Chen, John M. Simpson, and Terry M. Peters. 2020. Applications of VR medical image visualization to chordal length measurements for cardiac procedures. In *Medical Imaging 2020: Image-Guided Procedures, Robotic Interventions, and Modeling, Houston, TX, USA, February 15-20, 2020 (SPIE Proceedings)*, Baowei Fei and Cristian A. Linte (Eds.), Vol. 11315. SPIE, 1131528. https://doi.org/10.1117/12.2549597

[5] Jesse Donkervliet, Matthijs Jansen, Animesh Trivedi, and Alexandru Iosup. 2023. Can My WiFi Handle the Metaverse? A Performance Evaluation Of Meta's Flagship Virtual Reality Hardware. In *Proceedings of the International Conference on Performance Engineering, Coimbra, Portugal, April, 2023.*

[6] Mark Heider Draper and Thomas A. Furness. 1998. *The Adaptive Effects of Virtual Interfaces: Vestibulo-Ocular Reflex and Simulator Sickness.* Ph.D. Dissertation. USA. AAI9828482.

[7] Paul M.G. Emmelkamp and Katharina Meyerbröker. 2021. Virtual Reality Therapy in Mental Health. *Annual Review of Clinical Psychology* 17, 1 (2021), 495–519. https://doi.org/10.1146/annurev-clinpsy-081219-115923 arXiv:https://doi.org/10.1146/annurev-clinpsy-081219-115923 PMID: 33606946.

[8] Beat Games and Khronos Group. 2023. Keeping the Beat: Porting Beat Saber to OpenXR for An Improved Developer Experience. https://www.khronos.org/blog/keeping-the-beat-porting-beat-saber-to-openxr-for-an-improved-developer-experience:https://www.khronos.org/blog/keeping-the-beat-porting-beat-saber-to-openxr-for-an-improved-developer-experience Accessed 2024-01-04.

[9] Yong Guo and Alexandru Iosup. 2012. The Game Trace Archive. In *11th Annual Workshop on Network and Systems Support for Games, NetGames 2012, Venice, Italy, November 22-23, 2012.* IEEE, 1–6. https://doi.org/10.1109/NETGAMES.2012.6404027

[10] Business Insider. 2022. Charts: Meta's Metaverse Spending Losses, Reality Labs, VR, Mark Zuckerberg. https://www.businessinsider.com/charts-meta-metaverse-spending-losses-reality-labs-vr-mark-zuckerberg-2022-10 Accessed 2024-01-04.

[11] Meta. 2023. Meet Meta Quest 3, Our Mixed Reality Headset Starting at $499.99. https://about.fb.com/news/2023/09/meet-meta-quest-3-mixed-reality-headset Accessed 2024-01-04.

[12] Markus Murtinger, Jakob Carl Uhl, Helmut Schrom-Feiertag, Quynh Nguyen, Birgit Harthum, and Manfred Tscheligi. 2022. Assist the VR Trainer - Real-Time Dashboard and After-Action Review for Police VR Training. In *IEEE International Conference on Metrology for Extended Reality, Artificial Intelligence and Neural Engineering, MetroXRAINE 2022, Rome, Italy, October 26-28, 2022.* IEEE, 69–74. https://doi.org/10.1109/METROXRAINE54828.2022.9967532

[13] SteamDB. 2024. Most Played VR Only Games. https://steamdb.info/charts/?category=54 Accessed 2024-01-07.

[14] Laurens Versluis, Roland Mathá, Sacheendra Talluri, Tim Hegeman, Radu Prodan, Ewa Deelman, and Alexandru Iosup. 2020. The Workflow Trace Archive: Open-Access Data From Public and Private Computing Infrastructures. *IEEE Trans. Parallel Distributed Syst.* 31, 9 (2020), 2170–2184. https://doi.org/10.1109/TPDS.2020.2984821

[15] Paul Zikas, Antonis Protopsaltis, Nick Lydatakis, Mike Kentros, Stratos Geronikolakis, Steve Kateros, Manos Kamarianakis, Giannis Evangelou, Achilleas Filippidis, Eleni Grigoriou, Dimitris Angelis, Michail Tamiolakis, Michael Dodis, George Kokiadis, John Petropoulos, Maria Pateraki, and George Papagiannakis. 2023. MAGES 4.0: Accelerating the World's Transition to VR Training and Democratizing the Authoring of the Medical Metaverse. *IEEE Computer Graphics and Applications* 43, 2 (2023), 43–56. https://doi.org/10.1109/MCG.2023.3242686