BACHELOR THESIS

# Networked Physics Capabilities of Unity XR Projects: A Comparison of Object-Orientated and Component-Based Architectures

*Author:*
Florian DEMMLER

*Supervisors:*
Florian SCHIER
Kelsang MENDE

*Examiner:*
Jun. Prof. Dr. Matthew MCGINITY
Dr.-Ing. Ingmar FRANKE

*A thesis submitted in fulfillment of the requirements for the degree of Bachelor of Science*

*in the*

Junior Professorship In Immersive Media
Institute of Software and Multimedia Technology.


*Submitted:* September 8, 2024 *Version:* July 10, 2026

*"Do not go where the path may lead, go instead where there is no path and leave a trail. "*

Ralph W. Emerson

*"The development of immersive media is as much about understanding how we perceive the real world, as it is about technology. "*

Matthew McGinity

TECHNICAL UNIVERSITY OF DRESDEN

# *Abstract*

Institute of Software and Multimedia Technology.
Junior Professorship In Immersive Media

Bachelor of Science

**Networked Physics Capabilities of Unity XR Projects: A Comparison of Object-Orientated and Component-Based Architectures**

by Florian DEMMLER

The increasing prevalence of Extended Reality (XR) applications, particularly those incorporating networked physics simulations, necessitates robust and efficient networking solutions. The seamless integration of real-time physics interactions in shared virtual environments presents a considerable technical challenge, especially concerning latency, synchronization, and bandwidth utilization. The choice of networking architecture and framework significantly impacts the overall performance and user experience.

This thesis investigates the capabilities of Unity's networking libraries, Netcode for GameObjects (NGO) and Netcode for Entities (NCE), in facilitating real-time physics-based interactions within XR applications. The research primarily focuses on the technical intricacies and usability of these libraries, with a particular emphasis on their effectiveness in managing networked physics simulations. The study delves into the theoretical foundations of NGO and NCE, comparing their object-oriented and data-oriented paradigms, and highlighting their respective strengths and weaknesses in handling physics-based interactions. The core of the investigation lies in technical measurements conducted within a controlled laboratory environment, specifically examining the motion-to-photon (MTP) latency and network throughput, critical metrics for real-time responsiveness and efficient data transmission in XR. The impact of server load and varying network configurations on these metrics is thoroughly analyzed, providing insights into the scalability and performance limitations of these libraries. Furthermore, the thesis derives and calculates theoretical throughput requirements for various XR scenarios, highlighting the bandwidth challenges associated with networked physics simulations.

The findings of this thesis offer valuable guidance for developers navigating the complexities of networked physics in XR. The identified challenges and limitations underscore the need for careful consideration when selecting and implementing networking solutions for XR applications, particularly those involving physics-based interactions. The insights gained from this research can contribute to the development of more immersive, interactive, and realistic XR experiences that seamlessly blend the physical and virtual worlds, while efficiently utilizing available network resources.

# *Acknowledgements*

I would like to express my gratitude to my supervisors Florian SCHIER and Kelsang MENDE for providing me with the opportunity to delve into this subject and allowing me to contribute my own insights. I am particularly grateful for their ongoing assistance and encouragement throughout the entire process.

I would also like to acknowledge the inspiring and motivating seminar delivered by Professor Matthew MCGINITY during the winter term 2022/23. This seminar significantly deepened my understanding of the significance of scientific work and the contemporary challenges it faces.

Lastly, I want to express my gratitude to all those who have offered me encouragement and support throughout this thesis endeavor.

# Contents

# List of Figures

# List of Tables

# List of Abbreviations

| | |
|---|---|
| **HMD** | **H**ead **M**ounted **D**isplay |
| **VR** | **V**irtual **R**eality |
| **AR** | **A**ugmented **R**eality |
| **MR** | **M**ixed **R**eality |
| **XR** | E**x**tended **R**eality |
| | |
| **NGO** | **N**etcode for **G**ame**O**bjects |
| **NCE** | **N**et**C**ode for**E**ntities |
| **ECS** | **E**ntity-**C**omponent-**S**ystem |
| | |
| **LAN** | **L**ocal **A**area **N**etwork |
| **WLAN** | **W**ireless **L**ocal **A**area **N**etwork |
| **MAN** | **M**etropolitan **A**area **N**etwork |
| **WAN** | **W**ide **A**area **N**etwork |
| **GAN** | **G**locabl **A**area **N**etwork |
| | |
| **TCP** | **T**ransmission **C**ontrol **P**rotocol |
| **UDP** | **U**ser **D**atagram **P**rotocol |
| **RPC** | **R**emote **P**rocedure **C**all |
| | |
| **TTL** | **T**ume **T**o **L**ive |
| **RTT** | **R**ound **T**rip **T**ime |
| **QoS** | **Q**uality **O**f **S**ervice |
| **Cat. N** | **C**ategory **N** |
| | |
| **FPS** | **F**rames **P**er **S**econd |
| **MTP** | **M**otion **T**o **P**hoton Latency |
| **ROI** | **R**egion **O**f **I**nterest |
| | |
| **CSP** | **C**lient-**S**ide **P**rediction |

*Dedicated to P.B.*

# Chapter 1

# Introduction

The convergence of real-time physics simulations with extended reality (XR) technologies, including virtual reality (VR), augmented reality (AR) and mixed reality (MR), has opened up new possibilities for immersive, interactive experiences. XR applications are increasingly being used across diverse industries, from gaming and entertainment to education, healthcare, and engineering. One of the most compelling aspects of these applications is the ability to simulate and interact with realistic physical environments in real time. However, the implementation of real-time physics in networked XR environments remains a significant technical challenge, particularly due to latency issues and the inherent complexity of synchronization.

The need for precise and consistent physical interactions across multiple devices or users in a shared virtual environment brings latency to the forefront of development concerns. Even minor delays can lead to inconsistencies in the physics simulation, causing objects to behave unpredictably or unrealistically. This discrepancy not only disrupts the immersive experience but can also lead to significant usability and safety concerns, particularly in professional or educational applications where accuracy is critical.

Moreover, the complexity of integrating networked real-time physics in XR is exacerbated by the varying computational capacities of devices and the fluctuating quality of network connections. Achieving a seamless, synchronized experience across platforms requires advanced techniques in predictive modeling, interpolation, and error correction such as client-side prediction (CSP). Equally important is the selection of the appropriate networking architecture and framework, as it directly influences the overall performance and consistency of the system. Developers must be aware of the trade-offs associated with different architectures, such as server-client or peer-to-peer models, as each presents its own set of advantages and disadvantages in terms of scalability, latency management, and fault tolerance.

In this thesis, these challenges will be examined within the context of Unity's network solutions, specifically the high-level server authoritative networking libraries "Netcode for GameObjects" (NGO) and "Netcode for Entities" (NCE). These frameworks provide different approaches to networked physics simulations, with NGO focusing on a more traditional, object-oriented paradigm and NCE offering a more performance-optimized, data-oriented solution for large-scale simulations by an component-based entity-component-system (ECS). By analyzing metrics such as bandwidth usage and latency across both frameworks, this thesis aims to shed light on their respective strengths and limitations in networked real-time XR applications. Understanding how these architectures handle the complexities of networked physics will provide valuable insights into optimizing XR experiences for diverse use cases and ultimately improving user experience.

Balancing computational performance, network efficiency, and physics accuracy

is a delicate challenge, and resolving these issues is crucial for the future development of XR applications. Due to the scarcity of scientific resources and the limited availability of literature specifically addressing the performance of Unity's networking libraries, this thesis adopts an inductive approach to investigate the technical details and usability of both libraries in the context of XR applications.

## 1.1 Economical Growth and Importance

Industry 4.0, a concept unveiled at the 2011 Hannover Messe, describes the "fourth industrial revolution" and a strategic plan to strengthen Germany's international competitive position in manufacturing. This terminology has received international recognition and is associated with Germany.[1] Today it stands for the digitization of the industry.[2] There is a sense of confusion when approaching a clear definition for Industry 4.0, because of its ideas overlapping to other concepts. Key enabling technologies like the internet of things, cloud computing, big data analytics, machine learning and interoperability/cybersecurity solutions are already in our daily lives and are continuously improved and developed. [4] Therefore, it is anticipated that there will be a further increase in the usage and demand for networking technologies in our increasingly globalized world.

Immersive technologies such as AR, 3D-Services and multi-user interactions are one important emerging branch of this concept. The potential applications of augmented reality are outlined in the 5G white paper, published on February 17, 2015. It is expected that these technologies will play a crucial role beyond 2020. Further, this white paper primarily focuses on economic applications, emphasizing the potential for a fully mobile "smart life" to transform daily work environments. It envisions scenarios such as video communication with extremely high resolution in settings like 3D cyber-real offices or operating rooms. Additionally, the paper discusses entertainment applications, such as integrating physical and virtual information-like scores or details about athletes and musicians, during events at stadiums or open-air gatherings. [1] Westphal [5] further elaborates on various use cases for AR/VR, including office productivity, museums, education, sports, gaming, medical applications, and maps, emphasizing the broad potential of these technologies in diverse sectors. He notes that there are numerous other potential applications, highlighting the significant importance of this technology.

These different scenarios bring with them a variety of issues that need to be addressed. Technological knowledge coupled with the awareness for possible points of failures are required to manage and create viable solutions. For example, superimposing data onto the real world can enhance task completion speed and reduce errors. However, the task performance in augmented reality applications depends on the complexity and nature of the task, making the selection of a suitable task crucial. Masood and Egger [11] further found, that field experiments revealed, that the user's task performance in AR systems varied based on workers' expertise and experience. Untrained workers had the most gain and task performance improvements while already experienced workers could not improve much (Lower for experts and higher for untrained workers). It is important to note that the consideration of technological aspects is highly relevant. However, organizational issues are of fundamental

[1] https://www.dfki.de/web/news/zehn-jahre-industrie-4-0/ (retrieved 11.03.2024, 17:44)

[2] https://www.bmbf.de/bmbf/de/forschung/digitale-wirtschaft-und-gesellschaft/industrie-4-0/industrie-4-0 (retrieved 11.03.2024, 17:45)

importance, particularly during the initial use phase of new technologies. This is especially true for technologies that are to be used in economic environments, as this can promote further development of the technology through investment.

As capabilities have been raised to consumer level and reached the gaming sector, the demand further grows since. The common interest for such devices was increasing as the devices got more advanced and more commercially accessible. As the network infrastructure expanded, the conditions of multi-user XR experience improved. As more and more applications were discovered, their use increased. It is no coincidence that sthey have been described as the key enabling technology for the next industrial generation. Today, the gaming and entertainment sectors, followed by industrial interest which includes research, are the main drivers of this thriving development. The size of the VR market is expected to grow from 12.3 billion US$ in 2023 to 51.5 billion US$ in 2030.[3]

The exploration of immersive media has been a cornerstone of scientific inquiry for a long time, with their unique capacity to simulate real-world experiences making them invaluable for experimental research methods. This has enabled the investigation of human perception under controlled conditions. As the field has matured, so too has the number of studies, reflecting a growing interest and potential of immersive technologies. [18]

## 1.2 Immersive Technology

The efficacy of such technologies can be attributed to their capacity to immerse the user in a simulated environment that closely resembles the real world. Consequently, these technologies are often referred to as "immersive technologies". In more detail, the essential premise of immersive technology states, that if someone perceives representations in a similar manner as he would were they real and present, then an experience of them as real and present will follow. This is called immersion. [13] In summary, immersive technology aims to provide a lifelike experience through lifelike perception. However, it is neither necessary nor feasible to recreate all stimuli. Instead, we can focus on identifying the salient features of the surrounding world and our perceptual relationship with it. Therefore, the essence of virtual reality lies in recreating the salient features of the world and our perception of ourselves within it. The development of immersive media involves understanding how we experience and perceive the real world. It is important to note that a single concept of immersion does not exist; rather, there are different sub-immersions or aspects. [12]

For example objects can be related to aspects such as spatial presence, cognitive/perceptual realness, or the sense of agency. Spatial presence is multifaceted, encompassing the sense of opportunity for movement and action, independence, and a coherent, unified world. In summary, the imperceptibility of the real world is the main idea. Moreover, the behavior of an object is subject to cognitive realness. This is about the plausibility and veridicality of an object, which should respond as if it were real. The sense of agency refers to the ability to perceive causalities and connections between objects, such as collisions. [12] it is these diverse aspects of perceptual experience that makes us particularly sensitive to any unnatural alterations in the world and its behavior. If our's individual immersion is disturbed, users may feel uncomfortable and unable to accurately describe their experience. Motion sickness or cybersickness may occur, resulting in symptoms such as fatigue,

[3] https://www.globaldata.com/store/report/vr-market-analysis/ (retrieved 11.03.2024, 17:51)

disorientation, and nausea, which can undermine the user experience. Up to 40% of users may experience VR motion sickness, which can be triggered by a combination of factors. Triggers for discomfort often relate to camera or headset movement. Fast camera translational movement speeds and excessive camera acceleration are problematic. Additionally, excessive head movement may cause postural instability or vestibular-ocular reflex maladaptation, increasing discomfort. Intensive motion flows are also a major factor in causing simulator sickness. [20]

## 1.3 Immersive Real-Time Physics-Based Interactions and Simulation

Real-time physics refers to the study and application of physical principles as they occur in real-world scenarios, often using simulations or interactive models. In daily life, we constantly and often unconsciously engage with the physical world - often through simple actions like walking, jumping, or drinking water from a glass. These fundamental interactions become challenging to replicate in virtual environments due to the limitations of computational power, requiring the use of various techniques to create a realistic experience.

Our sensitivity to these interactions is heightened by our ingrained perceptions and expectations of how the physical world should behave, making the accurate simulation of physics in immersive media both a critical and delicate task. In the context of XR applications, it is essential that real-time physics simulations are completed by the beginning of each frame. However, this is not always feasible during computationally intensive moments, which can lead to significant issues.

Typically, there are two approaches to implementing real-time physics. The first approach involves completing the next simulation step within the current frame, which can severely impact frame rates as the system waits for the computation to finish. A drop in frame rates can disconnect the user from the immersive experience, potentially leading to motion sickness.

The second approach calculates simulation steps in parallel, independently of the frame rate, at predefined intervals. A minimum time step is established to determine the smallest possible interval for simulation calculations. Additionally, a maximum allowable time step is set, beyond which a simulation step will be aborted if it takes too long. This method is commonly used in development environments like Unity. However, parallel computation introduces a discrepancy between the application's update rate and the physics system's update rate, potentially leading to latency of up to one physics frame. Since partial simulation steps cannot be processed, the extra time is stored in a "time bank." Once this accumulated time exceeds the application's update rate, it is processed, meaning the physics simulation is effectively running in the past, anywhere from 0 to just under 1 frame.[4] Although this delay is usually negligible, it can become problematic if the physics framerate is particularly low.

One of the challenges with this approach is that as the time intervals decrease, the computational demand increases. The time interval must be aligned with the maximum achievable refresh rate of the interface being used, such as an head-mounted display (HMD), and should always be less than or equal to this rate. Two primary issues arise when simulation steps are dropped:

[4] https://docs.unity3d.com/Manual/TimeFrameManagement.html (retrieved 31.08.2024, 18:58)

1. The delay in the movement of physical objects, which can result in theoretical "interpenetration" of matter within the same space or delayed force transmission during collisions, both of which are physically impossible.

2. Perceptible jumps or abrupt movements from the user's perspective when a simulation step is skipped due to exceeding the maximum time interval. These disruptions contradict the user's understanding of inertia and momentum.

These problems can significantly disrupt and overwhelm the user's spatial presence, cognitive/perceptual realism, and sense of agency, ultimately leading to motion sickness or a disconnect from the immersive application. While methods such as interpolation and prediction can be used to fill in missing simulation steps, they add computational overhead and are only beneficial if the minimum time interval for simulation steps is reduced sufficiently so that the additional computational load does not negate the performance gains.

## 1.4 Networked Physical Interactions

Networked physical interactions in XR applications are central to the immersive experience, but they also present significant challenges due to network latency. Networking, or computer networking, involves the exchange of data between nodes over a shared medium, which can be physical (like cables) or logical (like wireless connections). The effectiveness of these interactions largely depends on minimizing network latency, which is the delay between sending a data packet from one point and receiving it at another.

Network latency is influenced by several factors, including the type of shared medium and the distance the data must travel. In XR applications, where users interact with virtual environments in real-time, even small delays can disrupt the sense of presence and immersion. Latency becomes particularly problematic when multiple HMDs are connected wirelessly to a router or access point, as the shared medium in wireless local area networks (WLANs) is prone to interference and congestion. Conversely, wired connections, often used in setups where a PC is connected to a router via an Ethernet cable, tend to offer lower latency and more reliable performance. However, the trade-off is the reduced mobility of the user, which can be a limitation in certain XR scenarios.

The complexity of networked physical interactions is further compounded in multi-user XR environments, where synchronization between different users' actions is crucial. For instance, in collaborative virtual workspaces or multiplayer games, any delay in one user's physical interaction can lead to noticeable discrepancies, reducing the overall experience's quality. This issue is particularly acute in scenarios involving hand-eye coordination, such as manipulating virtual objects with hand gestures. Humans have highly developed expectations for how physical interactions should unfold based on their real-world experiences, making them particularly sensitive to delays in these interactions. When a user's actions in the virtual world do not match the expected response time, it can lead to a disconnect, often manifesting as motion sickness or a break in immersion. On the other hand, observers, users who are not actively interacting but rather watching the interaction, are less affected by these latencies. This is because observers are not directly controlling the interaction, so their cognitive load and expectation are lower, resulting

in reduced susceptibility to motion sickness. Therefore, the primary focus in optimizing network latency should be on the interacting users, where the impact is most significant. [16]

To mitigate these challenges, an early and thorough requirement analysis is essential when designing XR applications. This analysis helps in identifying potential latency issues and synchronization challenges before they become critical, saving time and costs during development. Additionally, choosing the right hardware and network architecture is crucial, as these factors often dictate the upper limits of what can be achieved in terms of latency and overall performance. For example, leveraging high-performance routers with advanced wireless standards (such as Wi-Fi 6e) can significantly reduce latency, improving the user experience.

## 1.5 Unity - Developing XR Applications

Unity, first released in 2005 as a game engine for Mac OS, has evolved into one of the most prominent platforms for creating immersive media, particularly within the realm of XR. Given the complexity and time-intensive nature of developing immersive applications, a real-time 3D development environment like Unity is crucial. It currently supports over 25 platforms, making it highly versatile for developers targeting various devices. Unity, which self-proclaims to be the "world's leading platform for creating and operating interactive, real-time 3D (RT3D) content," holds a significant market share in the VR industry. Data from Steam shows that over 60% of the top-grossing VR experiences in 2023 were built using Unity. This dominance extends to the broader gaming market, with at least 29% of the top 1,000 PC games on Steam being Unity-powered.[5]

The engine is especially valued for its ease of use, which facilitates a smoother learning curve for beginners while also optimizing workflows for professionals. This focus on productivity and user-friendliness enables developers to effectively harness the platform's extensive tools and features.

A key strength of Unity lies in its robust support for XR development. Unity provides extensive resources, including XR courses, documentation, tools, and templates, which are particularly beneficial for developers who are new to immersive media.[6] The availability of Software Development Kits (SDKs) from leading companies such as Microsoft, Meta, and PlayStation further simplifies the initial setup process, allowing developers to integrate XR capabilities into their projects with relative ease.

In terms of physics simulation, Unity's engine is well-equipped to handle realistic physical interactions, which are essential for creating believable XR experiences. For those requiring even more advanced physics capabilities, Unity offers access to the Havok physics engine within its data-oriented technology stack, though this feature is available only through a paid plan.[7] This level of physics simulation is crucial for ensuring that user interactions within XR environments feel natural and immersive.

[5] https://unity.com/de/our-company (retrieved 31.08.2024, 21:47)
[6] https://unity.com/de/solutions/xr (retrieved 31.08.2024, 21:54)
[7] https://docs.unity3d.com/Manual/com.havok.physics.html (retrieved 31.08.2024, 22:16)

While Unity's networking capabilities were previously limited, the introduction of "Netcode for GameObjects" and "Netcode for Entities" in 2022 marked a significant leap forward.[8] These advanced networking libraries streamline the development process for networked applications in XR (Extended Reality). By abstracting the complexities of low-level networking code, they allow developers to prioritize crafting engaging multi-user experiences. Furthermore, the "Multiplayer Tools"[9] package expands the Unity development environment by providing additional profiler modules, including bandwidth analysis of network traffic. This functionality, coupled with the built-in network simulator, empowers developers to test their applications in simulated real-world scenarios. This enables them to identify and address potential network issues before deploying their XR projects.

Furthermore, Unity Teams facilitates collaborative development by allowing small teams to store, share, and synchronize their projects in a cloud-hosted environment. Overall, Unity's rich feature set, extensive platform support, and robust tools make it a leading choice for developers focused on creating advanced, networked, and immersive XR applications.

## 1.6 Research Challenge

The existing literature on networked simulated physics in XR applications using Unity is limited. Unity provides no performance benchmarks or detailed documentation on its networking libraries, particularly regarding network packet traffic, structure, or size, even after extensive investigation. The lack of comprehensive, practical data concerning both networking libraries significantly hinders developers' ability to accurately assess the limitations of them, especially in regard to XR and networked simulated physics in general. Consequently, developers are hindered in their ability to make informed decisions regarding its structural integration into applications and the evaluation of corresponding bandwidth requirements.

The challenge is to conduct a theoretical analysis of the usability of both networking libraries for networked simulated physics in XR applications, specifically regarding physics-based interactions. This includes identifying their underlying concepts, comparing the differences between them, and highlighting their strengths and weaknesses. Based on these theoretical findings, potential solutions will be proposed. Subsequently, bandwidth and latency measurements of commonly used aspects within each library will be performed regarding such use case. A key challenge in this process is developing appropriate measurement methods, especially when working with HMDs. Furthermore, working with diverse datasets requires clear presentation and classification to accurately reveal relevant relationships.

## 1.7 Goal and Purpose

This bachelor thesis aims to evaluate the capabilities of Unity XR projects regarding networked physics. Specifically, it will assess the suitability of Unity's NGO and NCE networking libraries for implementing physics-based interactions. This evaluation will be conducted through an theoretical and technical analysis.

Furthermore, the study aims to explore how hardware capabilities affect bandwidth and latency, both of which are critical to the performance of networked XR

[8] https://unity.com/de/products/netcode (retrieved 31.08.2024, 22:25)
[9] https://docs-multiplayer.unity3d.com/tools/current/about/ (retrieved 31.08.2024, 22:42)

environments. It will also investigate the synchronization of real-time physics using NGO and NCE, identifying potential bottlenecks and unique challenges, and develop methodologies to measure key performance metrics.

Ultimately, while networked physical interactions in XR applications offer significant potential, they also present considerable challenges, particularly in relation to network latency and its impact on user experience. Through a careful examination of these issues, this research will provide valuable insights for industrial, economic, and academic contexts where immersive technologies are becoming increasingly important.

## 1.8 Structural Overview of Thesis

This bachelor thesis adheres to the standard structure of scientific work. Following the table of contents, the document includes sections for the list of figures, the list of tables and abbreviations, which are all placed between the table of contents and the introduction.

After the introduction, the thesis begins with a rather small review of related work, providing the necessary context and background for the research. This is followed by the theoretical background chapter, which covers the essential fundamentals needed to understand subsequent chapters. This chapter also includes a theoretical analysis of both networking libraries under consideration.

The methodology chapter then details the experimental setup, methods, and approaches used in the research. The results chapter follows, presenting and describing the key findings, with numerous references to the appendix for additional diagrams that, while not central to the discussion, support the understanding of the results.

A discussion section then reflects on the entire thesis, drawing conclusions and providing a comprehensive overview. The thesis is rounded off with a discussion of the limitations of the research, suggestions for improvements, and potential avenues for future work.

# Chapter 2

# Related Work

The research methodology for this study commenced with an extensive literature review utilizing platforms such as Semantic Scholar, Google Scholar, and IEEE Xplore. Various search queries were utilized, including "Networked Physics", "Networked Physics Simulations" and "Physics-based Interactions in XR" alongside tags like "XR", "Unity", "Physics" and "Interactions" However, relevant literature specifically addressing networking in XR was limited. Most available research focused on XR interactions, the associated challenges, and how to design effective user interactions, often emphasizing usability over networking concerns.

Previous studies on virtual and augmented reality largely concentrated on the potential applications of these technologies and the rapid advancements in this field. With the development of robust and precise systems, VR/AR technologies have been found suitable for professional and scientific use. Despite these advancements, significant challenges persist, particularly concerning high motion-to-photon latencies and the complexities inherent in networked virtual worlds, as highlighted by Anthes et al.. [2].

The advent of 5G technology, with its promise of higher bandwidth and lower latency, sparked renewed interest in networked XR applications. Consequently, numerous studies have focused on the implications of 5G and multi-user scenarios within XR applications. These works have addressed the requirements and potential issues of networking in XR environments, as well as evaluating the impact of 5G on extended reality, as noted by Ruan and Xie [17].

Orlosky et al. [14] discussed the limitations of 5G networks in delivering high-fidelity telepresence and collaborative virtual and augmented reality applications. Recognizing these challenges, researchers and engineers have been designing 5G networks to meet the future needs of XR technologies. They identified seven key challenges associated with networked virtual worlds: latency, bandwidth, Quality of Service (QoS), availability, colocation, social adaptation, security, and safety. They predict that "[...] we will likely see a migration to virtual workspaces".

Supporting this hypothesis, Westphal [5] analyzed the requirements for network architectures in AR/VR scenarios. He referred to the earlier work of Macedonia et al. [10], listing requirements such as bandwidth, delay, and computational needs. The impact of AR/VR on the networking infrastructure was considered "[...] to be too difficult to meet", leading to a reevaluation of network architectures.

Furthering this discussion, Ruan and Xie [17] aimed to present the challenges of integrating VR with networks. One of the biggest challenges they identified is the massive raw data rate requirement of 5 to 60 Gbps for an immersive online experience indistinguishable from real life. While 5G's peak data rate of 10 Gbps and service delay of 5 milliseconds have enabled practical large-scale VR deployment, other aspects like ultra-low latency, client/network access, system deployment, edge computing/caching, and end-to-end reliability remain critical. They identified three

major challenges concerning network optimization: the computing capacity of mobile HMDs, network communication efficiency, and network service latencies.

In addition to networking challenges, the significance of real-time physics in XR was explored, as it greatly enhances the realism and immersion of virtual worlds. Many XR interactions rely on spatial or physics-based interactions between users and the virtual environment. The previously discussed network latencies make synchronizing of such interactions like throwing and catching a ball challenging, requiring sophisticated approaches. Unity's inbuilt physics system plays a crucial role in XR application development, offering a production-ready solution. However, achieving the high-resolution, realistic simulations needed for sensitive human perception often exceeds the computational capabilities of mobile HMDs.

To address these computational limitations, the calculation of real-time physics can be offloaded to the cloud. This reduces the computational load on mobile devices but introduces potential network latencies, which can cause local delays in physics simulations. Kokiadis et al. [7] proposed a framework to optimize XR performance and enhance user experience by maintaining synchronization, accuracy, and real-time performance in distributed XR systems. They outlined four key requirements: low latency for real-time interactions, accuracy for high-fidelity simulations, consistency in synchronized physics simulations across devices, and effective edge-cloud collaboration to balance performance and resource usage.

An application example of these findings could be a virtual classroom for physics education, where students experience hands-on learning in an interactive environment. For instance, Krstić and Mekterović [8] used the Unity Physics Engine to simulate complex physics problems, demonstrating its potential to make physics education more engaging and relevant for non-physics students. Similarly, Sung et al. [19] developed a system that allows virtual, deformable objects to be projected into the real world and simulated in real time, aiming to make physics lessons more interactive, visual, and motivating for students.

While there is a substantial body of research on both physics simulation and networking, the literature specifically examining the challenges and opportunities of networked simulated physics is limited, especially for XR applications. A significant portion of the existing research on networked simulated physics in XR is application-oriented, highlighting their particular challenges and limitations encountered in specific use cases. This scarcity highlights significant knowledge gaps that warrant further investigation.

# Chapter 3

# Theoretical Background

The theoretical framework of this thesis delves into key concepts related to networking and physics simulations within the context of XR environments. Further, it provides explanations of hardware components to minimize low-level bottlenecks. The primary focus, however, lies in exploring and comparing the suitability of NGO and NCE for this task. Through a analysis of their strengths, weaknesses, and specific challenges in relation to physics-based interactions, we aim to determine which architecture is better suited for the development of high-performance and scalable XR applications. Particular attention is paid to the potential of NCE, a relatively new architecture that has not yet been extensively used for network-based physics simulations in XR. The Findings are based on the respective documentations of here NGO 1.11.0[10] and NCE 1.0.17[11].

## 3.1 Networking Fundamentals

### 3.1.1 Architectures

A network is structured based on a predefined architecture that governs the interaction between its services, devices, and clients to meet specific system requirements. This architecture defines the layout, protocols, and connectivity patterns, ensuring that devices communicate effectively. The network's non-functional properties - such as performance, scalability, maintainability, stability, extensibility, and security - are unique to each system. These properties, however, are often in competition with each other, requiring trade-offs, as not all can be maximized simultaneously.

#### Client-Server Model

One commonly used architecture is the client-server model, frequently seen in networked systems, including XR applications. In this model, the server holds authoritative control over the game state or system, while clients communicate with the server to request or send updates. The server handles state management and dictates what happens within the network. A challenge in this architecture is ownership control, where objects or interactions are managed by a particular client. When ownership needs to be transferred, such as when a different client wishes to interact with an object, a request must be sent to the current owner. This transfer process is mostly done by a RPC that introduces latency, as the action is delayed by the need to communicate back and forth between the server and clients.

[10] https://docs-multiplayer.unity3d.com/netcode/current/about/ (retrieved 06.09.2024, 13:46)

[11] https://docs.unity3d.com/Packages/com.unity.netcode@1.0/manual/index.html (retrieved 06.09.2024, 13:52)

#### Peer-to-Peer

The peer-to-peer model is another network structure, often used for systems that require direct communication between clients without the central authority of a server. In this setup, clients are both consumers and producers of information. While this approach can reduce the server load, it is prone to synchronization issues and introduces security risks, as individual clients may control vital aspects of the system without a centralized authority to validate updates.

#### Other Approaches

A hybrid approach combines both models, where a central server is responsible for critical state updates and validation, ensuring synchronization and security for important system changes. Meanwhile, less critical, high-frequency data, such as user movements, can be handled via peer-to-peer communication, reducing latency. This hybrid architecture improves performance by offloading tasks from the server to clients, but it introduces complexity in implementation. The networking code needs to handle both peer-to-peer communication and centralized control, ensuring proper synchronization while maintaining system security.

Other architectures, such as the lockstep model, exist but are less relevant in this context due to their inherent latency, which contradicts the real-time demands of XR applications. These architectures aim to ensure perfect synchronization but can cause delays that affect the user experience, particularly in immersive, physics-driven systems like those used in XR.

### 3.1.2 Network Topologies

Typical network topologies are structured either by hardware or logical abstraction, often forming tree-like configurations. A local area network (LAN) is a network confined to a limited spatial extent, typically within a single building or household, and consists of routers and switches connecting devices such as computers, smartphones, printers, or HMDs. The LAN connects to a metropolitan area network (MAN), which spans larger areas like a city, and eventually links to a wide area network (WAN). A global area network (GAN) integrates multiple WANs, creating a hierarchical network architecture, often visualized as a tree structure.

As the physical distance between connected devices increases, latency becomes a critical issue, as data transmission times grow longer. Additionally, connection reliability tends to degrade over long distances, as external influences like signal degradation, congestion, or interference become more likely.

Moreover, users typically have limited control over network infrastructure beyond their immediate local network. Consequently, perfect network conditions are seldom achieved, as the underlying connectivity is affected by factors outside the user's control.

### 3.1.3 Network Communication

Network communication between devices in a system typically involves the exchange of small information units known as network packets. These packets are transmitted over TCP/IP networks, allowing information to be transferred efficiently without sending large files in one go. By breaking data into smaller packets,

network bandwidth can be used more effectively, allowing multiple devices to communicate in parallel. These packets are structured with a well-defined length and format, enabling checks for completeness and correctness upon receipt.

A network packet is composed of three primary components: the header, the payload, and, in certain instances, a trailer. The header encapsulates metadata essential for packet routing and processing, including its origin, destination, and the nature of the transmitted data. The payload constitutes the actual data being transmitted. A trailer may be appended to the packet for supplementary purposes, such as providing error detection or correction mechanisms. The size of the header and the presence of a trailer are contingent upon the specific network protocol employed.

Beyond the foundational Internet Protocol (IP), which features a 20-byte header serving as a framework for other protocols[12], the Transmission Control Protocol (TCP) and the User Datagram Protocol (UDP) are among the most prevalent protocols in networking.

#### Transmission Control Protocol

TCP provides reliable, connection-oriented communication by managing the transmission of data streams and ensuring that data is delivered in the correct order. TCP is responsible for establishing and closing connections and is typically used in unicast communication. It is resource-intensive, necessitating the exchange of three packets to establish a socket connection prior to the transmission of any user data. Its header size starts at 20 bytes but can expand up to 60 bytes with additional options. TCP's robustness, including features such as flow control, retransmission of lost packets, and error correction, makes it suitable for applications like email and file transfer protocols (FTP). [13]

#### User Datagram Protocol

In contrast to TCP, UDP is a connectionless protocol with minimal overhead, designed for situations where low latency is more critical than guaranteed delivery. UDP supports unicast, multicast, and broadcast communication but lacks mechanisms for error detection or correction. Consequently, its header is significantly smaller, only 4 bytes, making it ideal for applications that need to transmit small amounts of data frequently, such as online gaming and live streaming. However, because it does not guarantee the delivery of packets, UDP may result in packet loss, especially in cases of network congestion or transmission errors.[14]

When a network packet fails to reach its destination, this is referred to as packet loss. Packet loss can occur for various reasons, including data transmission errors or network overload. In scenarios where reliable communication is essential, protocols like TCP can recover from these losses, while UDP-based applications must account for this possibility by either tolerating or mitigating packet loss through other means.

### 3.1.4 Remote Procedure Call

A Remote Procedure Call (RPC) allows a system to execute a function on another machine or in a different address space, giving the impression of a local function call. It is characterized by two main processes: the synchronous transfer of the control

[12] https://datatracker.ietf.org/doc/html/rfc792 (retrieved 05.09.2024, 13:46)
[13] https://www.ietf.org/rfc/rfc9293.html (retrieved 05.09.2024, 14:10)
[14] https://datatracker.ietf.org/doc/rfc768/ (retrieved 05.09.2024, 14:19)

thread and the exchange of data, such as parameters and results. RPCs work at the level of the programming language, abstracting the network interaction and making remote method invocation as straightforward as local function calls.

RPCs operate across two different address spaces, often with narrow communication channels. This means that the client and server are separated, and data must be serialized and transmitted over these channels, introducing overhead and potential latency.

The design of RPCs is based on a proxy model, where a pair of proxies (one on the client side and one on the server) facilitates communication. These proxies handle the binding process to connect the client and server, and they include error-handling mechanisms to ensure reliable execution.

RPCs are particularly well-suited for one-off actions or events, such as triggering specific processes or events across a network. While they allow for data exchange, RPCs are not optimal for high-frequency state synchronization. The overhead of synchronous calls and the need for constant data transfer make them inefficient for tasks like real-time object updates, which require more lightweight and frequent communication methods

### 3.1.5 Network Traffic

Network traffic refers to the flow of data across a network, including all the packets sent and received between devices. It represents the communication processes between computers, servers, and other networked devices and serves as the foundation for any form of digital interaction, such as file sharing, streaming, and gaming. The key objective of analyzing network traffic is to ensure efficient data transfer while minimizing delays, congestion, and packet loss.

A network packet includes essential information for routing and delivery, such as the Time-To-Live (TTL) that is a 8-bit field in the IP Header. The TTL specifies how many intermediary points, or "hops", a packet can pass through before reaching its destination. The hop count directly influences the Round-Trip Time (RTT), which represents the total elapsed time for a packet to traverse the network path, reach its destination, and receive a response. Generally, an increase in hops correlates with an increase in RTT.

The analysis of network traffic, often referred to as packet sniffing, involves monitoring and evaluating the flow of packets across a network. Packet sniffing tools, whether hardware or software-based, capture these data streams and allow for detailed traffic analysis. Such tools track packet flows between devices on the network and also examine the data exchanged between computers and the internet. Popular examples of packet sniffing tools include tcpdump[15] and Wireshark[16], which are widely used for monitoring, diagnosing, and analyzing network performance and security issues.

### 3.1.6 Bandwidth and Data Throughput

Bandwidth and data throughput are fundamental concepts in network performance. Bandwidth represents the theoretical maximum rate at which data can be transmitted over a network connection. It defines the upper limit of data transfer capacity. Data throughput, on the other hand, refers to the actual rate of data transmission at

[15] https://www.tcpdump.org/ (retrieved 01.09.2024, 9:44)
[16] https://www.wireshark.org/ (retrieved 01.09.2024, 9:46)

a given moment, influenced by factors such as network congestion, protocol overhead, and application limitations. It can be measured in bits per second (bit/s) or bytes per second (bps) In essence, throughput is the realized portion of the available bandwidth.

When bandwidth is constrained, the network may not be able to handle the required data load, leading to issues such as delayed updates or loss of synchronization. This can result in noticeable lag or inconsistency. Therefore, it is essential to assess bandwidth requirements beforehand to optimize implementations or upgrade hardware as needed to support the desired level of interaction.

Moreover, bandwidth availability can fluctuate based on the physical structure of the real-world environment. For instance, the movement of a user wearing a HMD can affect both the stability and bandwidth of the connection. Several factors contribute to this issue:

#### Signal Interference

As the user moves, their body, furniture, or other objects in the room can block the line of sight between the HMD and the router, causing interference. Signals of Wi-FI are particularly prone to attenuation when passing through physical objects, which can degrade the signal strength and reduce the effective bandwidth.

#### Multipath Interference

Wi-Fi signals can bounce off walls, floors, and other surfaces, causing multipath interference. When the user moves, the angles and paths of these reflections change, potentially leading to fluctuations in signal quality and network stability.

#### Distance Variability

Movement naturally alters the distance between the HMD and the router. As the distance increases, signal strength generally decreases, resulting in reduced bandwidth and higher latency. Wi-Fi performance is highly dependent on proximity to the router, with longer distances leading to lower data throughput.

#### Dynamic Network Adjustments

Modern Wi-Fi technologies include mechanisms like beamforming, which attempt to direct the signal toward the client device, such as an HMD. While these technologies improve performance, continuous user movement may challenge the router's ability to consistently adapt, causing potential instability in the connection.

Consequently, user movement in the room can result in fluctuating bandwidth and temporary drops in connection stability between the HMD and the router. [9] These fluctuations directly affect the performance of networked applications, especially those involving physics-based interactions in XR environments, where consistent data transmission is critical for maintaining immersion and interactivity. Proper network planning and optimization, such as using higher bandwidth technologies or minimizing interference, are essential to support the demands of networked physics in XR.

### 3.1.7 End-to-End Latency

End-to-end latency in network communications denotes the temporal interval between the initiation of a request by a source entity and the subsequent reception and processing of that request by the destination. Specifically, it encompasses the duration required for information to traverse from the source application layer to the destination application layer. In contrast, network latency solely measures the time taken for a data packet to travel from its origin to its destination.

A more encompassing metric, RTT, incorporates both the transmission time and processing time at the destination. This comprehensive measurement serves as an invaluable tool for diagnosing network anomalies, evaluating signal transmission efficiency, and ensuring reliable communication

Several factors[17] influence RTT and, consequently, latency:

1. **Distance**: The physical distance between the source and destination significantly affects signal travel time.
2. **Transmission Medium**: The type of medium (e.g., fiber optic, copper, wireless) impacts the speed of signal transmission.
3. **Network Hops**: Each server or router the packet passes through increases the processing time, thus raising the RTT.
4. **Traffic Levels**: High network traffic can slow transmission, increasing RTT and overall latency.
5. **Server Response** Time: The time a server takes to process and respond to a request varies based on its capacity and the complexity of the request.

In XR applications, latency is a critical factor that directly impacts user experience. High latency can disrupt the real-time feedback necessary for immersive, physics-based interactions. Consequently, local hosting approaches are often preferred over remote servers provided by third parties. By using a local server, developers can significantly reduce the physical distance between the host and client, thereby minimizing latency and improving responsiveness in networked XR environments.

### 3.1.8 Hardware

#### Ethernet Cables

Ethernet cables are essential for ensuring stable, high-performance network connections, especially in demanding applications like XR environments. These scenarios, which often involve high data throughput and low latency requirements, are sensitive to network disruptions caused by interference from external signals, electromagnetic noise, and fluctuations in wireless connectivity. Therefore, reliable Ethernet connections help mitigate these disturbances, ensuring smoother and more consistent performance for immersive technologies like VR and AR. The following information is based upon the "IEEE Standard for Ethernet" from 2018. [6]

In most private applications, Ethernet cables of Category 5 (Cat. 5) are commonly found. Cat. 5 cables support a bandwidth of 100 MHz, translating to a data rate of up to 1 Gbps (typically 1000BASE-T). However, due to the demands of modern network environments, Cat. 5 cables have largely been surpassed by Category 6 (Cat. 6) cables.

[17] https://www.geeksforgeeks.org/what-is-rttround-trip-time/ (retrieved 01.09.2024, 10:53)

Category 6 cables support bandwidths up to 250 MHz and data rates of 1 Gbps over distances of 100 meters (1000BASE-T). When used for 10GBASE-T, they can achieve 10 Gbps data rates but only over shorter distances of 55 meters. In contrast, Category 6a (Cat. 6a) cables are characterized by a bandwidth of 500 MHz and offer enhanced crosstalk protection, allowing them to support 10GBASE-T over the full 100-meter length. Typically, Cat. 6 cables are constructed as shielded (S-) or unshielded (U-) twisted pair (TP) cables cables. Fully shielded variants like the aforementioned S/FTP are also present, especially for Cat. 6a. For high-interference environments, such as those with significant electromagnetic interference (EMI), S/FTP cables are recommended, as they preserve the integrity of transmitted data by minimizing signal disruptions.[18]

Category 7 (Cat. 7) cables offer even greater bandwidth, supporting frequencies up to 1000 MHz, resulting in data rates of up to 10 Gbps when used for the 10GBASE-T standard. These cables are typically constructed as S/FTP cables. Cat. 7 cables also feature two connector types: GG45 and RJ45. The GG45 connector allows full utilization of the cable's 1000 MHz bandwidth, while the more widespread RJ45 connector, commonly used with Cat. 5 and Cat. 6 cables, limits the bandwidth to 500 MHz, reducing the cable's potential performance.

Although Cat. 7 cables surpass the capabilities of most consumer-grade routers and devices, they are recommended for high-performance environments. The intensive use of routers can introduce instability to Ethernet connections between the router and the server.

#### Routers

Routers function as essential intermediary nodes within network infrastructures, enabling communication among diverse network devices. As such, routers must possess sufficient signal strength, bandwidth, and functionality to ensure robust network connectivity. These attributes are particularly crucial in wireless networks where signal propagation and interference can significantly impact performance.

Routers play a critical role in establishing and maintaining network parameters. They are responsible for configuring IP addresses, subnet masks, default gateways, and other essential network settings. By acting as the gateway between different networks, routers enable seamless data transmission and routing. Moreover, routers often incorporate quality of service (QoS) mechanisms to prioritize specific types of traffic, ensuring that time-sensitive applications, such as video conferencing or online gaming, receive adequate bandwidth.

Although routers are instrumental in establishing and managing wireless networks, the actual bandwidth and latency experienced by individual devices can vary significantly due to differences in their individual capacities. For example, the Meta Quest 3, a state-of-the-art HMD, is compatible with Wi-Fi 6e (802.11ax). Wi-Fi 6e is an extension of the Wi-Fi 6 standard that adds the 6 GHz frequency band, which provides more spectrum, reduced interference, and greater capacity compared to its predecessors. According to the Wi-Fi Alliance, Wi-Fi 6e was officially introduced in early 2020, marking a significant advancement in wireless technology. [22]

Despite its growing adoption, many households and institutions still rely on older routers that support only Wi-Fi 5 (802.11ac), which creates a disparity in network performance across different environments. [21]

[18] https://www.elliottelectric.com/StaticPages/ElectricalReferences/DataComm/cat3-cat5e-cat6-cat7-cat8-ethernet-cable-guide.aspx (retrieved 20.09.2024, 17:36)

Wi-Fi 6e brings several improvements over Wi-Fi 5, including increased bandwidth, reduced latency, and enhanced throughput due to its additional spectrum and higher efficiency protocols. Assuming perfect conditions, Wi-Fi 6e supports theoretical speeds up to 9.60 Gbps, compared to Wi-Fi 5's maximum of 6.93 Gbps[19], along with enhanced multi-user performance due to the implementation of Orthogonal Frequency-Division Multiple Access (OFDMA). These advancements make Wi-Fi 6e particularly well-suited for high-bandwidth applications such as networked XR environments. [23]

Theoretical bandwidths are often not achievable in practice due to suboptimal configurations and environmental factors. These include the receiver being located several meters away, or objects and people obstructing the signal, which is particularly challenging for HMDs. For instance, reaching the 6.9 Gbps maximum of Wi-Fi 5 requires the use of 160 MHz channels, 8 spatial streams, and 256 QAM modulation. However, most consumer devices do not fully support these parameters and typically operate with narrower channels and fewer spatial streams. According to the MCS Index, such configurations yield theoretical bandwidths in the range of 0.86 Gbps to 1.73 Gbps (2 to 4 spatial streams at 80 MHz), well below Wi-Fi 5's advertised maximum.

## 3.2 Hardware Constraints of Mobile HMDs

Maintaining an appropriate framerate (FPS) is crucial to align with the server's network tickrate. Framerate, which refers to the number of frames or images rendered per second, dictates how frequently visual stimuli can be presented to the user. In contrast, the network tickrate defines the frequency with which the server sends updates or snapshots about the shared game world to clients. These updates include new stimuli from the physics simulation or other clients. There are three primary outcomes based on the relationship between the client's update rate and the network tickrate.

1. **Client Update Rate Higher than Network Tickrate**: In this scenario, the client can process the latest server snapshot in a timely manner, ensuring a seamless integration of data.
2. **Client Update Rate Lower than Network Tickrate**: Here, multiple snapshots may be received within a single client update. However, clients typically process only the most recent snapshot. Discarding previously received snapshots can lead to data inconsistencies, resulting in visual artifacts such as abrupt jumps in moving objects.
3. **Network Client Update Rate Equal to Network Tickrate**: Although the rates are ideally matched, slight variations can still cause occasional jumps, though these are less frequent.

Additionally, varying network latency can exacerbate these issues by causing fluctuations in snapshot delivery times, leading to effects similar to those described in the second scenario. To mitigate these challenges, the client's update rate should be set higher than the network tickrate to accommodate latency variations.

Framerate heavily depends on the system's computational capabilities. Since HMDs are designed to be lightweight and unobtrusive, their computational power

[19] https://mcsindex.net/ (retrieved 19.08.2024, 13:54)

is limited. Therefore, the network tickrate should be tailored to each device and scenario.

## 3.3 Simulated Physics

Physics simulations in digital environments allow virtual objects to behave according to the laws of physics, such as Newtonian mechanics. These simulations aim to replicate real-world physical interactions, such as gravity, collision, friction, and material properties, providing a foundation for creating realistic virtual worlds. Physics simulations are essential for a variety of applications, from video games to scientific visualization, but their complexity grows significantly in the context of networked applications like XR.

At a technical level, physics engines manage objects through rigid body dynamics, collision detection, and constraint systems. In most cases, the simulation operates within a discrete time-step model, where the state of objects (position, velocity, acceleration) is updated in small time intervals (ticks).

### 3.3.1 Key Components

#### Rigid Body Dynamics

Rigid body dynamics represent objects as solid entities with constant shape and size that respond to forces and torques. Each rigid body in a physics simulation has properties such as mass, inertia, and center of mass. These properties influence how an object reacts to forces like gravity or collisions. Typically a physics engine simulates rigid body dynamics by solving equations of motion for each object, taking into account external forces and constraints.

For example, in XR, if a user picks up a virtual object, the physics engine ensures the object moves according to the user's input, factoring in gravity and collisions with other objects. A key challenge is the real-time responsiveness required in XR environments, where any delay or inaccuracy can lead to a noticeable break in immersion.

#### Collision Detection and Resolution

Collision detection is the process of determining when two or more objects come into contact within the virtual space. A physics engine uses bounding volumes (e.g., AABBs or spheres) and swept volumes to detect collisions efficiently. Once detected, a collision response algorithm is triggered, which calculates the forces required to resolve the collision and ensures objects don't overlap. This process typically involves calculating impulses based on the objects' masses and velocities to simulate realistic bounces or stops.

For networked simulations, maintaining collision consistency across multiple devices is crucial. Each client must resolve collisions similarly, meaning physics data must be transmitted or synchronized across the network in real time.

#### Constraints and Joints

In many simulations, objects need to interact with one another in a restricted way, such as a door hinge or a pendulum. Constraints or joints are used to limit the degrees of freedom between objects. Often a variety of joint types are provided

by a physics engine, including fixed joints, hinge joints, and spring joints. These constraints are calculated by solving a system of linear equations, which restrict how objects can move relative to each other, thus preserving the realism of the simulation.

For example, in an XR environment where two users are manipulating a connected system (e.g., a pulley), the joint constraints must be enforced across both users' devices. Networked physics simulations need to ensure that the interactions are synchronized, as any discrepancy would disrupt the realism and immersion.

#### Soft Body Physics and Cloth Simulation

While rigid bodies are static in shape, soft bodies deform under forces. Simulating soft-body dynamics requires more computational power, as it involves solving additional constraints for deformation, spring forces, and collisions with other objects. Physics engines often offer cloth type components, which use a simplified version of soft-body dynamics to simulate the movement of fabrics and flexible materials.

In XR, soft body simulations are essential for tasks like simulating human avatars, where clothes, hair, and skin must move and deform naturally to avoid breaking immersion. However, simulating these interactions in real time, especially in a networked environment, presents a significant challenge in terms of bandwidth and computational load.

### 3.3.2 Physics Capabilities of Unity

Unity offers various physics engine implementations tailored to different project needs, including 3D, 2D, object-oriented, and data-oriented, component-based approaches. For object-oriented projects, Unity provides built-in physics engines, such as the 3D physics engine integrated with **Nvidia's PhysX** and the 2D physics engine integrated with **Box2D**. These built-in engines are enabled by default when starting a new project.

When transitioning from MonoBehaviours to Entities, the ECS package must be installed. ECS is part of Unity's Data-Oriented Technology Stack (DOTS)[20] and relies on its other components. DOTS enables high-performance C# code through the Burst Compiler and C# Job System. Notably, switching to ECS does not automatically include a physics system. Unity provides various options for integrating physics with ECS.

- **Unity Physics Package**: This is the default DOTS physics engine that must be installed to simulate physics in data-oriented projects.
- **Havok Physics for Unity Package**: This package provides an implementation of the Havok physics engine as an extension of the Unity Physics package.

The Havok Physics package is subject to a specific licensing scheme and is only available with Unity Pro or Enterprise subscription models.

When comparing how many objects can be handled at 30 FPS using MonoBehaviours versus various combinations of DOTS, a clear difference emerges, as seen in fig. 3.1[21]. The use of DOTS with the Burst Compiler can already handle twice as many objects. Furthermore, incorporating the C# Job System, which enables multithreading, allows for a tenfold increase in the number of objects processed. This

[20] https://unity.com/dots (retrieved 08.09.2024, 19:29)
[21] https://github.com/UnityGameAcademy/DOTSCompare (retrieved 08.09.2024, 19:34)

performance gain is consistent with the findings of Versteeg's analytical comparison of DOTS and MonoBehaviours [3].

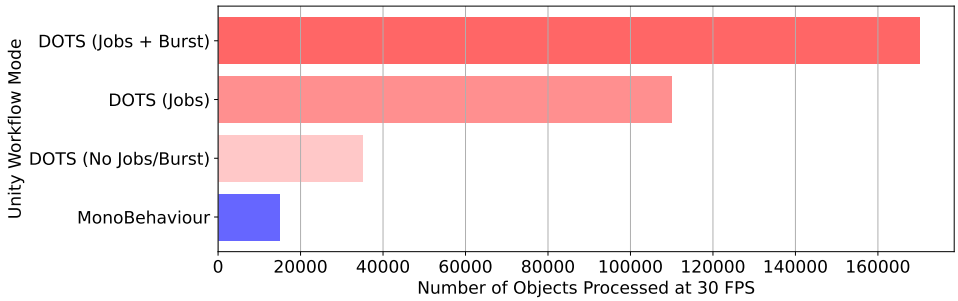



FIGURE 3.1: Performance Comparison in Object Count at 30 FPS Between MonoBehaviour and DOTS

## 3.4 Requirements of Networked XR Applications

Networked XR environments introduce a unique set of challenges that are crucial to address for maintaining real-time interaction and immersion. The demands placed on such systems are further highlighted in the 5G White Paper V1, published in 2015 [1]. This paper outlines specific requirements for general networked XR applications, including:

1. **Low Latency**: Minimal delay between user actions and visual feedback is crucial for a realistic experience.
2. **Synchronization**: All users must share a common perception of the virtual world for seamless interactions.
3. **Scalability**: The application must support a large number of concurrent users.
4. **Interactivity**: Users should be able to interact with objects in the virtual environment.
5. **Network Quality**: A stable and high-speed internet connection is essential for a smooth experience.
6. **Hardware Capabilities** The performance of devices (computers, VR headsets) significantly impacts the quality of the experience.
7. **Data Privacy** Protecting user data is a critical aspect of multi-user XR applications.the request.

## 3.5 Networked Simulated Physics

When physics simulations are distributed across a network, each client device must maintain a consistent state of the physics simulation while communicating with other devices. Given the inherent variability of latencies and bandwidths in real-world scenarios, as well as the potential for packet loss or other forms of interference, it is necessary to employ a range of techniques to meet the requirements for networked simulation physics.

#### State Synchronization

In a networked physics simulation, each client must receive updates on the state of objects in the environment. This typically involves synchronizing object positions, velocities, and other physical properties across devices. In Unity, this is often handled through snapshot interpolation, where periodic snapshots of an object's state are sent across the network. Each client interpolates between these snapshots to estimate the current state of the object.

For XR applications, precise synchronization is essential for physics-based interactions. For example, if one user throws a virtual ball, the other users must see the ball's trajectory in real time, without noticeable latency or lag.

#### Predictive Algorithms and Client-Side Prediction

In cases where high latency or network jitter is present, predictive algorithms are used to reduce the perceptual impact of network delays. Client-side prediction involves using the local physics engine to predict the state of objects based on known inputs (e.g., user actions) and the last known object state. When new state updates are received from the server, the local predicted state is corrected, minimizing noticeable desynchronization.

However, the use of predictive algorithms can sometimes result in corrective jumps, where an object suddenly shifts position if the local prediction differs significantly from the server's authoritative state. In XR environments, these sudden shifts can be jarring for users, breaking immersion.

#### Rollback Mechanisms

Another approach to handling networked physics is through rollback mechanisms, where the state of the simulation is periodically recalculated based on the most recent, authoritative data from the server. This technique is common in competitive multiplayer games but is challenging in XR due to the real-time constraints on interaction. Rollback can introduce noticeable pauses or delays, disrupting the fluidity required for physics-based XR interactions.

## 3.6 NGO and NCE for Networked Simulated Physics XR

There are two approaches to network architectures in Unity that are relevant to this study: object-oriented architectures and component-based architectures.

**Netcode for GameObjects** is Unity's primary networking solution for traditional game development, following an object-oriented approach. It integrates closely with Unity's game objects, providing a straightforward method for developers to synchronize objects and their properties across clients. However, its object-oriented nature can result in higher network overhead, particularly in complex simulations where multiple objects interact in a physics-based environment. For XR applications, NGO may be more suitable for smaller-scale interactions or where ease of development is prioritized over scalability.

**Netcode for Entities**, on the other hand, uses Unity's ECS and is designed for high-performance applications. NCE excels in scenarios where many entities need to be synchronized across a network, making it more suitable for large-scale XR environments. Its data-oriented design minimizes the amount of data transmitted

over the network, which is critical for maintaining high performance in resource-constrained XR devices. However, NCE's complexity can present a steeper learning curve for developers compared to NGO.

### 3.6.1 Network Communication

Both NGO and NCE support RPCs, but the underlying architecture impacts how they are implemented and their efficiency. In NGO, RPCs are tightly integrated with game objects, making them intuitive for developers familiar with Unity's traditional object-oriented design. In contrast, NCE treats RPCs more abstractly, operating at the system level, which can offer more flexibility and efficiency for large-scale simulations.

Other methods include state synchronization and snapshot replication, where the entire state of an object (e.g., its position, rotation, velocity) is periodically sent to ensure consistency across all clients. For physics-based interactions, these methods are particularly important, as they ensure that physical states are kept consistent in real-time.

### 3.6.2 Bandwidth and Latency Considerations

Bandwidth and latency are critical factors for real-time networked physics in XR, where even minor delays in object interaction can disrupt user immersion. NGO is more bandwidth-intensive due to its reliance on RPCs, Network Variables and other Network components to synchronize physics states across the network. In situations where multiple users interact with objects simultaneously, this can lead to significant bandwidth consumption and noticeable latency, especially in XR environments where immediate feedback is critical.

In contrast, NCE can provide better network efficiency due to its deterministic and chunk-based processing model, which reduces the need for frequent updates. By leveraging deterministic physics, NCE ensures that all clients in the network can simulate the same physics interactions without requiring constant synchronization, which significantly reduces bandwidth usage.

Latency is also a key factor, as delays in network communication can cause desynchronization of physical states. For instance, if two users attempt to manipulate the same object simultaneously, network latency can result in conflicting actions, leading to an inconsistent experience. Both architectures need to minimize latency through efficient communication protocols and synchronization techniques.

### 3.6.3 Physics and Networking

When comparing NGO and NCE for networked physics simulations in XR applications, notable differences arise. NGO provides a simpler, more intuitive workflow but is less efficient in terms of bandwidth and scalability. It struggles with larger simulations involving numerous users and objects, especially when real-time physics synchronization is crucial. NCE, while more complex, delivers superior performance for large-scale simulations, making it more suitable for applications where physics-based interactions are central to the user experience.

NGO does not support predictive algorithms, client-side prediction, or rollback mechanisms. The client can visually anticipate values, but this anticipation is purely local. With re-anticipation, these local values are reset to match the values contained in the server snapshot at each tick. Anticipation here refers to the client's ability

to guess upcoming states based on the current context, but these guesses are not validated or synchronized with the server.

In contrast, NCE incorporates predictive physics, specifically forward prediction, and lag compensation. Predictive physics involves forecasting the future state of objects based on current data and velocity, which helps mitigate the impact of network latency. Lag compensation adjusts for delays in network communication, improving the accuracy of interactions.

Both systems require custom implementation of client-side prediction (CSP). CSP involves the client predicting its own future state based on local data and user inputs, then reconciling these predictions with the server's authoritative state to correct discrepancies.

### 3.6.4 Practical Usability with Existing XR SDKs

When evaluating the practical usability of NGO and NCE for networked simulated physics in XR applications, the fundamental challenge revolves around compatibility with existing XR SDKs.

The key issue lies in the fact that current XR SDKs, such as those provided by Oculus, HTC Vive, and Microsoft HoloLens, are tightly coupled with `MonoBehaviours`, meaning they depend on Unity's `GameObject` system. Since NCE operates in a completely separate paradigm, integrating existing XR features like input systems, tracking, and headset-rendering pipelines into an NCE-based project would require significant restructuring. In practice, if developers wish to fully utilize NCE's advantages, such as scalable and high-performance physics, they would need to port their entire XR systems into ECS. This porting would involve rewriting the entire system in a data-oriented way, which is a time-consuming and complex process.

Given the importance of networked simulated physics in XR for immersive interactions, the complexity of transitioning to NCE is a crucial factor in its practical usability. While NCE offers better scalability and deterministic simulations, it also introduces significant hurdles for projects already reliant on `MonoBehaviours`.

### 3.6.5 The Hybrid Approach: Leveraging NGO with ECS

The core idea behind this hybrid approach is that NGO continues to manage the core XR interactions, such as camera rendering, input management, and user interfaces, while the ECS system is responsible for the more performance-intensive tasks like physics calculations. Since `Entities` and `MonoBehaviours` cannot directly communicate due to their architectural differences, developers would need to use bridging mechanisms. This involves setting up reference points between the two systems where NGO components can pass data to ECS-based systems or vice versa. For instance, a `MonoBehaviour` object (such as a player's hand controller in an XR setting) could pass its transform data to an entity for physics calculations, allowing ECS to handle real-time physics interactions while the rest of the XR system remains in NGO.[22]

This hybrid model allows developers to take advantage of ECS's performance benefits without the need to completely overhaul existing XR systems. It also mitigates some of the compatibility issues between the two frameworks by isolating physics calculations to ECS while maintaining user-facing features in NGO.

[22] https://wirewhiz.com/setting-up-unity-xr-for-ecs/ (retrieved 07.09.2024, 11:32)

### Bandwidth Considerations

The hybrid approach addresses this challenge by integrating the potential bandwidth efficiency of ECS for handling physics interactions with the simpler NGO framework for other XR functionalities. By confining physics calculations to the ECS, the system can reduce network traffic through the use of a customized state synchronization[23] with snapshot compression[24], which synchronizes the state of the ECS world. This allows for physics-based interactions while preserving low-latency communication for user inputs and interactions managed by NGO. Consequently, this approach strikes a balance between minimizing bandwidth consumption and ensuring the responsiveness and real-time feedback essential for XR applications.

### Practical Considerations for Implementation

When implementing such approach, developers must carefully manage the interaction between ECS-based physics and `MonoBehaviour`-based XR systems. One of the key challenges is ensuring that data is effectively transferred between the two systems without introducing significant overhead or latency. In practice, this could involve setting up reference bridges between `MonoBehaviours` and ECS components, allowing for transform data to be passed to ECS systems for physics simulations.

Another consideration is the synchronization of physics states. While ECS's deterministic nature helps reduce the number of physics updates required, developers must still ensure that network events (such as object interactions or collisions) are accurately communicated between clients. By isolating physics simulations within ECS, developers can take advantage of ECS's parallel processing and deterministic simulation to maintain smooth interactions, even in environments with a lot of concurrent users (CCU).

### Conclusion

While ECS offers significant advantages in terms of performance, scalability, and network efficiency for networked simulated physics, its practical usability in XR applications is limited by the incompatibility with existing `MonoBehaviour`-based XR SDKs. Fully transitioning to ECS would require porting entire systems to the ECS architecture, which is often not practical for existing XR projects or XR prototypes.

However, by adopting a hybrid approach, developers can leverage the performance benefits of ECS for physics interactions while continuing to use NGO for other XR functionalities. This approach provides a balance between reducing bandwidth consumption and latency for physics interactions, while maintaining compatibility with existing XR systems. The hybrid model also allows for more flexibility in project design, enabling developers to use the most appropriate tools for each part of the system without needing to completely restructure their project architecture. For XR applications that rely heavily on networked simulated physics, this hybrid approach presents a practical and efficient solution.

[23] https://gafferongames.com/post/state_synchronization/ (retrieved 08.09.2024, 14:12)
[24] https://gafferongames.com/post/snapshot_compression/ (retrieved 08.09.2024, 14:14)

# Chapter 4

# Methodology of Data Collection and Comparison

This section outlines the methodology used for data acquisition and analysis in the thesis. It explains the reasoning behind key decisions, such as the selection of hardware components, and introduces the tools and materials utilized, along with the custom implementations developed for this research. Brief descriptions of these tools and their application to the study can be found in the appendix.

The research is based on a server-client architecture, which provides greater control, monitoring capabilities, and the potential for direct intervention via the Unity Editor when a personal computer is employed as the dedicated server. This setup is particularly valuable for research and reflects a practical approach. Consequently, a client-host model is not employed due to the limited computational power of mobile HMDs.

Performance assessments are made through the analysis of selected metrics and corresponding measurement data, which are discussed throughout the chapter. The research primarily employs a quantitative methodology under controlled conditions to minimize external influences on the measurements. The specifics of data collection, such as the number of data points and the processing methods, are outlined in the relevant sections, as these details vary depending on the measurement scenario.

To ensure consistency across all measurements, the hardware selection, including Ethernet cables and routers, is discussed first. This ensures that the appropriate equipment is chosen, with particular attention to potential bottlenecks. The choice of a HMD is also examined in detail. Additionally, the setup and processing of video data for motion-to-photon latency measurements, which are critical for two experimental scenarios, are explained. The hardware setup is again listed and organized to provide a comprehensive overview of the system before transitioning to the actual experiment scenarios.

The section covering the experimental scenarios provides an overview of the various measurement setups, each grounded in the theoretical findings. It includes detailed explanations of their specific characteristics and measurement methodologies. Each scenario was implemented as a Unity scene, with the workflow involving the deployment of a server and a client application. The server initiated the measurement process by signaling the client, which then collected the relevant data. This data was subsequently visualized. All external software used is thoroughly described, and custom implementations are explained in detail. Pseudocode is provided to enhance understanding of the measurement processes and procedures.

The processing and visualization of all measurement data will be carried out using PYTHON version 3.12.2, unless otherwise specified. Data visualizations will be generated using the MATPLOTLIB package version 3.8.3, while NUMPY version 1.36.4 will be employed for data processing. Additionally, the pre-installed OS package will

be utilized for file handling and data management tasks. The individual plotting implementation will be hosted online in a GitLab repository.[25]

This chapter concludes with a tabular summary of all measurement scenarios and their respective objectives, providing a comprehensive overview of the research process.

## 4.1 Hardware Setup

All measurements will be conducted in the Immersive Experience Lab (IXLab) of the TU Dresden located in the APB building on the campus.[26] This room serves as an ideal testing environment due to its thick, insulating concrete walls that minimize external electromagnetic interference, ensuring precise control over the test conditions. Before conducting the experiments, the actual levels of external high-frequency electromagnetic radiation, including those emitted by other routers, were assessed to ensure accurate measurements.

As the network architecture employs a client-server model and heavy calculations like the physics simulations are going to be handled by that server, a dedicated server machine was used. For mobility purposes an upgraded version of the ROG Strix G17 G713RW[27] was chosen. Key features of that laptop include an ASUSTeK G713RW mainboard, that supports Ethernet speeds up to 2.5 Gbps by a RJ45 connector, 2x 16 GB DDR5-4800 MHz dual channeled RAM, and a Ryzen 9 6900HX CPU. These components are essential for ensuring fast computations times, robust communication and processing capabilities within the network. Additionally, due to the better Ethernet connectivity, this laptop was preferred to the stand-alone computers available.

In the actual setup, tables with a height of 75 cm were utilized, comprising a metal frame and a wooden top. The configuration is centered around the middle table, with a NETGEAR Nighthawk (RS700S)[28] router positioned on it. This Wi-Fi 7 router is capable of supporting the full range of Wi-Fi 6e specifications, as it is designed to handle up to 12 Wi-Fi streams, employs 4096-QAM modulation, and utilizes up to 320 MHz channels. To further minimize potential interference, no objects other than the server were placed within a 200 cm radius of the router. A Meta Quest 3 headset, chosen for its advanced features and compatibility with Wi-Fi 6e, was situated 400 cm from the router at a height of 1.78 meters above the floor. The headset maintained a clear line-of-sight to the router, ensuring an optimal wireless connection. In order to mitigate potential signal interference and ensure the highest possible stability, a Cat. 7 cable has been selected over a Cat. 6 cable for Ethernet connectivity. The additional shielding provided by Cat. 7 helps minimize error sources and ensures consistent network performance. Therefore a 2 meter-long Cat 7 cable was used to connect the server machine to the router. The cable is laid without any sharp bends or coils to prevent signal degradation. As the server's Ethernet capabilities are exceeding the 1 Gbit Lan Ports the 10Gbps Ethernet LAN RJ-45 port of the router was used.

The above described setup will be consistently applied across all subsequent measurement scenarios, unless otherwise specified. A scheme of it is demonstrated in section 4.1 and the real setup in the lab is shown in section 4.1 below.

[25] https://dev.ixlab.inf.tu-dresden.de/student-projects/thesis/networking-florian-demmler/data-plotter (retrieved 08.09.2024, 15:24)

[26] https://navigator.tu-dresden.de/raum/542100.2590 (retrieved 29.08.2024, 20:36)

[27] https://rog.asus.com/de/laptops/rog-strix/rog-strix-g17-2022-series/ (retrieved 29.07.2024, 21:09)

[28] https://www.downloads.netgear.com/files/GDC/RS700S/RS700S_TS.pdf (retrieved 29.08.2024, 21:14)

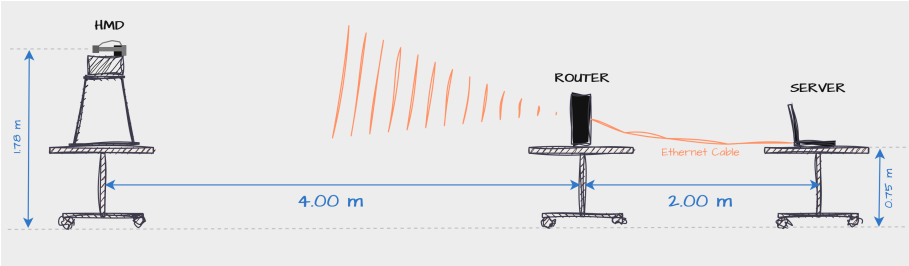



FIGURE 4.1: Scheme of the Measurement Setup
A schematic demonstration from the side of the planned in-situ experiment setup.

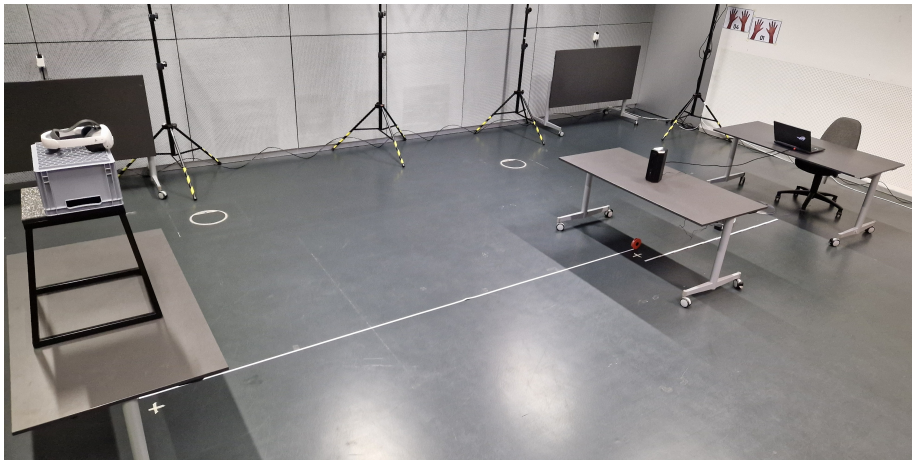


FIGURE 4.2: In-Situ Measurement Setup at the IXLab

## 4.2 Measurement Approaches and Methods

### 4.2.1 Comparison of 5 GHz and 6 GHz Wi-Fi Frequency

To assess the possible performance differences between Wi-Fi 6 and Wi-Fi 6e routers when used with the Meta Quest 3, a comparative analysis focusing on bandwidth and latency was conducted. Using the same measurement setup as described previously, the router with each of the two models under consideration was replaced. Prior to testing, the capabilities of both routers were examined to ensure a correct comparison if possible. To minimize interference, all surrounding devices were set to airplane mode, with Wi-Fi and Bluetooth disabled. Additionally, surrounding networks, signal strengths, and bands were recorded before the measurements to exclude potential interferences. For this the Wi-Fi Analyzer[29] application was used.

The first router evaluated was the previously described NETGEAR Nighthawk. Subsequently, the TP-Link Archer AX50, a Wi-Fi 6 router with only 1 Gbps LAN ports and support for the 5 GHz band was tested.

Latency measurements were conducted using HRPING[30] and the PING TOOLS app, measuring respectively the latency from the server to the router and from the HMD to the server.

For bandwidth measurements, IPERF3[31] was utilized. IPERF3 measures bandwidth using a client-server model, which required running this software on both devices. To use IPERF3 on the Android-based Meta Quest 3, the PING TOOLS[32] app,

[29] https://apps.microsoft.com/detail/9nblggh33n0n (retrieved 22.08.2024, 14:51)
[30] https://www.cfos.de/en/ping/ping.htm (retrieved 19.08.2024, 17:36)
[31] https://iperf.fr/ (retrieved 19.08.2024, 17:28)
[32] https://play.google.com/store/apps/details?id=ua.com.streamsoft.pingtools (retrieved 25.08.2024, 17:14)

that offers various tools for examining network metrics, was employed. Initially, JPERF, a Java-based graphical interface for IPERF3, was used for testing on the server, but for subsequent tests, IPERF3 was run in the console. Given that the network traffic for NGO primarily relies on UDP packets, UDP bandwidth measurements were prioritized. The bandwidth results were then compared to the maximum downlink and uplink speeds displayed in the Wi-Fi settings of the Meta Quest 3.

The outcomes of these network performance tests will elucidate the hardware limitations and potential bottlenecks or issues in each scenario. The collected data will also be cross-referenced with the observed network packet sizes in NGO enabling a detailed analysis of bandwidth efficiency and identification of possible limitations. Ultimately, this investigation aims to highlight the bandwidth differences between Wi-Fi 6e and Wi-Fi 6 and identify any potential latency issues.

### 4.2.2 Motion-to-Photon Latency Measurements

Given the challenges of accurately measuring one-way latencies due to discrepancies in application and system times, and the difficulty in achieving precise synchronization, a measurement technique similar to that used for MTP latencies has been employed in this study. This approach is required in two measurement scenarios. Consequently, throughout this document, the term "MTP latency" will be used, even though in a virtual setup this is meant in a more figurative sense.

As all these MTP latency measurements are conducted using a specific setup, this section details the setup, including the data processing involved. A dedicated setup is particularly necessary because MTP latency measurement involves comparing pixel regions across two displays. The critical aspect here is that a video provides the temporal basis for determining latencies. Therefore, at least the relevant areas of the displays on both the server and the client must be visible in the same video.

In this setup, the HMD must be positioned close to the server's screen. A stable video is crucial for analysis as it facilitates the processing of the pixel regions being compared. To achieve this, a camera tripod was used to hold a Samsung Galaxy S22, which captured both displays within the frame. This setup is demonstrated below in fig. 4.3.

The Samsung Galaxy S22 was selected for video recording because none of the other available devices were capable of capturing footage at a frame rate exceeding 240 frames per second (fps). As a result, the temporal resolution of the measurement is approximately 4.17 ms. This temporal resolution is deemed sufficient given that the HMD has a maximum refresh rate of 120 Hz and the server's display refreshes at 240 Hz. Since the display refresh cycles and the camera capture are not synchronized, recording may introduce a temporal uncertainty of up to one frame on the server side. However, this uncertainty is considered negligible for the purposes of this measurement. With more samples, the measurement accuracy can be further improved through averaging.

Due to the repositioning of the HMD, it is now located 2 m from the router instead of 4 m. Because the HMD is placed in front of the screen, the line of sight to the router is partially obstructed. However, this is not expected to cause a significant increase in latency or interference.

Since the measurements rely on color differences, it is important to maximize the difference in the Euclidean distance of a pixel's color vector in RGB format. As the video data processing is conducted using 8-bit RGB color (ranging from 0 to 255),


FIGURE 4.3: In-Situ MTP Measurement Setup at the IXLab
A laptop is positioned on a desk, with a Meta Quest 3 placed on top of it at screen height. A Samsung Galaxy S22 is mounted on a tripod to capture video of both the laptop's display and the HMD.

the minimum measurable vector magnitude is $1/\sqrt{3} = 0.577$. To maximize the measurable distance, both displays are set to either black (0, 0, 0) or white (255, 255, 255). This maximized color difference ensures high accuracy in video data processing.

The processing of the recorded video is carried out using Python 3.12, primarily utilizing the wrapper package[33] version 4.9.0.80 for the `OpenCV`[34] library. The video is first checked and converted to MP4 format if necessary. Then, the two pixel regions, referred to as regions of interest (ROIs), are selected from the first frame of the video. To enhance measurement precision, only the center pixel of each ROI is used. This is demonstrated in chapter B.

Further processing is conducted separately for each pixel. Frame by frame, the color of the pixel is determined, and the magnitude of the RGB vector is calculated. (fig. B.2 and fig. B.3) This value is then added to a list. The data in this list is subsequently differentiated to identify frames that show significant changes in the Euclidean distance according to the set thresholds. Positive and negative pre-configured thresholds are used to filter these changes, which are referred to as "spikes". This is visualized in fig. B.5 and fig. B.6. These spikes are then sorted into pairs using a custom clustering algorithm with a pre-configured window size. The frame intervals are calculated, and these intervals are converted into actual elapsed time using the inverse of the capture frequency. The resulting time represents the latency between a color change in the first ROI and the corresponding change in the second ROI within a specified maximum frame distance. This value represents the MTP latency.

These latencies are averaged, and a moving average is calculated. A trendline is then determined using linear regression. Figure B.7 illustrates a sample visualization based on the data utilized in the aforementioned figures. The library `matplotlib` is

[33] https://pypi.org/project/opencv-python/ (retrieved 01.09.2024, 8:25)
[34] https://opencv.org/ (retrieved 01.09.2024, 8:29)

used to visualize all these computed data. The implementation will be made available in a GitLab repository.[35]

To verify the validity of this method, a validator was developed using the same Python version. This validator generates two windows with adjustable latency, which were then recorded and processed as described above. By comparing the latencies obtained from this setup, the method was validated as accurate.

This measurement method is affected by a general discretization issue caused by the sampling frequency of 240 Hz. The sampling interval of 4.17 ms implies that if a color change occurs on the screen immediately after a frame is captured, introducing a temporal uncertainty of at most one sampling interval (4.17 ms). This effectively reduces the resulting MTP latency, as the color change is only registered in the subsequent frame. This issue is demonstrated for "ROI 1" in fig. 4.4. The orange dotted line marks the frame where a color change is detected, while the arrow of the same color indicates the actual MTP change difference. From that visualization, it is also clear that, in theory, the difference between the frames in a frame pair could theoretically be zero. However, this is not the case here, as the MTP latency is expected to exceed 4.17 ms by a few times. It is also worth noting that if a color change were to occur twice in quick succession, for instance, from black to white and white to black, no latency would be recorded. However, such a scenario is unlikely, as the measurement setup was designed with this consideration in mind.

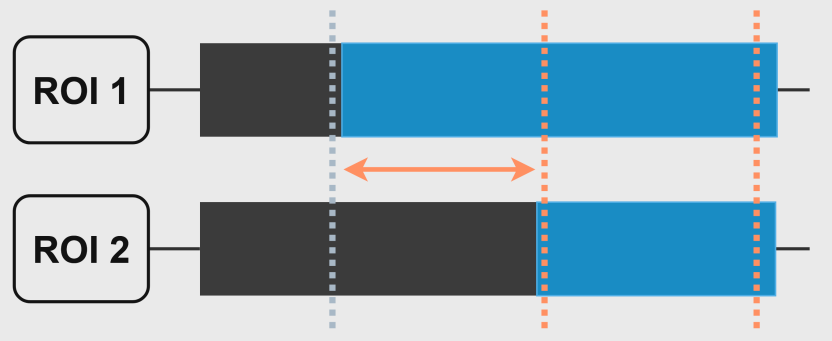



FIGURE 4.4: Demonstration of a Discretization Issue

### 4.2.3 Netcode for GameObjects

All scenarios in this study were implemented using Unity 2022.3.43f1. The default settings of a newly created 3D project utilizing the inbuilt render pipeline were retained, with adjustments only made in the specific implementations of the measurement scenarios. OpenXR 1.12.0 and the Meta Quest All-in-one SDK version 68.0.0 were subsequently imported. Additionally, NGO version 1.10.0, which relies on Unity Transport 1.4.1, was integrated. Requisite dependencies were automatically installed during the package setup process.

Building upon NGO, a utility tool was developed in the form of a 3D World UI. This tool, created as a prefab, could be employed in each measurement scenario for establishing connections between client and server, as well as for debugging purposes. The tool was particularly useful for displaying frame rates and Debug.Log messages, as illustrated in fig. 4.5

[35] https://dev.ixlab.inf.tu-dresden.de/student-projects/thesis/networking-florian-demmler/mtp-latency (retrieved 08.09.2024, 15:28)

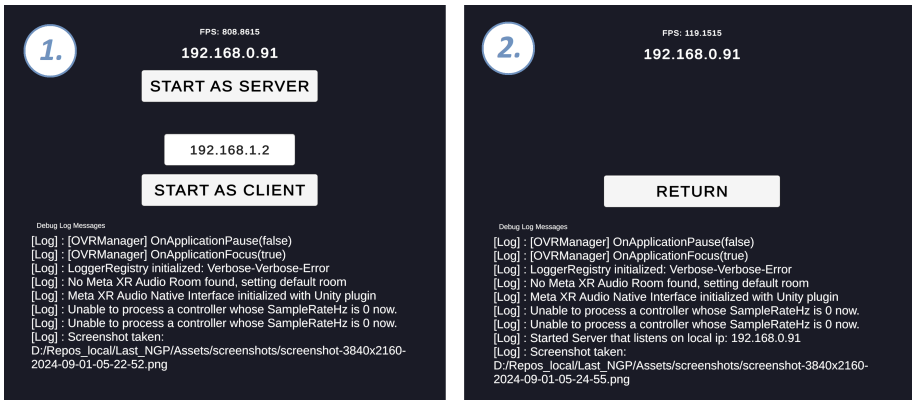



FIGURE 4.5: XR Connection Helper for Debugging and Testing
Shown is the Connection Helper Prefab, a 3D world UI, that enables to debug connection and measurements issues. *1.* Entry Point; Choose between of starting either as a Server or a Client. *2.* Example of a listening Server.

Measurement data was recorded in CSV files. Each data point was implemented as a struct, with an overridden `toString()` function to output the values as a comma-separated string. This approach was utilized within a custom generic class to store and write data from all measurement scenarios.

The data was stored in the persistent folder of the respective application, located under *storage/emulated/0/data/* on Android devices/ Accessing these files requires connecting the HMD to a PC via cable, followed by enabling USB sharing through a time-limited notification within the virtual environment.

All subsequent scenarios were based on this software setup.

### Timing of Client-Side Snapshot Processing

This section aims to demonstrate when a new snapshot is usable for the client, specifically investigating whether received snapshots are always provided to the next frame or if there are any discrepancies. Understanding this is crucial for minimizing client-side latency.

The client's update rate corresponds to the `Update()` function of `MonoBehaviours` or `NetworkedBehaviours`. The availability of the latest received snapshot and the ServerTime of the server were compared within the `FixedUpdate()` method.

The measurement focused on the `NetworkManager.Singleton.ServerTime` and the data synchronized via snapshots, including a `NetworkVariable<Int>` that increments with each server tick and a `NetworkTransform` component that increments in its position by 1 in each server `FixedUpdate()` loop. RPCs were excluded from this analysis.

The measurement was conducted with a server tick rate of 90 ticks/s, a server update rate of 120 Hz, and a fixed timestep of 0.01 ms. The client's update rate was aligned with the lowest refresh rate of the HMDs, set at 72 Hz. To detect any differences, the fixed update had to be executed at least once per frame. For more precise insights, a fixed timestep of 0.0034 ms was used, representing a quarter of the inverse of the update rate.

The results of the `NetworkVariable` and `NetworkTransform` measurements can be applied to all other components, such as the `NetworkAnimator` or own implementations transmitted via snapshots.

### RTT of RPCs, Network Variables and Unity Transport

Latency is crucial in any interaction within a virtual environment, whether involving the user's surroundings or virtual objects. To address this, the RTT of two fundamental concepts, the `RPC` and the `NetworkVariable`, are analyzed under various configurations. As the Unity Transport protocol provides the underlying network infrastructure for NGO, its RTT was additionally measured.

A fixed configuration is established to provide a baseline, consisting of a network tick rate of 90 ticks per second and update rates of 120 Hz for both the server and client. Variations are introduced by adjusting one parameter at a time: the server's tick rate is varied at 30, 60, 90, and 120 ticks/s; the server update rate at 30, 60, 90, and 120 Hz; and the client's update rate is aligned with the Meta Quest's refresh rates, tested at 72, 80, 90, and 120 Hz. The study concludes by focusing on the most realistic configuration and the best possible outcome. The combination of these measurements resulted in 30 datasets across 36 measurements. The measurements for the varying server and client update rates were conducted fully automated in sequence. Since the tick rate cannot be changed during runtime, a new server build was created and executed for each value. Although the client's tick rate is determined by the server, the client was reinstalled as a precaution.

For accurate implementation, it is crucial that the process is not a simple ping-pong exchange, where a ping triggers a pong and vice versa. Such a setup would reduce the sampling rate as latency increases. Instead, pings need to be sent at regular intervals, achieved by using `FixedUpdate()`.

The RTT of the RPC implementation follows this principle. The client sends an RPC containing an `int` identifier at regular intervals, necessary for later association in case of significant delays. The server receives this RPC and immediately returns an RPC with the same identifier. The client measures the elapsed time using `Time.realtimeSinceStartup` and records it.

The RTT measurement for the NetworkVariable implementation followed a similar approach. The client modifies its `NetworkVariable<int>` during the first executed `FixedUpdate()`, which is then sent to the server at the end of the frame. This change prompts the server to update its `NetworkVariable<int>`, which is sent back to the client with the next snapshot, triggering `OnValueChanged()` on the client. The client then calculates and stores the RTT.

The RTT of the underlying Unity Transport Layer was determined by sending a ping from the client to the server using the `GetCurrentRtt(ServerClientId)` function. The function call was invoked within the `FixedUpdate()`.

Due to the extensive data and number of diagrams generated, the analysis will focus only on the most relevant results, with additional data provided in the thesis appendix.

### Network Traffic and Packet Analysis

To understand actual bandwidth utilization and gain insights into underlying packet transmission mechanisms, a thorough understanding of network traffic is essential. Measuring network packet sizes is fundamental for estimating required bandwidth in various network scenarios. To accomplish this the bandwidth of RPCs, Network Variables, and the Network Transform component were measured.

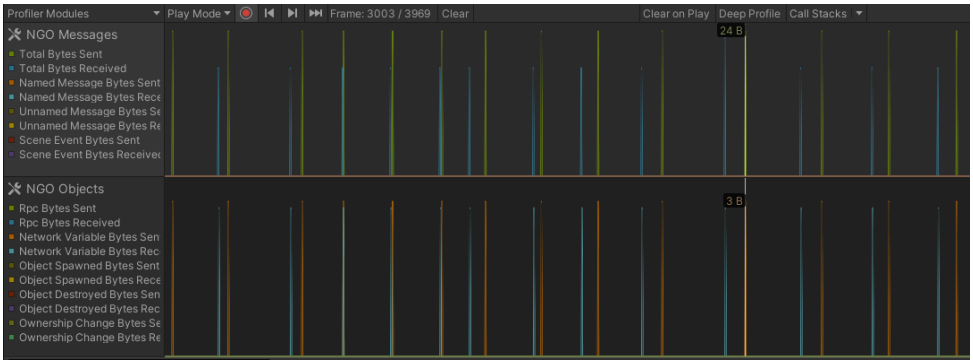
The measurements included RPCs and network variables for all basic unmanaged types, such as `byte`, `short`, `int`, `long`, `float`, `double`, `decimal`, and `char`. Additionally, the `FixedString512Bytes` data type, that represents an unmanaged UTF-8 string, was used to represent a larger data type.

Given the widespread use of the `NetworkTransform` component, its bandwidth consumption was a primary focus of this study. To quantify this, the network traffic generated by increasing numbers of `NetworkTransform` components was measured. Additionally, the potential for bandwidth savings through the use of half-precision floating-point numbers was investigated, albeit at the expense of accuracy. The ultimate goal was to determine the maximum theoretical bandwidth requirements imposed by the `NetworkTransform` component

All of these measurements are qualitative in nature, with each packet being sent and received ten times to ensure consistency, as packet sizes typically remain stable unless the payload changes. Notably, the maximum packet size configurable in the Unity Transport Component was not exceeded, thus preventing any fragmentation.

For these measurements, a different setup was used from what was previously described. The network traffic was analyzed on a single computer with both the client and server running locally, without any intermediary. The Unity "Multiplayer Tools"[36] package version 1.1.0 was first used to determine the outgoing payload of a network packet as displayed in the Unity Profiler. Subsequently, the exact network traffic on the loopback interface was recorded and analyzed using Wireshark version 4.2.6. Figure 4.6 demonstrates a sample analysis using both tools. The Unity Profiler highlights blue spikes representing the packets exchanged for server time synchronization. Wireshark, shown on the right, was used to identify and mark these network packets in gray.

(A) Unity Profiler




(B) Wireshark

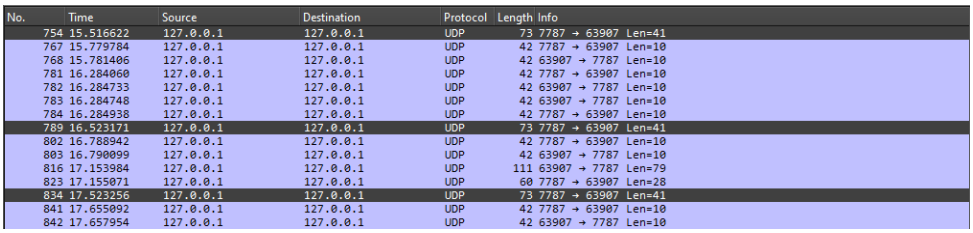



FIGURE 4.6: Example Workflow for Acquiring Network Packet Sizes

[36] https://docs-multiplayer.unity3d.com/tools/current/about/ (retrieved 01.09.2024, 9:42)

### Bandwidth Calculations

To enhance the practical utility of this study and to identify potential limitations across various scenarios, the knowledge gained regarding the size of network packets was applied to calculate the bandwidth requirements for specific example scenarios.

The theoretical bandwidth required for synchronizing a user with varying levels of detail was calculated by considering three cases, where each specified component was represented by a `NetworkTransform`:

1. Head and two hands.
2. Head, body, and two hands.
3. Head, body, two hands, and two feet.

In these cases, the `NetworkTransform` components only synchronize position and rotation.

Although avatars, typically synchronized using a single `NetworkTransform` component, were not examined in detail, it is understood that avatars are defined by their state and numerous associated animation variables. A precise calculation of the bandwidth requirements for avatars was not possible due to the absence of standardized avatar animations and the varying resolution of these animations. Nevertheless, the theoretical bandwidth for specific avatars can be approximated using the packet size overview provided in the appendix.

The bandwidth originating from a single user serves as the basis for all subsequent calculations. Building on the scenario of one user with a head, body, and two hands, bandwidth was calculated for increasing numbers of users in various virtual environments, such as virtual collaboration, classrooms, showrooms, presentations, and virtual concerts or events.

Further calculations related to Network Variables and RPCs were not conducted in this analysis.

### End-to-End Latency of Ownership Acquisition

This measurement scenario represents a common application in server-client architectures that utilize server authority, particularly in scenarios involving physical interactions. In such contexts, object manipulation is often restricted to the owner, who is also responsible for calculating the physics of the object. Consequently, a user wishing to move an object must first obtain ownership, which introduces latency. This scenario measures the end-to-end latency from the client's request to the actual possession of the object.

The request to change ownership is sent via a reliable RPC from the client, transmitting an `ulong` that represents the initiating user's ID. By checking separately the `IsOwner` Property of the requested object within the `FixedUpdate()` and `Update()` methods, the respective time could then be measured. To ensure accurate comparisons, `Time.realtimeSinceStartupAsDouble` was employed to measure time precisely, as the discrepancies between the two time measurements could be quite small. Comparing the times recorded by these methods can highlight potential optimizations when using this technique.

To assess the impact of varying fixed timesteps and server update rates, experiments were conducted under a standardized baseline configuration: a network tick rate of 90 Hz, a server update rate of 10 Hz, a server fixed timestep of 10 ms, a client update rate of 120 Hz, and a client fixed timestep of 1 ms. Exceptionally long fixed

timesteps of 5 and 10 seconds were employed to emphasize the influence of the fixed update rate on RPCs, particularly in this scenario. Additionally, end-to-end latency for ownership acquisition was measured across different server update rates to provide a practical context. The collected data was then visualized for analysis.

#### MTP Latency of a Moving Networked Physics-Simulated Objekt

In Unity, real-time physical objects are created by adding the `Rigidbody` component to them, which governs their physics behavior. NGO provides a corresponding component known as the `NetworkRigidbody`, which primarily sets the Rigidbodies of GameObjects into kinematic mode on all non-authoritative instances, disallowing clients to influence the physics locally. As the `NetworkRigidbody` does not synchronize translations, it requires a `NetworkTransform` to synchronize movements, rotations, and scaling across the network.

This experiment aims to investigate the end-to-end latency of the movement of a `NetworkRigidbody` using a `NetworkTransform` between the server and a client. Rather than measuring the RTT, the focus is on the end-to-end latency from a change occurring on the server to that change being reflected on the client. Accurately determining one-way latency is challenging due to the practical impossibility of synchronizing application time and network time precisely and continuously. To address this, the previously described method for measuring MTP latencies is utilized.

As previously mentioned, this method is based on changes in display color and is implemented as follows: A `Rigidbody` is catapulted back and forth from left to right in the simulation. A `Physics.RaycastNonAlloc`, which registers only one hit, is used to detect the passage of the object. When the catapulted object intersects with this ray, the display is turned white; otherwise, it remains black.

Applied to the client-server architecture, this means that the server owns the catapulted object and, therefore, calculates its physics. The server then sends a snapshot to all clients at each tick, containing the updated position data of this object. Upon receiving this snapshot, the client updates the corresponding values, in that case the position of the flying object. Since the server and then the client change their respective screen colors according to the latency being measured, this latency can now be determined using the method described earlier. Measurements were conducted at a server tick rate of 90 Hz and with both the server and client updating at a rate of 120 Hz. The collected data were subsequently visualized accordingly.

#### MTP Latency of a Moving Networked Physics-Simulated Objekt under Increasing Server Load

This performance benchmark builds upon the previously described measurement scenario and utilizes the same implemented Unity scene. In this scenario, a single, real-time simulated physical object moves from side to side on the server. The movement of this object is synchronized with the client, and the latency of this synchronization is measured. The same approach and measurement method were applied here, with the key distinction that the computational load on the server was progressively increased.

Specifically, the server spawned a certain number of real-time simulated physical objects at fixed intervals. To avoid incorrect measurements due to temporary heavy loads on the CPU during the spawning of these objects, a specific period of time was allowed to elapse before proceeding to the next interval.

This process continued until the `maximumAllowedTimestep` of 50 ms was reached, which represents the upper threshold of acceptable latency. All used objects were equipped with a `NetworkRigidbody` and a `NetworkTransform`. In this setup the `NetworkTransform` did not employ interpolation and was only responsible for synchronizing position and rotation. It is crucial to note that the `NetworkTransform` only synchronizes data when changes occur. To ensure continuous changes in position and rotation, a large rotating box was implemented in the background, into which the objects were spawned. This is demonstrated in fig. 4.7 below.

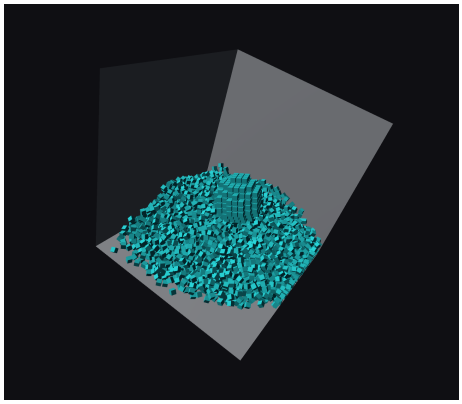


FIGURE 4.7: Rotating Cube to Keep Network Transforms Busy

During the experiment, special consideration was given to the issue of objects potentially being ejected from the box due to the discretization of real-time physics at a `fixedDeltaTime` ranging from 10 to 50 ms on the server. This problem arises because the physics engine may not accurately calculate collisions under these conditions. Although enabling continuous collision detection could address this issue, it would also impose additional computational demands on the server. To minimize the rendering overhead of these objects, occlusion culling was utilized.

A critical aspect of this experiment was the need to correlate the measured latencies with the spawned networked objects. This was accomplished by using the spike frames as reference points. Initially, the distance between the first and last spike was calculated. Following this, the theoretical number of frames required to complete the benchmark was determined by considering the maximum number of intervals, the time interval, and the duration of a video frame. The difference between the calculated frame count and the actual frame count was expected to be relatively small. This discrepancy was then divided by the number of intervals, and the resulting proportion was added to the number of frames per interval. This method ensured that any potential differences were evenly distributed across all intervals, thereby maintaining the validity of the results.

In addition to measuring the latency described above, the CPU utilization of both the HMD and the server was recorded in parallel using `Time.deltaTime`. The data collected through this measurement method were subsequently visualized.

### 4.2.4 Using ECS in NGO for Networked Simulated Physics XR Interactions

In light of the theoretical challenges outlined in the previous sections regarding the use of NCE in Unity XR projects involving physics-based interactions, further exploration of NCE alone is considered beyond the scope of this research, as it would require developing an entire XR system from scratch. Instead, as a hybrid approach

was proposed, the research will be focusing on implementing a bridging mechanism that allows communication between both architectures. This will investigate the feasibility of enabling physics-based interactions between object-oriented game objects and data-oriented entities, with a focus on demonstrating the practicality and usability of this approach.

# Chapter 5

# Results

## 5.1 Comparative Analysis of Wi-Fi 6e and Wi-Fi 5 Results

Figures fig. 5.1 and fig. 5.2 depict the measured utilization of Wi-Fi channels in the 2.4 GHz and 5 GHz bands, respectively, within the IXLab environment. In these diagrams, the colored hills associated with each channel represent Wi-Fi networks with a certain signal strength, measured in decibels relative to one milliwatt (dBm).

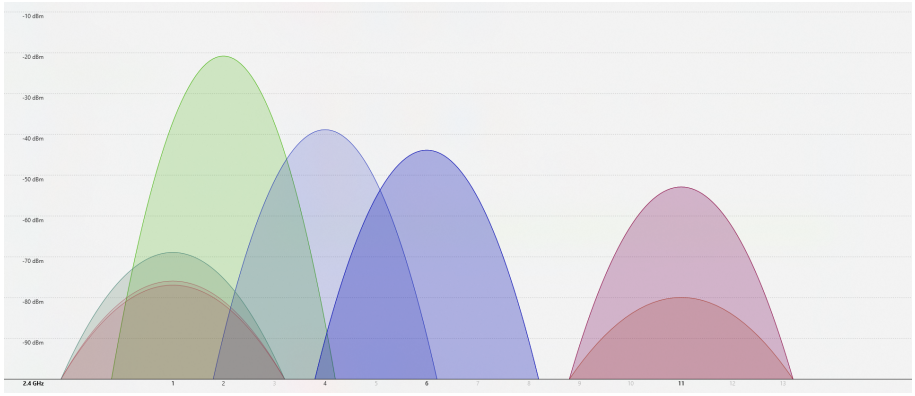


FIGURE 5.1: Utilization of the 2.4 GHz Band Channels of Access Points

FIGURE 5.2: Utilization of the 5 GHz Band Channels of Access Points

The data reveals a moderate utilization of the 2.4 GHz band with 8 Wi-Fi networks operating on 13 channels. In contrast, the 5 GHz band is utilized by 11 Wi-Fi networks, albeit on a significantly higher number of 38 measured channels. The Netgear router employed in this test supports the 2.4 GHz, 5 GHz, and 6 GHz bands. Consequently, it can be identified infig. 5.1 as the largest green hill on channel 2 with a signal strength of approximately -21 dBm. Similarly, in fig. 5.2, the same router can

be located as a prominent purple hill on channel 36, exhibiting an almost identical signal strength of -22 to -24 dBm. The second visible, slightly darker purple hill with a weaker signal strength of approximately -27 dBm corresponds to a mobile hotspot from a Samsung Galaxy S22, which was subsequently switched off in further tests.

Following the analysis of channel utilization, RTT measurements were conducted between the Meta Quest 3 and the server using two different routers. The results are depicted in fig. 5.3, which presents the RTT for each of the 50 measurements. The Netgear router exhibited an average RTT of 16.28 ms, while the TP-Link router recorded an average of 15.50 ms, resulting in a difference of 0.78 ms. The Netgear router displayed a slightly higher standard deviation of 4.19 ms compared to the TP-Link router's 3.89 ms. Although this difference is negligible, it indicates that the RTT values exhibit considerable variability around their respective means.

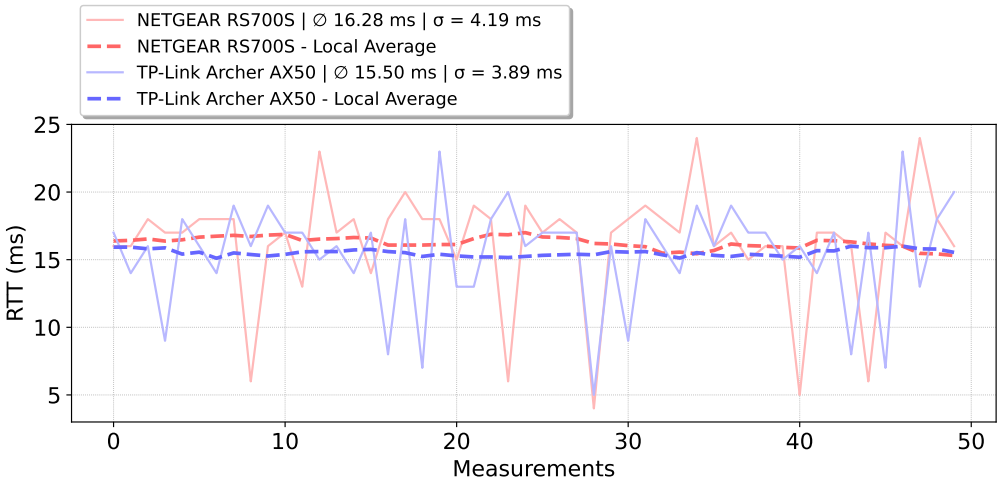



FIGURE 5.3: Comparison of RTTs Between the Meta Quest 3 and the Server when using Two Different Routers

It is important to note that ICMP ping measurements between the server and the HMD were inconsistent and unreliable. While the Windows `ping` command and the `hrPing` application occasionally returned values as low as 3 ms, many requests timed out, indicating significantly higher or even immeasurable latency. Given that the online functionalities did not exhibit such latency, these results appeared unrealistic.

Following the latency measurements, bandwidth was assessed. The downlink speed from the server to the HMD was measured first. Figure fig. 5.4 illustrates the measured bandwidth in Megabits per second (Mbit/s) over a 49-second interval for each router connection using UDP packets. The connection via the Netgear Router's 6 GHz network achieved an average of 2392.37 Mbit/s, approaching the advertised maximum downlink speed of the Meta Quest 3. The standard deviation of 1.52 Mbit/s indicates a highly stable bandwidth. Conversely, the connection via the TP-Link Router's 5 GHz network attained an average bandwidth of 956.98 Mbit/s, falling short of the stated 1200 Mbit/s downlink speed by 243.02 Mbit/s. Nevertheless, the bandwidth remained stable, as evidenced by a standard deviation of 0.59 Mbit/s.

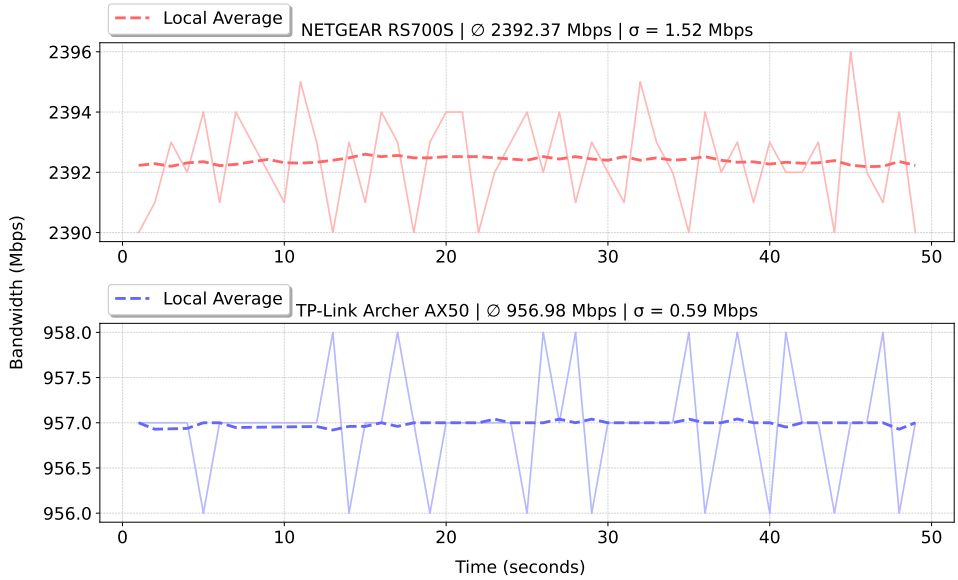



FIGURE 5.4: Meta Quest 3 Local Downlink Speeds of Two Different Routers

Secondly, the uplink bandwidths from the HMD to the server were measured. Figure 5.5 presents the measured bandwidth in megabits per second (Mbit/s) over a 300-second interval for connections established through each router using TCP packets. The connection via the Netgear Router's 6 GHz network achieved an average of 945.97 Mbit/s with a standard deviation of 72.94 Mbit/s, indicating unstable bandwidth. The connection via the TP-Link Router's 5 GHz network reached an average bandwidth of 434.39 Mbit/s. However, the bandwidth also fluctuated significantly, with a standard deviation of 31.92 Mbit/s. Notably, both average values deviate substantially from the theoretical uplink speeds of 2400 Mbit/s for 6 GHz and 1200 Mbit/s for 5 GHz. The increase in measurement time and the resulting larger number of data points are attributed to the previously observed lower bandwidth.

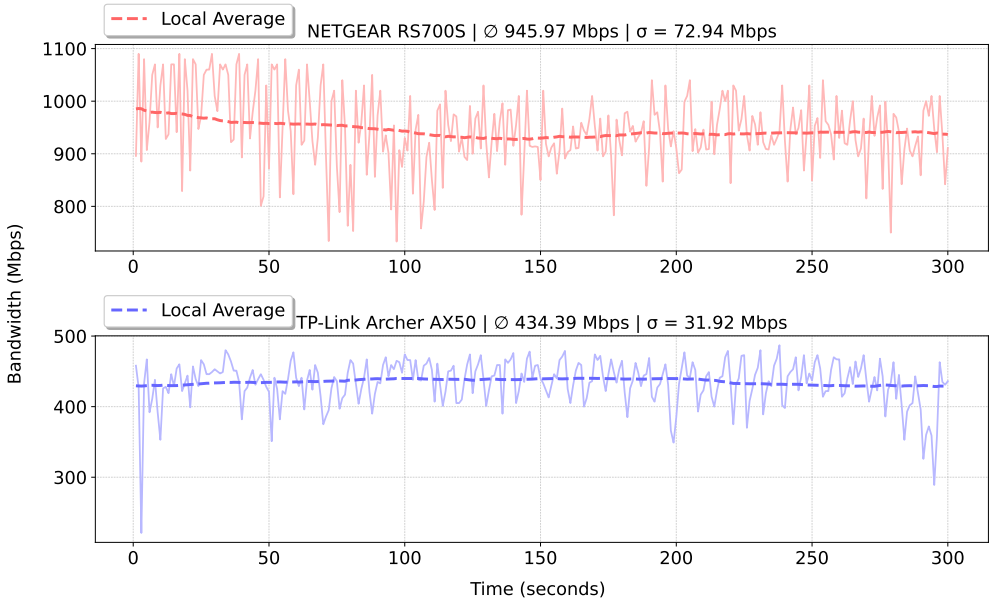



FIGURE 5.5: Meta Quest 3 Local Uplink Speeds of Two Different Routers

## 5.2 Netcode for GameObjects

### 5.2.1 Timing of Client-Side Snapshot Processing

This experiment demonstrates the precise timing at which new snapshot values become available to the client. The tick rate was set to 90 ticks per second, exceeding the client's update rate of 72 Hz, while the server operated at 120 Hz. Figure fig. 5.6 presents four consecutive client frames, each distinguished by a unique color. The x-axis represents time in milliseconds, and the y-axis indicates the current NetworkManager.Single.ServerTime in milliseconds. Each data point contains two additional pieces of information: the x-coordinate of a `NetworkTransform`'s position and the value of a server-side texttttNetworkVariable<int>. The x-coordinate of the `NetworkTransform`'s position is shown above the data point, and the value of the `NetworkVariable<int>` at its respective time is displayed below. A circle denotes a query made within `FixedUpdate()`, while a pentagon represents a query made within `Update()` of the NetworkBehaviour.

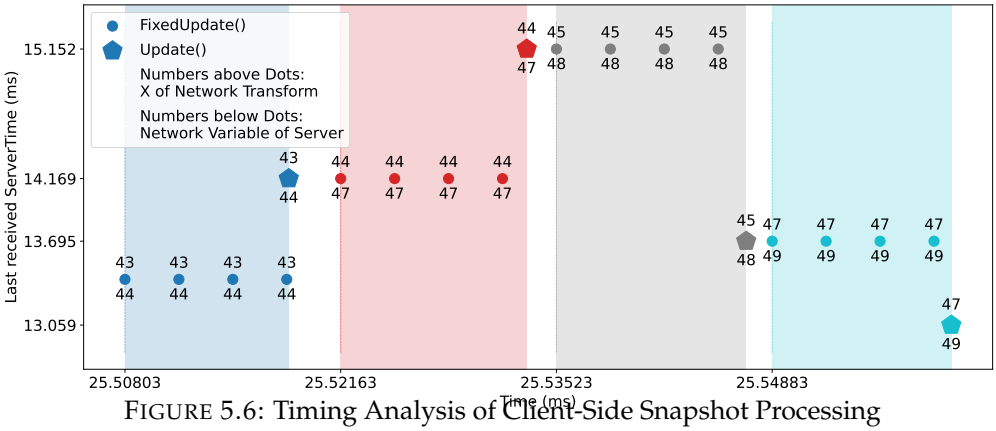



FIGURE 5.6: Timing Analysis of Client-Side Snapshot Processing

It can be observed that new values, such as the updated position of the `NetworkTransform` and the `NetworkVariable<int>`, are available to the client in the first `FixedUpdate()`. These values remain constant throughout the client's frame. Between the first (blue) and second (red) frames, as well as between the third (gray) and fourth (turquoise) frames, it can be seen that the values of the `NetworkVariables` and `Network- Transforms` increase differently. This is because the `NetworkVariable` is incremented by 1 with every server tick, and the position of the `NetworkTransform` is shifted by the Vector (1, 1, 1) in each `FixedUpdate()`. Since the network tick rate of 90 ticks/s is higher than the client's update rate of 72 Hz, and the server's `FixedUpdate` timestep is also 10 ms (100 Hz), jumps in the values can occur. As the server sends a new snapshot with every tick, it can be observed between the first and second frames that only the last received snapshot is accepted by the client. This demonstrates that the client only applies the values of the last received snapshot. Consequently, the client's update rate should not be lower than the server's tick rate.

In contrast to this snapshot-based synchronization, which occurs at each tick event, Remote Procedure Calls (RPCs) are processed and accepted with every frame event. Therefore, while snapshot synchronization aligns with the tick rate, RPCs offer more frequent updates, as they are transmitted independently of the tick events.

A further peculiarity is the `NetworkManager.ServerTime`, which only changes in the client's `Update()`. Research has shown that this `ServerTime`, in addition to the `NetworkManager.LocalTime`, is used to account for the different time perspectives

of the client and server, thus ensuring the most accurate synchronization of game objects and events, despite the inevitable network latency.[37]

Furthermore, a time measurement of the individual steps and the visualization from the NGO documentation[38] were used to trace the synchronization process of a measurement of the RTT via a `NetworkVariable<int>`. Since this synchronization is tied to snapshots, it can be extended to other components such as the `NetworkTransform`".

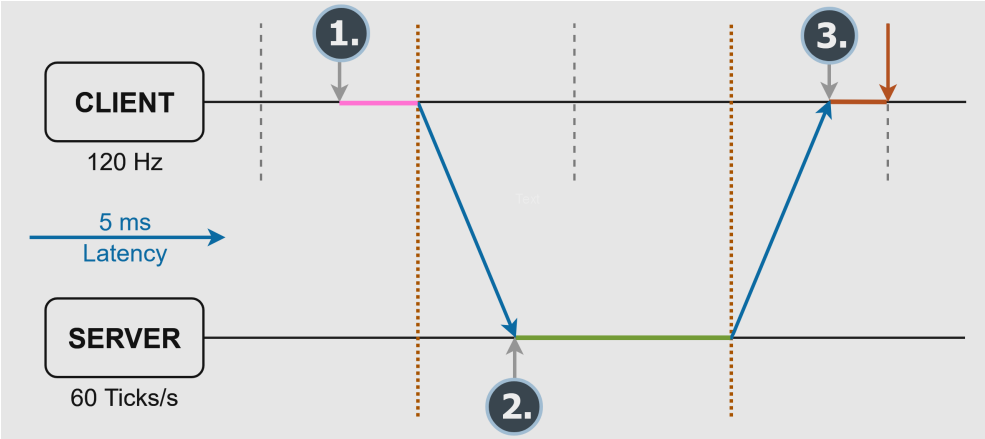



FIGURE 5.7: Timing Analysis of Client-Side Snapshot Processing
Steps 1, 2, and 3 are elucidated below.

The sequence diagram in fig. 5.7 presents an idealized synchronization process of an RTT measurement using a `NetworkVariable`. Assuming perfect synchronization between the client's update rate of 120 Hz and the server's tick rate of 60 Hz, with a network latency of 5 ms, the transmission is visualized using blue arrows. The gray dotted lines indicate the time intervals for the start and end of each client frame. Orange dotted lines representing the time interval of server ticks. The synchronization process is divided into three distinct steps:

1. The client initiates a single change to its `NetworkVariable` mid-frame, which triggers the dirty flag for that variable. The updated value is buffered until the next tick event of the `NetworkManager` (represented in pink), at which point the value is transmitted to the server.

2. The server receives this send information 10 ms after the last tick event, which triggers an `OnValueChange()` event for the client's `NetworkVariable`. Within this event handler, the server modifies its own `NetworkVariable` in a similar manner as the client. The updated value is buffered (depicted in green) and sent back to the client during the next tick event of the `NetworkManager`.

3. Upon receiving the server's snapshot 10 ms after its last tick event, the `OnValueChange()` event of the server's `NetworkVariable` is invoked. Although the updated value is immediately available, its visual representation can only be seen after next `Update()` cycle.

The outlined process illustrates the sequence in which data is received, transmitted, and buffered within a networked system. The first step, up until the server receives the information, broadly represents a synchronization event, such as that

[37] https://docs-multiplayer.unity3d.com/netcode/current/advanced-topics/networktime-ticks/ (retrieved 02.09.2024, 17:29)

[38] https://docs-multiplayer.unity3d.com/netcode/current/learn/rpcvnetvar/ (retrieved 01.09.2024, 11:24)

of a `NetworkTransform`. In this context, the RTT using a `NetworkVariable` was employed solely to demonstrate the flow of data during the synchronization process.

### 5.2.2 RTT of RPCs, Network Variables and Unity Transport

The 30 datasets detailed in the methodology were synthesized into 10 diagrams, comprising three for each RTT implementation and an additional diagram offering a comprehensive overview of the latencies for a particular configuration. To simplify the measurement process, it was chosen to measure the RTT. of the measurement process. All diagrams can be found in chapter A. These diagrams depict the RTT in milliseconds for each configuration with one variable, corresponding to a specific RTT implementation. The x-axis represents individual measurements, ordered sequentially as the tests were executed consecutively.

It should be noted that the measured values of the variable Server Update Rate SUR do not fully correspond with the CUR, making direct visual comparison challenging. The CUR values are based on the refresh rates of the Meta Quest 3, which explains the inconsistency.

The diagrams reveal "corridors" or "tubes", within which various levels or patterns can be observed. Since all measurements were performed by the client, the RTTs with the `NetworkVariable` are influenced by the CUR. For RTT via RPCs, the different update rates are crucial, as they are always sent at the start of a new frame and thus depend on both CUR and SUR. Consequently, these measurements, including the RTT of the Unity Transport, likely result from a combination of SUR and CUR, with the smaller value having a greater impact on the resulting RTT.

These corridors indicate that a lower CUR consistently worsens the RTT. This is evident across all three RTT types when the CUR is variable, except for the RTT of the Unity Transport in fig. A.9, where a CUR of 90 Hz (green) resulted in a higher average RTT of 31.07 ms compared to a CUR of 80 Hz (blue) at 30.67 ms. Given that the differences are minimal and both have a standard deviation of at least 4.44 ms, significant fluctuations are visually apparent in the diagram, suggesting measurement inaccuracies. It is still expected that a lower CUR would lead to worse RTTs.

For the RTT of an RPC, several observations can be made. A variable network tickrate has a minimal numerical effect on the RTT of an RPC, as shown in fig. A.1. Visually, the RTT corridor exhibits an upper limit of just over 50 ms and a lower limit of just under 42 ms, indicating a difference of approximately 8 ms, which reflects the inverse SUR or CUR of 120 Hz. Network latencies or synchronization issues result in outliers around these 8.33 ms intervals.

Figure A.2 illustrates the impact of varying SUR. The figure highlights a significant visual and numerical difference in the RTT as the SUR changes. At a SUR of 30 Hz (depicted in red), the average RTT reaches 107.29 ms with a standard deviation of 10.47 ms, indicating substantial variability. Doubling the SUR to 60 Hz (depicted in blue) reduces the RTT to 67.77 ms, accompanied by a lower standard deviation of 7.99 ms. Notably, around the 50th measurement, a sudden spike in RTT is observed, potentially caused by fluctuations such as temporary server or client update rate (CUR) instability or network disturbances.

At a SUR of 90 Hz, the RTT further decreases, more than halving from the initial 30 Hz configuration. At 120 Hz, the average RTT is 45.19 ms, with a standard deviation of 4.17 ms. Additionally, as the SUR increases, the RTT corridor narrows, shrinking from approximately 21 ms at 30 Hz to around 8 ms at 120 Hz. Visually, the peaks in the RTT graph exhibit fewer distinct levels as the SUR increases. For example, with a SUR of 30 Hz, five levels or four gradations in peaks are observable,

whereas, at 120 Hz, only two levels are visible, except for an outlier around the 25th measurement. These gradations result from the smaller client frame time, while the SUR defines the overall envelope for the RTT fluctuations. Increasing the SUR from 30 to 120 Hz results in a potential RTT reduction of 62.19 ms.

In section A.1, the influence of different client update rates (CUR) is depicted. The average difference in RTT between a CUR of 72 Hz and 120 Hz is 11.89 ms. As expected, increasing the CUR reduces the RTT for RPCs. However, the scaling of the CUR (from 72 to 80, 90, and 120 Hz) cannot be directly compared with the SUR results in fig. A.2. The difference in average latency between 90 and 120 Hz is only 0.86 ms, and the values for 120 Hz remain identical. This suggests that for these higher update rates, both SUR and CUR have an equally significant effect on the RTT of RPCs, consistent with theoretical expectations.[39].

The formation of the RTT for a `NetworkVariable` was previously explained in section 5.2.1 and visualized in fig. 5.7. The following section extends this explanation by analyzing the snapshot-based `NetworkVariable` measurements. In fig. A.4, the impact of the network tick rate on the RTT of a `NetworkVariable` is evident. At a tick rate of 30 Hz, the average RTT is measured at 68.48 ms, with a standard deviation of 13.54 ms, indicating significant variance, which is further emphasized by spikes at the beginning and four other points throughout the plot. These initial spikes are also visible at 60 Hz, but with no additional outliers. Interestingly, at 120 Hz, the RTT starts high, around 50 ms, and gradually decreases to approximately 43 ms by measurement 175, stabilizing with fewer spikes, indicating a more consistent RTT. Overall, increasing the tick rate positively impacts RTT, with the gains diminishing as the rate increases.

The influence of SUR is shown in section A.2. A significant difference of 43.13 ms is observed between SUR values of 30 and 60 Hz, while the difference between 60 and 90 Hz decreases to 13.71 ms. These large differences arise because the SUR is lower than the tick rate of 90 Hz. At 30 Hz, the server takes longer than a tick to process a network variable change, with occasional spikes still occurring at 60 Hz. Once the SUR reaches 90 Hz or higher, this effect disappears, and only the overall processing time is reduced.

In fig. A.6, the RTT of `NetworkVariable` is compared across different CUR. As expected, higher CUR values result in lower RTTs. However, the differences are smaller. At 30 Hz, the RTT is 61.51 ms with a standard deviation of 6.92, while at 120 Hz, the RTT is 51.48 ms with a standard deviation of 7.05. The standard deviations and the plot reveal noticeable fluctuations, suggesting that the impact of CUR on RTT behaves similarly to the SUR. Visually, no significant differences are observed due to the CUR steps.

Regarding the RTT from Unity Transport, the tick rate appears to have no influence, as seen in fig. A.7. There is no clear order among the tick rates, with all values fluctuating around a common mean of 24.54 ms. These consistent variations are likely due to the sampling rate. Although 300 measurements were taken, the temporal interval between them was precisely one frame of the CUR. A larger interval could improve measurement accuracy.

Finally, fig. A.8 demonstrates the effect of SUR on the RTT of Unity Transport. At a SUR of 30 Hz, which is three times smaller than the tick rate of 90 Hz, the measurements are leveled, following a slight hill-shaped trend, with higher latencies in the middle and lower latencies at the edges. Additionally, the RTT at 90 and 120 Hz is nearly identical, suggesting that block size may be related to the proximity of the

[39] https://docs-multiplayer.unity3d.com/netcode/current/advanced-topics/message-system/rpc/ (retrieved 04.09.2024, 21:19)

CUR. The client frequently queries the server, but the server has not yet responded, so the client receives the same buffered RTT from `GetCurrentRtt`. The results show a doubling of RTT at 30 Hz and an increase by a factor of 1.5 at 60 Hz. The average latencies for SUR values of 90 and 120 Hz are 26.81 ms, consistent with this pattern. The calculated RTT difference of 2.65 ms at 60 Hz and 53.62 ms at 30 Hz, along with standard deviations of 6.94 ms and 8.30 ms, strongly supports the validity of these observations.

Figure 5.8 compares the RTT of various transmission methods on a fixed baseline. The Unity Transport layer demonstrates the lowest latency, with an RTT of 26.72 ms. The RTT for a RPC is measured at 45.19 ms, while the RTT for a Network Variable is slightly higher, at 51.48 ms. This hierarchical structure of RTT values was anticipated. However, a notable deviation is observed in the RTT of the `NetworkVariable`, starting around measurement 125, where a peak occurs. Considering that the previously measured baseline RTT was 16.28 ms, as shown in fig. 5.3, these RTT values are relatively high. Therefore, they are not recommended for synchronizing real-time simulated physics due to the potential delay.

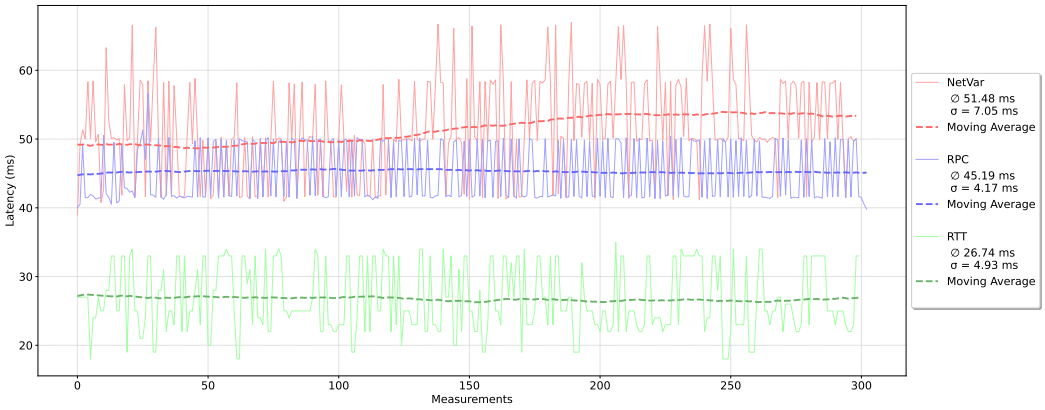



FIGURE 5.8: RTT of a RPC, Network Variable and Unity Transport Message at a Proposed Set of Parameters
Configuration: CUR = 120 Hz, SUR = 120 Hz, TR = 90 Hz

The analysis shows that the network tick rate does not impact the behavior of RPCs or the RTT within Unity's transport layer. However, it significantly affects snapshot-based synchronization mechanisms. Results demonstrate that the SUR must exceed the server's tick rate to prevent double processing times caused by timing misalignments. Additionally, the SUR has a substantial negative effect on RTT, particularly when it is lower than the network tick rate. The CUR exerts a similar influence on RTT, mirroring the effect of the SUR.

### 5.2.3 Network Traffic and Packet Analysis

#### RPC and NetworkVariable

The packet sizes for data synchronization via RPC and Network Variables, obtained according to the methodology outlined previously, are shown in table C.1. A brief summary of these measured packet sizes is provided in table 5.1, offering a clear and concise overview of the collected data for analysis. All measurements were conducted using the Unity Transport Component with its default settings, including a `Max Packet Queue Size` of 128 and a `Max Payload Size` of 6144 (bytes).

TABLE 5.1: Overview of the actual Size of Sent Network Packets by RPC and NetworkVariable
All values presented are expressed in units of bytes

| ValueType | Size | RCP | Network Variable |
|---|---|---|---|
| bytes | 1 | 95 | 95 |
| short | 2 | 95 | 95 |
| int | 4 | 95 | 103 |
| long | 8 | 103 | 103 |
| float | 4 | 95 | 103 |
| double | 8 | 103 | 103 |
| decimal | 16 | 111 | 111 |
| char | 2 | 95 | 95 |
| FixedString512 | 512 | 607 | 607 |

The first column in table 5.1 presents the transmitted unmanaged types associated with each transmission method. The second column indicates the size of the value types in bytes. Subsequent columns detail the actual sizes of the network packets for the respective transmission methods, either via RPC or Network Variable.

As illustrated in table C.1, Unity imposes a fixed header size of 31 bytes for both Network Variable and RPC transmissions. This was determined by subtracting the overall sent bytes from the payload size recorded in Wireshark. However, for the periodic time synchronization messages (TimeSyncMessage), this overhead was significantly lower, amounting to only 8 bytes. The NGO documentation provided no further clarification regarding these overheads.

Upon further analysis of Unity's individual payloads, it becomes evident that the "Overall" payloads are packed in increments of 8 bytes. For instance, when comparing the network variable transmission of a short and an integer, the overall message size is 8 bytes larger, despite only 2 additional bytes being necessary. This discrepancy of 6 bytes could potentially become problematic with frequent usage. However, it is unclear if this 6-byte overhead is consistently present, or if these additional bytes are filled when other values require synchronization. No documentation clarifying this behavior was found.

Additionally, RPCs were found to incur a higher object overhead, as they include not only the parameters but also the relevant object, component, and method information associated with the RPC call[40]. Notably, even an empty RPC generates a 10-byte message. When an RPC is broadcast to all, an additional "Proxy Message" of 15 bytes is produced.

Additional insights were gained during the measurements:

- Unity only synchronizes values that have been flagged as "dirty", i.e., those that have been changed. For example, even if position, rotation, and scale are meant to be synchronized by a network transform, no packets are sent unless a specific change occurs. If only the Y-axis value changes, only that value is transmitted.

[40] https://docs-multiplayer.unity3d.com/netcode/current/advanced-topics/messaging-system/ (retrieved 03.09.2024, 23:17)

- A constant packet exchange between users was detected, as shown in fig. 4.6 (B), using Wireshark. This traffic, however, is not reflected in the Unity Profiler's "NGO Messages" or "NGO Object" modules. Server and client exchange UDP packets of 42 bytes, with a 10-byte payload. This pattern, undocumented in NGO, suggests a periodic ping to check the server and client's connectivity. In tests with multiple clients, it was confirmed that each client establishes this traffic independently. It appears that both the server and client independently ping each other approximately once per second. This results in an average of 4 to 6 additional packets per user per second, corresponding to a throughput of 168 to 252 bytes per second per user.

### Network Transform

The measured sizes of the network packets sent by the `NetworkTransform` Component under various configurations are presented in table 5.2. All sizes shown are in bytes. The first column, "Combination" represents the respective transform properties, such as position, rotation, and scale. The column "Theoretical Payload" indicates the size of the datatype. Since transform properties, represented as Vector3, consist of three floats, each single vector has a size of 12 bytes. The following column shows the overall message size as displayed by Unity's profiler module. In the second-to-last column, the actual packet sizes of the UDP packets, as measured with Wireshark, are listed. The "Data Payload Proportion" (DPP) column indicates the proportion of the data payload as a percentage of the maximum achievable payload without compression. Since UDP packets are being used, a constant header size of 32 bytes is subtracted from the packet size, and the data payload is then related to this value. All measurements were conducted using the Unity Transport Component with its default settings.

TABLE 5.2: Overview of Sent Network Packets by the NetworkTransform Component
All values presented are expressed in units of bytes

| Combination | Data Payload | Message Size | Resulting Packet size | Data Payload Proportion (%) |
|---|---|---|---|---|
| Position | 12 | 40 | 103 | 16.90 |
| Rotation | 12 | 40 | 103 | 16.90 |
| Scale | 12 | 40 | 103 | 16.90 |
| Position, Rotation | 24 | 56 | 119 | 27.59 |
| Position, Rotation, Scale | 36 | 64 | 127 | 37.89 |

The data presented in Table 5.3 reveals that synchronizing individual transform properties, such as position, can result in a disproportionately high bandwidth consumption relative to the size of the data being transmitted. This is reflected in the low DPP values associated with such scenarios. However, as additional transform properties are synchronized, the DPP tends to increase, suggesting a more efficient utilization of bandwidth.

The primary reason for the increased DPP is that the *Data Payload* did not necessitate the creation of additional UDP packets. Consequently, the *Data Payload* increased relative to any overhead of Unity and the UDP packet. "The DPP excludes the UDP overhead and inherits the previously shown fixed Unity overhead of 31

bytes (cf. table C.1 in chapter C). Moreover, since the measurements were conducted on a single component, the overhead associated with assigning the *Data Payload* within Unity remained relatively constant at 28 bytes, except for combined position and rotation data, which incurred an overhead of 32 bytes. Despite this improvement, it is evident that Unity's underlying system introduces a significant overhead. For a fully synchronized `NetworkTransform`, this overhead constitutes 62.11% of the total packet size. When only the position is synchronized, the overhead increases even further to 83.10%.

The analysis of network traffic for `NetworkTransforms` under varying quantities and with the use of half-precision floating-points is detailed in table 5.3. The first column specifies the number of `NetworkTransforms` involved. The "Data Payload" column, consistent with the earlier discussion, refers to the total size of the synchronized data. The subsequent columns provide the payload size as determined by Wireshark and the actual packet size. Column five offers the DPP for the actual packet size for full-precision floating-points. In the last two columns, the actual packet size when using half-precision floating-points for position and rotation is recorded. The final column then shows the corresponding DPP for this reduced data payload.

The reduction in packet size through half-precision floating-points for position and rotation was achieved through an adjustment in the `NetworkTransform` component, leading to lower data payloads with less accuracy of the synchronized data.

TABLE 5.3: Overview of the actual Size of Sent Network Packets by Network Transforms
All values presented are expressed in units of bytes

| Count | Data Payload | Resulting Payload | Resulting Packet size | DPP (%) | Half-Float Precision | DPP (%) |
|---|---|---|---|---|---|---|
| 1 | 24 | 87 | 119 | 27.58 | 103 | 16.90 |
| 2 | 48 | 119 | 151 | 40.33 | 127 | 25.26 |
| 5 | 120 | 223 | 255 | 53.81 | 191 | 37.73 |
| 10 | 240 | 399 | 431 | 60.15 | 295 | 50.84 |
| 25 | 600 | 895 | 927 | 67.03 | 615 | 51.45 |
| 50 | 1200 | 1886 | 1950 | 63.62 | 1167 | 52.86 |
| 100 | 2400 | 3629 | 3725 | 66.13 | 2350 | 55.27 |
| 1000 | 24000 | 33,990 | 34,822 | 70.60 | 23,302 | 51.49 |

Table 5.3 provides an overview of the actual sizes of network packets transmitted as the number of `NetworkTransform` components synchronizing position and rotation is increased. Consistent with the findings in table 5.2, a larger data payload generally correlates with a more efficient utilization of the *Resulting Payload*. This is primarily attributed to the constant overhead of 63 bytes per UDP packet and the slightly variable overhead associated with assigning the payload within Unity. It is expected that this assignment overhead increases with the number of `NetworkTransform` components.

A slight drop in the DPP is observed when the number of `NetworkTransform` components reaches 50. This can be attributed to Unity Transport adhering to the Maximum Transmission Unit (MTU) as defined in RFC 791[41] and further elaborated

[41] https://datatracker.ietf.org/doc/html/rfc791 (retrieved 04.09.2024, 10:53)

in the Unity Transport documentation[42]. Measurements have shown a consistent maximum size of 1355 bytes for UDP packets. Since 50 `NetworkTransform` components exceed this limit, the data must be fragmented across multiple UDP packets. For instance, with 50 `NetworkTransforms`, one packet is filled to the maximum capacity of 1355 bytes, while the second packet was measured at 595 bytes. The decrease in DPP occurs because the second packet has a lower DPP compared to the first, leading to an overall reduction.

To investigate this further, a measurement was conducted with 1000 `NetworkTransform` components to approximate the maximum achievable DPP. With a total *Data Payload* exceeding 24000 bytes, 26 UDP packets were transmitted. Notably, 25 of these packets were 1355 bytes each, while the last packet was only 947 bytes. Consequently, the average DPP reached 70.60%, closely approaching the theoretical maximum.

The DPP metric allows us to estimate the *Resulting Payload* size for any given number of `NetworkTransforms` under the specified conditions. For a conservative estimate, we can consider the lowest DPP value in Table 5.3. For a large number of `NetworkTransform` components exceeding 1000, a DPP of 70% can be safely assumed, providing a margin of error for unforeseen variations.

When comparing the data with half-precision floating points, which effectively halve the data payload, a reduced bandwidth demand is observed. The average DPP is 13.09% lower than that for full-precision floating point synchronization. While this indicates a less efficient use of payloads, the total bandwidth for 1000 `NetworkTransform` components is 33.08% lower. This reduction is largely due to the smaller payload, which requires fewer UDP packets, thereby also decreasing the overall UDP header overhead.

Furthermore, rotations can be synchronized using quaternions, which require 16 bytes, 4 bytes more than a `Vector3`. However, with a more efficient compression algorithm, this size could potentially be reduced to 4 bytes, which is even smaller than when using half-precision floating points.

An additional observation was that, with the "Use Unreliable Deltas" Option of a`NetworkTransform` disabled, a 60-byte packet was sent every tick. This suggests the presence of an acknowledgement packet in response to the recently received data. Enabling "Use Unreliable Deltas" option introduces the risk of data loss, as these acknowledgements are not sent.

The findings suggest a direct correlation between the size of the *Data Payload* $S_{DP}$, determined by `NetworkTransform` components, and the *Resulting Packet Size* $S_{RPS}$. This relationship can be approximated based on the DPP and the constant UDP packet size of 1355 bytes.

A quadratic regression, based on the data from table 5.3, was employed to model the connection between the *Data Payload* and the *Resulting Payload*. The resulting relationship is detailed in eq. (5.1) and visualized in section 5.2.3.

$$DPP(x) = -0.00000391x^2 + 1.50835135x + 40.16607809 \tag{5.1}$$

[42] https://docs-multiplayer.unity3d.com/transport/current/utp-2-faq/ (retrieved 04.09.2024, 11:02)

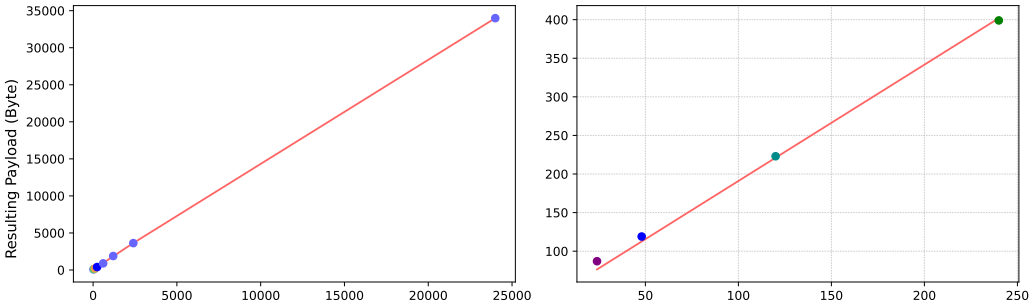



FIGURE 5.9: Linear Regression of DPP

An accuracy of eight decimal places was used to emphasize the presence of a quadratic relationship. However, this effect is so minimal that it would only influence the overall relationship at unrealistically high numbers of `NetworkTransform` components. Therefore, a linear relationship is more likely. Compared to the DPP values in table 5.3, this linear relationship exhibits a relative error of 3.39%. This error is an average and is relatively high due to the limited amount of data. For small data payloads, the relative error is significantly higher, as evidenced by a 12.22% relative error for a 24-byte payload. In contrast, the error decreases to 0.8% for a 120-byte payload and to a mere 0.03% for a 2400-byte payload.

By combining the relationship of $DPP(x)$ with the UDP overheads of each respective 32 bytes, eq. (5.2) illustrates the correlation between the *Resulting Packet Size* $S_{RPS}$ and the size of the *Data Payload* $S_{DP}$. Since the DPP encompasses the constant Unity overhead of 31 bytes, the maximum *Resulting Packet Size* $S_{MRPS}$ represents the maximum size of a UDP packet exclusive of its 1323-byte UDP header.

$$S_{RPS}(S_{DP}) = DPP(S_{DP}) + 32\ \text{Byte} \cdot \left\lceil \frac{S_{DP}}{S_{MRPS}} \right\rceil \tag{5.2}$$

The measurement results indicate that Unity introduces a significant overhead relative to the size of individual data types. Additionally, there is a lack of documentation explaining the composition of network packets and the mechanisms Unity employs to optimize bandwidth.[43] This gap in documentation could hinder further optimizations in network traffic handling and efficiency.

### 5.2.4 Bandwidth Calculations

As outlined in the methodology, the following overview presents various practical scenarios based on packet sizes. Table 5.4 details the required bandwidth in kilobytes per second (kbps) for tick rates of 30, 60, 90, and 120 Hz, respectively, depending on user complexity. User complexity is defined by the number of freely moving, synchronized objects, specifically the count of `NetworkTransform`. The presented values were obtained using unreliable delta synchronization, where acknowledgment packets of 60 bytes are omitted. For instance, using a tick rate of 60 Hz for a user with a head, body, and two hands would increase the required bandwidth of 13.705 kbps by 3.6 kbps to 17.765 kbps. Furthermore, the TimeSync message of 73 bytes per second was included. Additionally, the basic, non-profiled packet traffic of a user was considered, assuming six exchanged packets per second, resulting in a data traffic of 252 bytes/s. This yields an extra of 0.325 kbps for each user. Through a customized `NetworkTransform` component, is was guaranteed that each user has

[43] https://docs-multiplayer.unity3d.com/netcode/current/components/networktransform/ (retrieved 03.09.2024, 23:58)

ownership of their personal NetworkObjects. This is described in the documentation of NGO.[44]

TABLE 5.4: Bandwidth Requirements of Varying Amounts of User Body Parts.
All values presented are expressed in kbps

| Body Parts | 30 Hz | 60 Hz | 90 Hz | 120 Hz |
|---|---|---|---|---|
| Head, two hands | 6.055 | 11.785 | 18.225 | 23.245 |
| Head, body, two hands | 7.015 | 13.705 | 20.395 | 27.085 |
| Head, body, two hands, two feet | 7.975 | 15.625 | 23.275 | 30.925 |

As expected, the bandwidth requirements, presented in table 5.4, increased with both the server tick rate and the number of body parts per user that needed to be synchronized. Building upon the basic user model with a head, body, and two hands, table 5.5 outlines the scenarios detailed in the methodology, varying in terms of user count and server tick rate. The calculations are based on synchronization using unreliable deltas and include the fundamental user packet traffic. Notably, each user has exclusive ownership of their `NetworkObject`. These findings are derived under idealized conditions and pertain to entirely virtual environments, as in MR scenarios where only virtual objects are employed, users often do not require explicit representation. It is crucial to recognize that the theoretical bandwidth demand comprises two overlapping components:

1. The accumulated bandwidth demand of individual users to the server.
2. The bandwidth demand for transmitting the server's snapshot to all clients.

As the number of users increases, the server snapshot scales accordingly, leading to a proportional increase in bandwidth requirements. Given that network packets are sent at a fixed interval, every tick, a linear relationship can be established. Equation 5.3 derives the overall bandwidth demand $B$ as a function of the tick rate $T$, the individual bandwidth consumption of each client $B_U$, and the bandwidth required to broadcast a snapshot to all users $B_S$.

For $B_S$, the calculation from eq. (5.2) denoted as $S_{RPS_S}$ can be employed. Here, $S_{DP}$ represents the total size of the synchronized `NetworkTransform` components across all clients. $S_{DP}$ is composed of the number of users $N_U$, the individual number of `NetworkTransform` components per user $N_{NT_U}$, and $S_{NT}$, the maximum *Data Payload* of a single synchronizable `NetworkTransform` component. Since this snapshot is broadcast to all users, the *Resulting Packet Size* must be multiplied by $S_U$.

The bandwidth consumption of a single user $B_U$ is similarly derived from the *Resulting Packet Size* in eq. (5.2) ans is labeled as $S_{RPS_U}$. $S_{DP}$ in this case represents the total size of the synchronized `NetworkTransform` components of a single client, which is calculated as described above, with $N_U = 1$. Additionally, a constant bandwidth consumption of 0.325 kbps per user must be considered

$$\begin{aligned} B &= T \cdot (B_S + B_C) \\ B_S &= N_U \cdot S_{RPS_S}(N_U \cdot N_{NT_U} \cdot S_{NT}) \\ B_U &= N_U \cdot (S_{RPS_U}(N_{NT_U} \cdot S_{NT}) + 325\,\text{byte}) \end{aligned} \tag{5.3}$$

[44] https://docs-multiplayer.unity3d.com/netcode/current/components/networktransform/ (retrieved 04.09.2024, 18:23)

The goal of this section is to incorporate the context described above into the relevant equations and simplify them accordingly. Since the value of $S_{RPS_U}(96byte)$ with a has been empirically determined to be 215 bytes through previous measurements, this specific value is used here. Otherwise, a value of 216.97 byte would be assumed. In the following calculations, rounding is performed only at the final step to avoid the introduction of rounding errors that could affect the overall results. The equation for $B$, derived from eq. (5.4), is employed to calculate the bandwidth for the scenarios depicted in table 5.5. The unit of $B$ is bytes per second, as $T$ is given in hertz ($Hz = s^{-1}$).

$$\begin{aligned} N_{NT_U} \cdot S_{NT} &= 4 \cdot 24 \text{ byte} = 96 \text{ byte} \\ B_S &= N_U \cdot S_{RPS_S}(N_U \cdot 96 \text{ byte}) \\ B_U &= N_U \cdot (215 \text{ byte} + 325 \text{ byte}) = N_U \cdot 540 \text{ byte} \\ B &= T \cdot N_U \cdot (S_{RPS_S}(N_U \cdot 96 \text{ byte}) + 540 \text{ byte}) \end{aligned} \tag{5.4}$$

The relationship is illustrated graphically in fig. 5.10 below, where a clear non-linear correlation becomes evident.

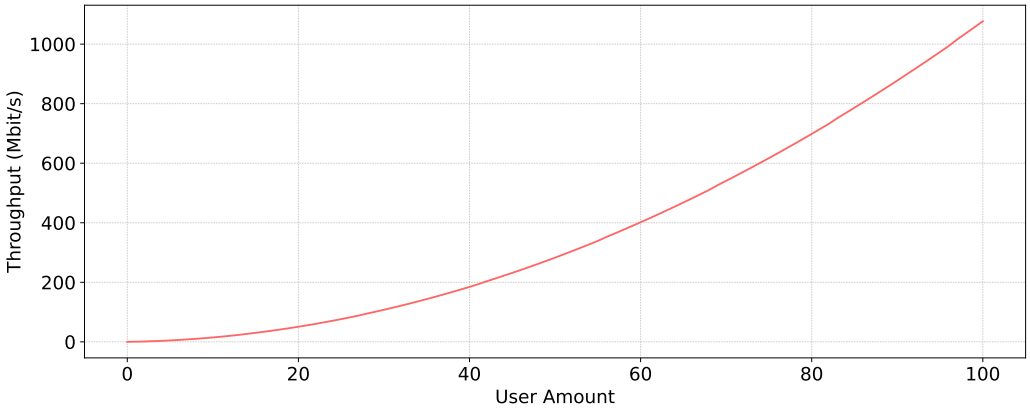



FIGURE 5.10: User-Dependent Throughput Demands for Synchronized Head, Body, and Hand Movements
Calculation based on eq. (5.3) with $S_{NT} = 24$, $N_{NT_U} = 4$, $S_{MRPS} = 1323$ byte

TABLE 5.5: Throughput Requirements of Different Multi-User XR Scenarios
A user's body is defined by a head, two hands and a body. Values are rounded up and given in Megabyte per second

| Scenario | User Amount | 30 Hz | 90 Hz | 120 Hz |
|---|---|---|---|---|
| Office | 5 to 10 | 0.21 to 0.63 | 0.41 to 1.26 | 0.81 to 2.51 |
| Classroom | 12 to 35 | 0.86 to 6.02 | 1.72 to 12.04 | 3.43 to 24.08 |
| Showroom | 15 to 50 | 1.27 to 11.89 | 2.53 to 23.77 | 5.06 to 47.54 |
| Presentation | 20 to 150 | 2.14 to 99.01 | 4.27 to 198.02 | 8.54 to 396.03 |
| Concert | 100 to 1000 | 45.16 to 3360.11 | 90.32 to 6720.21 | 180.63 to 13440.41 |

The presented scenarios are based on idealized conditions that may not fully reflect real-world scenarios. Factors such as interference-free environments and consistent router connection quality can adversely affect bandwidth and latency. Additionally, it is important to note that the unit used here is megabytes per second (Mbps), while manufacturers typically specify bandwidth in megabits per second (Mbit/s). This distinction is crucial when interpreting the results.

For instance, a concert with 1000 attendees under the current conditions would require approximately a throughput of 13440.41 Mbps or 107.52 Gbit/s. This exceeds the theoretical maximum throughput of a Meta Quest 3 by a lot. This scenario would even not be possible when using Wi-Fi 7 with a maximum throughput of around 46 Gbit/s.[45] Assuming a throughput of 800 Mbit/s, we observe in Figure 5.10 that 85 concurrent users (CCU) necessitate such bandwidth.

Based on the data presented in this section, along with other relevant factors, suggests that a complex, synchronized real-time physics simulation involving numerous objects and multiple users could potentially saturate the bandwidth capacity of a state-of-the-art HMD like the Meta Quest 3 when utilizing NGO. However, for real-time physics-based spatial interactions, bandwidth may not be the primary limiting factor. These interactions typically involve a relatively small number of synchronized objects, and latency is often a more critical concern.

As stated by the calculation of the overall required bandwidth $B$ in eq. (5.3), accurately assessing bandwidth needs can be challenging. Table 5.5 presents an approximation of bandwidth requirements for different user scenarios. The relationships presented in this table can be extended to provide more specific estimates for individual use cases.

### 5.2.5 End-to-End Latency of Ownership Acquisition

The concept of ownership acquisition is central to the server-authoritative paradigm, where only the owner of an object possesses the authority to modify it. The process of requesting and transferring ownership from the current owner to another entity is a critical component of object interactions in networked XR applications. The measured end-to-end latencies in this study represent the time elapsed for a successful Ownership Acquisition via a RPC.

Figure 5.11 visualizes the end-to-end latency of Ownership Acquisition over 300 measurements, based on the baseline configuration detailed in the methodology and employing two variable server fixed timesteps of 5 and 10 seconds.

[45] https://www.wi-fi.org/beacon/the-beacon/wi-fi-7-advanced-connectivity-for-the-next-generation (retrieved 07.09.2024, 9:47)

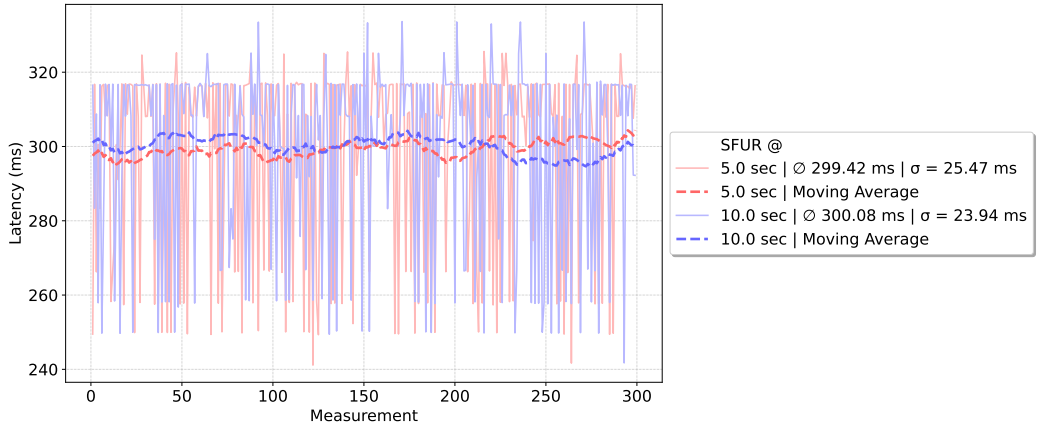



FIGURE 5.11: Comparison of the End-to-End Latency of Ownership Acquisition by RPC under Two Different Fixed Timesteps

A significant variance in end-to-end latency is evident, as highlighted by the standard deviations of 25.47 ms and 23.94 ms for fixed timesteps of 5 and 10 seconds, respectively. The client update rate is also discernible through the fine-grained peaks occurring at intervals of 8.33 ms. However, when comparing the averages of 299.42 ms and 300.08 ms for fixed timesteps of 5 and 10 seconds, respectively, the difference of 0.66 ms is negligible in relation to the overall latency. Thus, it can be concluded that the server's fixed update rate has a negligible impact on Ownership Acquisition latency in this scenario.

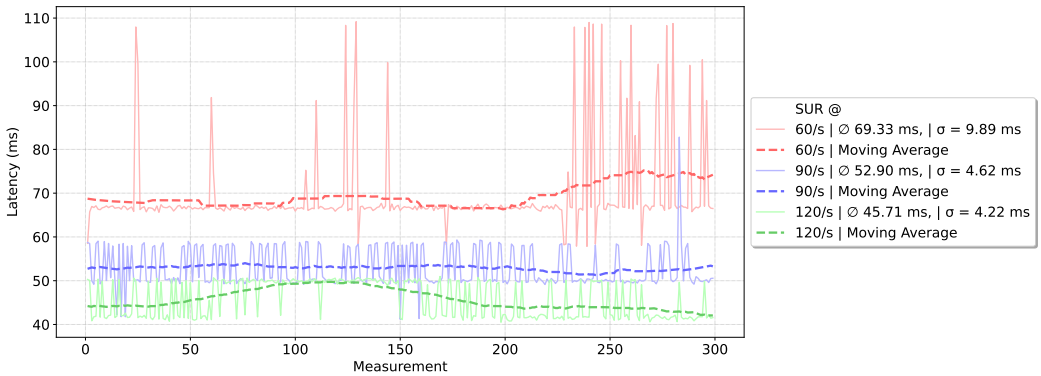



FIGURE 5.12: Comparison of the End-to-End Latency of Ownership Acquisition by RPC under Different Server Update Rates

Figure 5.12 employs three distinct server update rates: 60, 90, and 120 Hz. In alignment with previous findings, the end-to-end latency of RPCs decreases as the server update rate increases. Consequently, the measured end-to-end latencies of Ownership Acquisition also exhibit this trend.

Notably, there are prominent peaks in the data between measurements 230 and 300. Comparable peaks were detected at approximately measurements 25 and 125. Their similar magnitudes suggest a recurrent, albeit random, issue. This unexpected behavior could be attributed to lags within the Unity Editor, as the spikes are of nearly identical height, ruling out external network influences. However, the precise cause of these elevated latencies remains uncertain.

These results suggest that utilizing the highest possible server update rate is beneficial for maintaining low latency when users request ownership changes.

### 5.2.6 MTP Latency of a Moving Networked Physics-Simulated Objekt

Figure 5.13 illustrates the MTP latency of a single simulated physics object, measured over 250 data points collected within a time frame of 5 minutes and 30 seconds. The data exhibits an average latency of 35.36 ms with a standard deviation of 6.78 ms. A linear regression line, along with a moving average, has been included to provide visual context and highlight any underlying trends. Linear regression was employed instead of a simple average to visualize potential issues such as garbage collection, memory leaks, or inconsistencies. This approach allowed for a more granular analysis.

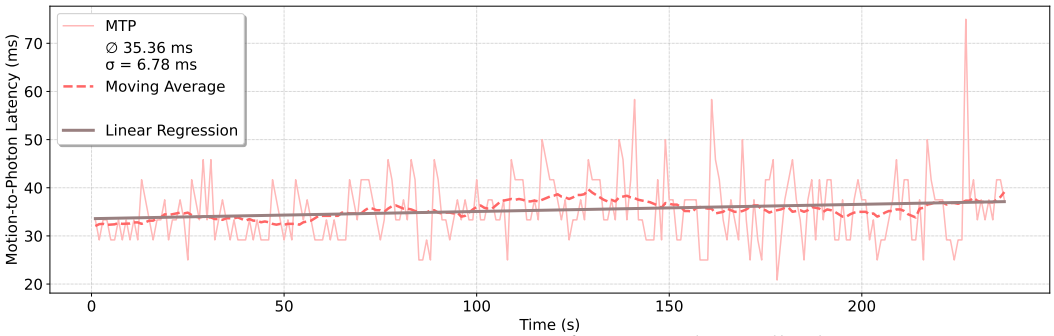



FIGURE 5.13: MTP Latency of a Network Rigidbody

The positive slope of the line, with $m = 0.01\ ms/measurement$, indicates an upward trend of 2.5 ms over the entire measurement period. A closer examination of the MTP latency reveals a noticeable increase in variance in the second half of the measurements compared to the first. Particularly between measurements 120 and 175, there are significantly higher peaks, while troughs are less frequent and less pronounced. A notable spike of almost 75 ms around measurement 30 contributes substantially to the overall upward trend and the relatively high standard deviation. These fluctuations, attributed to external factors and system variability, cannot be linked to a specific underlying issue.

Given a CUR and SUR of 120 Hz each and a tick rate of 90 Hz, the lowest latency in this setup would be the combined network latency of half the RTT of 8.14 ms and the tick rate latency of 11.11 ms, resulting in a total of 19.25 ms. This represents a difference of 16.11 ms from the overall average.

Considering the aforementioned variability, a small hill can be observed between measurements 75 and 150. While this cannot be definitively categorized, it suggests a constant deviation.

Further analysis reveals that the peaks differ by approximately 4.17 ms, which can be attributed to the inherent issue of discretization, as previously discussed in the introduction of the measurement method employed in this study (see section 4.2.2). In the case of constant latency, averaging would not be possible, and the average would instead be a multiple of the capture frequency. The actual average MTP latency could therefore be up to 4.17 ms higher, placing it within a range of 35.36 to 39.53 ms, considering the variability. Analogously, the difference would then be in a corridor of 16.11 to 20.28 ms.

Given these potential influences, latency variations, and the imprecision of the measurement method, a precise value cannot be determined. Considering the possible differences, it can be concluded that the most significant factor is the actual network latency. In a controlled network environment, the MTP latency of 35.36 to 39.53 ms was found to be sufficient for networked physics-based interactions. These

findings highlight the applicability of NGO for networked physics-based XR applications.

### 5.2.7 MTP Latency of a Moving Networked Physics-Simulated Objekt under Increasing Server Load

This measurement follows the same method as described in section 5.2.6. Data was collected over a period of 13 minutes and 45 seconds, resulting in 798 individual measurements. fig. 5.14 presents the average MTP latency in milliseconds, based on the number of networked physics-simulated objects, ranging from 0 to 1200, on the server. The MTP values were averaged because every 5 seconds, 15 objects were added, effectively smoothing out any overhead caused by spawning objects. The MTP latency, measured in frames, is illustrated in fig. 5.15, highlighting the importance of examining the progression of averaged MTP latencies.

Additionally, the duration of each frame for both the client and server was recorded. fig. 5.16 shows the frame delta time in milliseconds as a function of the number of networked physics-simulated objects on the server.

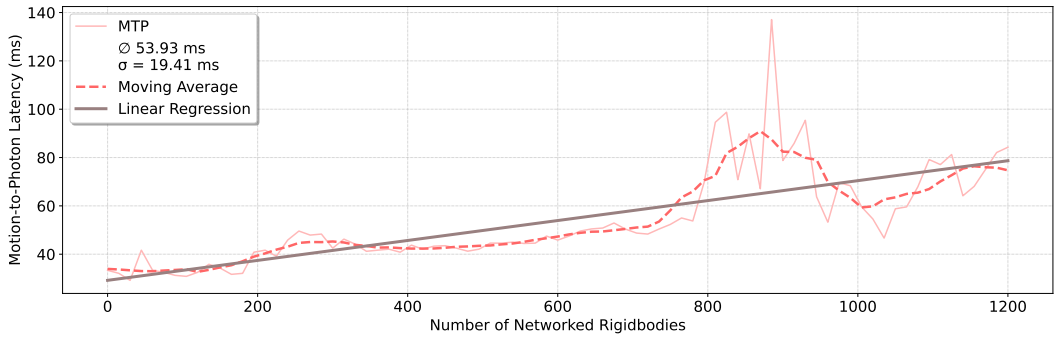



FIGURE 5.14: Average MTP Latency of a Moving Networked Physics-Simulated Objekt under Increasing Server Load

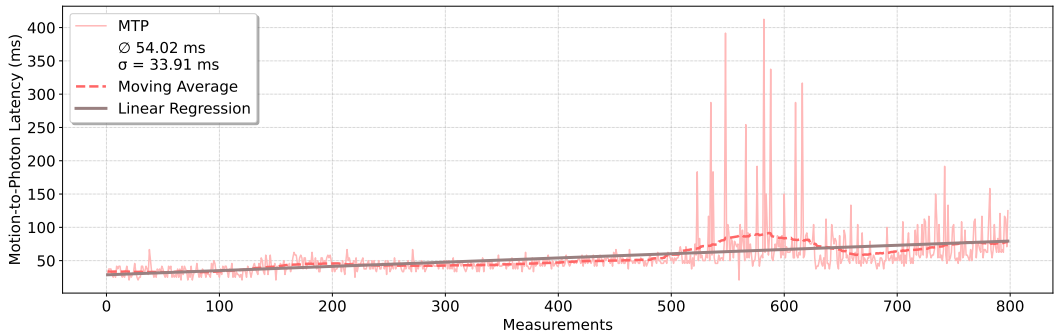



FIGURE 5.15: MTP Latency of a Moving Networked Physics-Simulated Objekt under Increasing Server Load

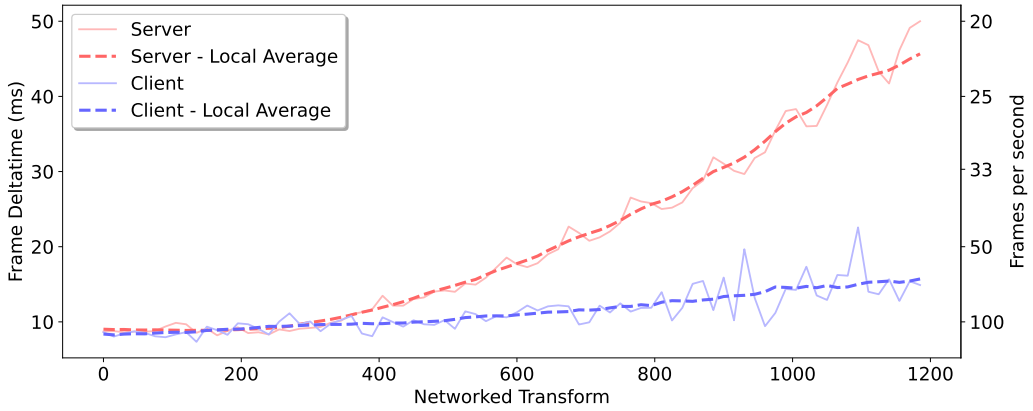



FIGURE 5.16: Frame Delta Time of Client and Server under Increasing Server Load

Analysis of both sets of MTP Latencies reveals that as the number of simulated objects increases, so does the MTP latency. The MTP latency grows irregularly, whereas the frame time increases non-linearly, with the server frame time (red) outpacing that of the client (blue), particularly after 300 objects. Up to 300 objects, the frame times for both server and client are approximately similar, with only minor variance.

The inverse of the frame time represents the SUR and CUR. As previously determined in section 5.2.2 both significantly influence the MTP latency. Therefore, any impact from network traffic remains indiscernible. These measurements function more as a performance benchmark for the server, as discussed earlier. Notably, even with 1000 networked transforms, no significant increase in latency was observed, suggesting that the increase in MTP latency stems primarily from the server's computational load due to the real-time physics simulation.

A discrete physics simulation timestep of 10 ms was used in these measurements, chosen in relation to the tick rate of 90 Hz. Examining fig. 5.14, a peak between 750 and 1000 objects interrupts the otherwise linear increase in MTP latency. This phenomenon can be attributed to spikes observed in fig. 5.15, which represent sudden, significant variations in MTP latency. These spikes begin around the 520th measurement, increasing in frequency until around measurement 620, after which they flatten but persist until the end of the test. This indicates an increased instability as the number of simulated objects grows. The frame time of the server in fig. 5.16 also exhibits wave-like fluctuations, which are mirrored in the MTP latency readings between measurements 620 and 798, suggesting that frame time, rather than network traffic, is the primary factor influencing MTP latency.

As noted in section 5.2.6, the issue of a 4.17 ms recording frequency persists, though its impact is reduced in this context. A linear regression (represented in brown/grey) was applied to the averaged and raw MTP latencies, visualizing the trend of increasing latency. This outcome was expected and visually confirmed, with the server's frame time increase also aligning with predictions. While the linear regression aligns closely with the raw MTP data, the averaged MTP values are skewed higher due to the smoothing of spikes.

As outlined in the methodology, occlusion culling of cameras was employed to exclude rendering overhead. The increase in client frame time remains inconclusive, possibly resulting from either the processing of objects or the overhead from Network Game Object (NGO) data processing. Previous findings suggest that network packet processing is not a contributing factor. As the number of networked physics

objects increases, the update rate drops from 120 Hz to around 72 Hz even before reaching 1000 objects.

In conclusion, increasing the number of networked physics-simulated objects leads to a reduction in SUR, CUR, and overall synchronization stability. No significant changes were observed up to 150 objects. However, with approximately 200 objects, latency issues begin to arise, and beyond 400 objects, these issues grow substantially, with a notable drop in CUR that users may perceive negatively.

Overall, the real-time physics simulation primarily places a computational burden on the server, with a lesser impact on the client. No direct evidence was found linking network traffic to processing delays in this scenario.

## 5.3 Leveraging NGO with ECS for Networked Simulated Physics XR Interactions

As discussed in the theoretical background, there is currently no integration of the provided SDKs with the ECS package and consequently with NCE. An attempt to implement the hybrid approach, where ECS is used solely for networked simulated physics, has proven to be challenging. As expected, the initial implementation effort is substantial and presents several issues.

A major challenge arises from the need for most of the environment to run on the ECS side, given the use of two separate physics systems. As a result, all objects and environments require duplication on the ECS side. Additionally, the Universal Render Pipeline (URP) must be employed to render the entities using the "Graphics for Entities" package.

A simple server-authoritative example, involving the picking up and throwing of a box, was used to explore this. Initially, the client is not the owner of the box, which is a `GameObject` converted into an `Entity` at the start of the scene through a "baking" process. This involves using authoring components, which are `MonoBehaviours` designed to assign the appropriate ECS components to a newly created `Entity` during baking.[46]

To interact with the object, a hand must be represented as a `MonoBehaviour` within ECS to trigger collisions and enable grabbing the box.

Another issue is that every server tick requires updating the position of all `Entities` via a snapshot from the server. This demands a custom implementation within the NGO framework, which must efficiently manage transformation data to ensure accurate synchronization.

Development of this hybrid solution was halted due to its complexity and time requirements. Theoretically, upon collision, ownership of the networked object could be requested via an RPC, and an `Entity` would need to check ownership of its corresponding GameObject every frame. If ownership is granted, the movement of the `Entity`-hand could directly affect the object.

Although this approach could leverage ECS with DOTS for server-side performance gains, the feasibility of enabling physics-based interactions between object-oriented `GameObjects` and data-oriented entities was not demonstrated. This suggests that such an implementation is likely complex and may require custom solutions built on Unity's Transport system. Further investigation is needed to optimize this approach.

[46] https://docs.unity3d.com/Packages/com.unity.entities@1.0/manual/baking.html (retrieved 06.09.2024, 18:12)

# Chapter 6

# Analysis and Interpretation

The present thesis aimed to evaluate the networked physics capabilities of Unity XR projects, specifically examining the suitability of NGO and NCE for implementing physics-based interactions. The investigation encompassed both theoretical analysis and technical measurements of bandwidth and latency, crucial factors in the performance of networked XR environments. It also explored the feasibility of integrating the ECS with NGO to potentially enhance the performance of physics simulations. The findings of this thesis contribute to a deeper understanding of the strengths and limitations of Unity's networking solutions in the context of XR development, and offer insights for optimizing networked physics-based interactions in immersive applications.

The scarcity of existing literature directly comparing the performance of Unity's networking libraries in the context of XR applications necessitates that the subsequent discussion primarily relies on inferences drawn from the observed measurements and the official Unity documentation, rather than extensive comparisons with established research findings.

### Theoretical Findings

The theoretical findings of this study underscore the complexities and challenges associated with developing XR applications incorporating networked physics. The juxtaposition of NGO and NCE highlights the distinct trade-offs between these two architectures. NGO, with its object-oriented structure, provides an intuitive and easily accessible development environment but exhibits weaknesses in terms of bandwidth efficiency and scalability, particularly in large-scale simulations involving numerous objects and users. In contrast, NCE, characterized by its data-oriented architecture, enables high performance and scalability, making it particularly attractive for XR applications where physics-based interactions are paramount. However, the integration of existing XR SDKs with NCE poses a significant challenge, as these SDKs are typically `MonoBehaviour`-based and not directly compatible with the ECS architecture of NCE. The proposed hybrid solution, combining NGO and ECS, offers a promising approach to leveraging the strengths of both architectures. Nevertheless, implementing such a hybrid system necessitates advanced techniques and considerable development effort.

The theoretical analysis elucidates that the choice between NGO and NCE hinges on the specific requirements of the XR application. For smaller projects or prototypes, where ease of use is a priority, NGO may be a suitable choice. However, for large, complex simulations with many users and objects, particularly those emphasizing physics-based interactions, NCE offers clear advantages in terms of performance and scalability. The hybrid solution represents a compromise that combines

the strengths of both architectures, albeit at the cost of increased implementation complexity.

Overall, the theoretical investigation emphasizes the necessity of carefully considering the requirements and constraints when selecting a network architecture for XR applications with networked physics. It also demonstrates that, despite advancements in Unity's networking solutions, there remain open challenges, particularly concerning the integration of existing XR SDKs and the optimization of bandwidth efficiency.

### Comparative Analysis of Wi-Fi 6e and Wi-Fi 5 Results

Section 5.1 presented a comparative analysis of Wi-Fi 6e and Wi-Fi 5 performance, focusing on network latency and bandwidth - critical factors for XR applications. The experimental setup involved router replacements and measurements using tools like `hrPing`, `iPerf3`, and `Ping Tools`.

Access point measurements revealed broader 2.4 GHz network utilization and concentrated 5 GHz usage, with many access points on Channel 100, possibly a default setting in some routers. The router used in all measurements showed a strong signal and no interference on Channel 36, although this was only assessed initially.

RTT measurements were inconsistent with high fluctuations, and unexpectedly, the Netgear router exhibited higher latency than the TP-Link. Latency measurements from the server to the HMD were unsuccessful, likely due to external factors or the Meta Quest 3's potential difficulty in handling ICMP packets.

In downlink speed measurements, the Netgear router nearly reached the Meta Quest 3's theoretical maximum of 2400 Mbit/s, while the TP-Link fell short at 250 Mbit/s. Both used two data streams. The TP-Link's underperformance could be attributed to hardware, configuration, or software.

Uplink speeds were significantly below expectations for both routers, with the Netgear achieving roughly double the throughput of the TP-Link, mirroring the theoretical maximum doubling from 1200 Mbit/s (5 GHz) to 2400 Mbit/s (6 GHz). Both achieved roughly a third of their maximum throughput, suggesting a bias by configuration issues, though the source (HMD or server) remains unclear.

Overall, the data indicates no significant latency difference, but a substantial bandwidth advantage for Wi-Fi 6e over Wi-Fi 6, highlighting the hardware benefits of the newer standard.

### Timing of Client-Side Snapshot Processing

The findings in section 5.2.1 underscore the complexities of snapshot processing in a networked environment where server tick rates exceed client update rates. The observation that clients process only the latest snapshot values, even if multiple snapshots arrive within a single frame, highlights the potential for data inconsistencies and desynchronization. This behavior could lead to visual artifacts or unpredictable behavior in physics-based interactions, particularly when the time difference between server ticks and client updates is significant. The implication is that careful consideration must be given to aligning server tick rates with client capabilities to ensure smooth and consistent data synchronization.

The study also reveals the advantage of RPCs in terms of their update frequency, being independent of server ticks. This characteristic makes RPCs well-suited for real-time communication, where events need to be processed immediately without waiting for the next snapshot synchronization cycle. In the context of networked

physics simulations, RPCs could be leveraged for critical interactions that demand instantaneous feedback, such as triggering immediate reactions to collisions or user inputs.

The use of `NetworkManager.ServerTime` emphasizes the importance of managing time discrepancies between clients and servers in networked environments. Network latency inevitably introduces delays in data transmission, which can lead to inconsistencies in the perceived game state across different clients. By utilizing techniques like `NetworkManager.ServerTime`, developers can mitigate the impact of latency and ensure a more synchronized and coherent experience for users, particularly in scenarios involving real-time physics interactions where even minor timing differences can be disruptive.

The visualization shown in fig. 5.7 of the `NetworkVariable` synchronization process further illustrates the intricacies of networked communication. The inherent delays and buffering involved in data transmission, even under ideal conditions, can lead to a discrepancy between the availability of updated values and their actual visual representation. This buffering effect underlines the need for careful timing and synchronization strategies in networked physics simulations, where real-time responsiveness is crucial for maintaining immersion and interactivity.

#### RTT of RPCs, Network Variables and Unity Transport

The analysis of RTT in section 5.2.2 reveals several key insights into the dynamics of networked communication within NGO. The measurements demonstrate the significant influence of both CUR and SUR on latency.

A higher CUR generally leads to reduced RTT, facilitating more frequent and timely processing of `RPCs` and `NetworkVariables`. However, the marginal gains observed between higher CUR values suggest a point of diminishing returns, beyond which further increases in CUR may not yield substantial latency improvements.

The impact of SUR on RTT is particularly pronounced in the context of the synchronization of `NetworkVariables`. Lower SUR values translate to increased RTTs, as the server requires more time to process and propagate changes. The data emphasizes the importance of aligning the SUR with or exceeding the network tick rate to avoid delays arising from timing mismatches. The substantial reduction in RTT observed when SUR surpasses the tick rate underscores this point, highlighting the critical role of server-side processing efficiency in minimizing latency.

The observed dependence of `NetworkVariables`' RTT on the tick rate aligns with the expectation that they are synchronized per snapshot, as outlined in the NGO documentation. The measurements confirm that the frequency of server snapshots, which is directly tied to the tick rate, influences the latency of NetworkVariable updates. In contrast, the minimal sensitivity of `RPCs` to the tick rate is consistent with their message-based nature, as they are processed independently of the snapshot synchronization cycle. These findings validate the documented synchronization mechanisms in NGO and highlight the importance of selecting the appropriate communication method based on the specific needs of the application.

The Unity Transport Layer proves to be a dependable infrastructure for networked communication, exhibiting consistently low RTTs across a range of network tick rates. This stability underscores its design for maintaining predictable latency, rendering it invaluable for time-sensitive applications where consistent performance is paramount. The observed plateau effect at lower SURs suggests potential buffering or batching mechanisms within the transport layer to ensure smooth data transmission even under less favorable server conditions.

The fluctuation observed in all measured RTT values is noteworthy, primarily attributed to the interplay of varying frequencies and, in the case of `NetworkVariables`, the additional buffering mechanism. However, considering the high fluctuations in network latency observed in section 5.1, it is reasonable to assume that these network inconsistencies contribute significantly to the RTT variations. In network scenarios with more stable latencies, the measured RTTs would likely exhibit less fluctuation, resulting in flatter visualizations.

Overall, the measurements highlight the intricate interplay between client and server update rates, network tick rate, and the choice of synchronization mechanisms in determining RTT. The results emphasize the importance of optimizing both client and server performance to achieve the lowest possible latency in real-time networked applications. While the Unity Transport Layer provides a robust and stable foundation, careful consideration must be given to the selection and configuration of `RPCs` and `NetworkVariables` to ensure optimal performance and responsiveness in the context of networked physics simulations and other time-sensitive interactions within XR environments.

### Network Traffic and Packet Analysis

The analysis of network traffic and bandwidth calculations in section 5.2.3 and section 5.2.3 provides valuable insights into the data transmission characteristics of NGO and their implications for networked XR applications. The findings highlight both the strengths and limitations of NGO's communication mechanisms, particularly in the context of bandwidth usage and scalability.

**RPCs and NetworkVariables**
The examination of `RPC` and `NetworkVariable` packet sizes reveals that Unity introduces a substantial overhead, particularly for individual data types. The inherent metadata overhead of `RPCs`, encompassing object, component, and method identifiers alongside transmitted parameters, can lead to suboptimal bandwidth utilization, particularly in scenarios involving the synchronization of numerous small data elements. The message-based nature of `RPCs`, which precludes the efficient packing and collective transmission characteristic of snapshot-based NetworkVariables, further exacerbates this bandwidth inefficiency.

The lack of detailed documentation regarding the composition of network packets and Unity's optimization strategies further complicates efforts to fine-tune bandwidth usage. The observed 8-byte packing of payloads, although potentially beneficial for larger data transfers, might introduce unnecessary overhead for frequent transmissions of smaller data types. The additional object overhead incurred by `RPCs`, while providing context for remote function calls, further contributes to bandwidth consumption. The discovery of undocumented periodic ping messages between server and client, while ensuring connectivity, adds to the overall network traffic, emphasizing the need for careful consideration of background communication in bandwidth-constrained environments.

**Network Transforms**
The analysis of `NetworkTransform` traffic demonstrates the trade-offs between data accuracy and bandwidth efficiency. Synchronizing individual transform properties results in disproportionately high bandwidth usage due to the fixed overhead associated with each packet. However, synchronizing multiple properties improves bandwidth utilization, as the data payload increases relative to the overhead. The observed adherence to the MTU and the resulting packet fragmentation for larger

data payloads further emphasize the need for careful management of network traffic to avoid unnecessary overhead and potential latency issues. The investigation into half-precision floating-point numbers demonstrates a potential avenue for bandwidth reduction, albeit at the cost of reduced accuracy. The significant overhead introduced by Unity's underlying system, even with optimized data payloads, highlights the need for further optimizations within the NGO framework itself.

**Bandwidth Calculations and Scenarios**

The bandwidth calculations and scenarios presented in section 5.2.4 offer a glimpse into the scalability challenges of NGO in networked XR applications. The idealized derivation of bandwidth requirements, while providing a theoretical framework, underscores the difficulty of accurately predicting real-world bandwidth demands in complex, multi-user scenarios. The exponential increase in bandwidth with the number of users and synchronized objects highlights the potential limitations of NGO for large-scale XR environments, especially when considering the theoretical throughput constraints of even state-of-the-art HMDs like the Meta Quest 3. The scenarios presented, ranging from small-scale office collaborations to large virtual concerts, illustrate how quickly NGO can reach its limits in terms of bandwidth capacity. While NGO might be suitable for applications with limited user interactions and object synchronization, its scalability for massive, physics-driven XR experiences remains a concern.

In conclusion, the findings in section 5.2.3 and section 5.2.4 provide a nuanced understanding of NGO's network traffic characteristics and bandwidth demands. The analysis reveals both strengths, such as the flexibility of `RPCs` and the improved bandwidth efficiency of synchronizing multiple transform properties, and limitations, including significant overhead, lack of documentation, and scalability concerns. These insights are crucial for developers navigating the complexities of networked physics simulations in XR, guiding informed decisions about optimization strategies, hardware choices, and the potential need for alternative networking solutions or architectures.

**End-to-End Latency of Ownership Acquisition**

The analysis of end-to-end latency for ownership acquisition in section 5.2.5 offers practical insights into the performance characteristics of a prevalent use case in server-authoritative networked applications, especially those involving physics-based interactions within XR environments. The measurements highlight the impact of server update rates on the latency associated with transferring object ownership, a crucial step for enabling user interactions in such environments.

The experiment's focus on varying server fixed timesteps and server update rates aimed to identify potential bottlenecks in the ownership acquisition process. The negligible difference in latency observed between fixed timesteps of 5 and 10 seconds suggests that the server's fixed update rate has minimal impact on the overall latency in this scenario. This can be attributed to the fact that the ownership request is made via an RPC, which operates independently of the fixed update rate. The fixed update rate does not dictate the frame time for the entire game loop, and RPCs are processed within the frame update cycle. This demonstrates that the RTT of an `RPC` is not influenced by the server's fixed update rate.

The substantial reduction in latency with increasing server update rates underlines the importance of server-side responsiveness for minimizing delays in ownership acquisition. The findings suggest that higher server update rates enable more frequent processing of ownership requests, leading to faster ownership transfers

and, consequently, reduced latency for user interactions. This observation aligns with the general principle that increasing the frequency of server updates improves the responsiveness of networked applications.

Comparing the measured end-to-end latency of ownership acquisition (approximately 45.71 ms at 120 Hz SUR) to the `RPC` RTT under similar conditions (45.19 ms) reveals a marginal difference. This suggests that the overhead associated with the ownership transfer process itself is relatively small compared to the inherent latency of `RPC` communication. This observation highlights the efficiency of NGO's ownership management mechanisms, even in scenarios where real-time responsiveness is crucial.

The unexpected latency spikes observed in the measurements warrant further investigation. While potential causes such as Unity Editor lags or network fluctuations were considered, the precise origin remains unclear. Understanding these sporadic delays is essential for ensuring consistent performance in real-world XR applications, where even brief interruptions in interactivity can disrupt the user experience.

Overall, the findings in section 5.2.5 emphasize the importance of optimizing server update rates to minimize latency in ownership acquisition scenarios. The relatively low overhead associated with the ownership transfer process itself suggests that NGO's mechanisms are well-suited for handling object interactions in networked XR environments. However, further research is needed to address the observed latency spikes and ensure consistently smooth and responsive user experiences in real-world applications.

#### MTP Latency of a Moving Networked Physics-Simulated Objekt

The analysis of MTP latency in section 5.2.6 and section 5.2.7 offers key insights into how well NGO handles real-time physics simulations over a network, specifically focusing on the synchronization of `NetworkTransforms` attached to `NetworkRigidbodies` and the effects of server load. The use of video analysis, while offering a visual representation of synchronization behavior, introduces inherent limitations due to its reliance on frame-based discretization. This methodology, though intuitive, may not fully capture the subtle timing nuances crucial for understanding the underlying network dynamics. The results highlight several key findings regarding MTP latency in networked physics-simulated environments.

**Single Network Rigidbody**

The analysis of MTP latency for a single networked Rigidbody in section 5.2.6 reveals an average latency of 35.36 to 39.53 ms with the mentioned hardware and measurement setup. Diese Range kommt aufgrund discretization effects in the measurement method zustande und basiert hier auf dessen kleinstem messintervall von approximately 4.17 ms (section 4.2.2). These resulting MTP latencies are not within an acceptable range of 15 to 30 ms for interactions in XR. [15] Allerdings lässt vermuten hier mit der richtigen Netzwerkonfiguration und Aufbau eine MTP Latency von kleiner gleich 30 ms erreicht werden. Falls dies nicht möglich sein sollte, können techniques wie interpolation und client side prediction höhere Latenzen, zum Preis von CPU time ausgleichen.

However, the measurements exhibit fluctuations and an slight upward trend, suggesting potential performance variability that could impact user experience. The observed latency spikes, with one reaching nearly 75 ms, contribute to the overall increased standard deviation. The discrepancy between the average MTP latency and the theoretically achievable minimum latency further underscores the presence of

unaccounted-for factors influencing the measurements. The video-based measurement methodology, while providing a visual representation, might not fully capture subtle timing nuances, potentially impacting the accuracy of the analysis.

**Increasing Server Load**

The investigation into the impact of increasing server load on Motion-to-Photon (MTP) latency, as illustrated in Figures fig. 5.14 and fig. 5.15, reveals a clear correlation between the number of networked physics-simulated objects and the observed latency. As the number of simulated objects increased, both the MTP latency and frame times exhibited a non-linear growth pattern. The irregular increase in MTP latency, coupled with the disproportionately escalating server-side frame times, particularly beyond 300 objects, strongly suggests that the server's computational burden is the primary bottleneck in maintaining low MTP latency. The degradation in server performance, as evidenced by the rising frame times, directly translates to increased MTP latency, highlighting the critical role of server-side processing efficiency in maintaining low latency in networked physics simulations. However, without direct network traffic measurements, it cannot definitively rule out the possibility that network congestion also contributed to the observed latency increases, especially at very high object counts. However, by utilizing eq. (5.2), the throughput can be estimated. With 1200 `NetworkTransforms` synchronizing only position and rotation, the Data Payload $S_{DP}$ amounts to 28.80 kilobytes. At a tick rate of 90 Hz, this translates to a theoretical throughput of 3.68 Mbps or 29.47792 Mbit/s. Comparing this to the available bandwidth, network congestion can be ruled out as a contributing factor.

The observation that server performance consistently declined with an increasing number of networked Rigidbodies, coupled with the controlled environment of the experiment, further strengthens the direct relationship between server frame time and MTP latency. The controlled environment minimizes the influence of external factors like network fluctuations, thus emphasizing the impact of server-side computational load on the observed latency increases. The degradation in server performance, as evidenced by the rising frame times, directly translates to increased MTP latency, highlighting the critical role of server-side processing efficiency in maintaining low latency in networked physics simulations. The emergence of latency spikes and increased instability as the number of simulated objects grows further highlights the scalability challenges associated with networked physics simulations in NGO. The significant drop in client update rate, falling to around 72 Hz even before reaching 1000 objects, indicates that the client's performance is also impacted, potentially leading to a degraded user experience. These findings suggest that real-time physics simulations impose a substantial computational load on the server as the number of objects scales, whereas the client experiences a comparatively moderate impact. The peak observed between 750 and 1000 objects further accentuates a threshold where performance instability becomes pronounced, with spikes in MTP latency correlating with fluctuations in server frame time.

Moreover, the reduction in synchronization rates (SUR and CUR) as the object count increased—especially beyond 400 objects—highlights a critical limitation. While no significant performance degradation was observed with up to 150 objects, the results suggest that beyond 200 objects, noticeable latency issues begin to emerge, and synchronization stability deteriorates further as the object count increases. This degradation in performance underscores the scalability challenges associated with networked physics simulations in large-scale environments, even with optimized networking libraries like NGO.

Overall, the MTP latency analysis reveals a complex interplay between server load, synchronization rates, and the inherent limitations of the measurement methodology. The findings suggest that while NGO can handle moderate loads efficiently, scalability remains a challenge for real-time networked physics simulations in large-scale environments. The computational burden on the server, more so than network traffic, is the limiting factor in maintaining low MTP latency, and strategies to reduce server load or optimize frame processing are crucial for maintaining performance as object counts rise.

#### Leveraging NGO with ECS for Networked Simulated Physics XR Interactions

The exploration of a hybrid approach, combining the traditional GameObject-centric NGO with the performance-oriented ECS, in section 5.3, encountered significant challenges. The fundamental incompatibility between `MonoBehaviours`, integral to most XR SDKs, and the ECS paradigm necessitates substantial restructuring to integrate existing XR features into an ECS-based project. The requirement for environment duplication and the need for custom snapshot management further amplify the complexity of this hybrid implementation. The development of this approach was ultimately halted due to these complexities and the time constraints of the project. The findings suggest that while a hybrid NGO-ECS solution may theoretically offer benefits, its practical realization demands extensive development effort and may not be feasible for many XR projects, especially those already reliant on `MonoBehaviour`-based XR SDKs. The investigation underscores the need for further research and development to bridge the gap between these two architectures and enable a more seamless integration of ECS's performance advantages into existing XR workflows.

## Limitations

The present thesis, while providing valuable insights into the networked physics capabilities of Unity XR projects, acknowledges certain limitations that may influence the interpretation and generalizability of the findings.

Firstly, the scope of the research was constrained by time limitations, which prevented a more extensive and in-depth exploration of certain aspects. The focus on understanding the basics of NGO and its application to three specific use cases limited the ability to conduct comprehensive testing and evaluation across a wider range of scenarios and configurations. The pre-compiled nature of the physics engine in Unity further restricted the ability to delve into its internal workings and explore potential optimizations at a lower level.

Secondly, the measurements and experiments were conducted in a controlled laboratory environment, which may not fully reflect the complexities and challenges of real-world XR deployments. Factors such as network congestion, interference from other devices, and variations in hardware configurations could impact the performance and behavior of networked physics simulations in real-world scenarios. The limited sample size and the specific hardware and software configurations used in the study might also affect the generalizability of the findings to other contexts.

Thirdly, the measurement techniques employed, while carefully designed and validated, have inherent limitations. The MTP latency measurements, for instance, rely on color changes on displays and are subject to the limitations of the camera's frame rate and potential errors due to external factors. The accuracy and precision of these measurements could be further improved with more sophisticated equipment and techniques.

Finally, the hybrid approach combining NGO with ECS, while theoretically promising, was not fully implemented and evaluated due to its complexity and time constraints. The feasibility and effectiveness of this approach in practical XR applications with networked physics interactions remain to be explored in future research.

Despite these limitations, the thesis provides a foundation for understanding the networked physics capabilities of Unity XR projects and offers valuable insights for developers. The identified limitations underscore the need for future research in developing ECS-compatible XR SDKs, refining hybrid NGO-ECS approaches, and establishing more robust measurement techniques. These advancements would contribute to a deeper understanding of the complex interplay between networking and physics within the realm of XR, facilitating the development of increasingly immersive and interactive experiences.

# Chapter 7

# Conclusion

This thesis explored the potential of networked physics capabilities within Unity XR projects by evaluating the suitability of NGO and NCE regarding implementing physics-based interactions. Initially, a theoretical comparison of NGO and NCE was conducted, which informed subsequent measurements of bandwidth and latency for key components, starting with NGO. The results demonstrated that physics simulations utilizing the ECS are, on average, ten times more performant than traditional `MonoBehaviour`-based implementations. Additionally, data-oriented NCE proved to be more bandwidth-efficient compared to NGO. However, both NGO and NCE lack support for client-side prediction, and there are no existing XR SDKs for ECS and thus NCE. It is noteworthy that ECS can be integrated with NGO, although this integration is complex and resource-intensive.

A technical analysis of NGO, along with attempts at a hybrid implementation, revealed that NGO imposes significant computational overhead and throughput demands when managing numerous networked objects. Despite these challenges, NGO was capable of maintaining smooth and stable latency for up to 100 networked physics-simulated objects on the hardware used in this study. While NGO and ECS can theoretically operate together, achieving practical integration requires sophisticated techniques and considerable implementation effort.

These findings lead to several conclusions:

1. NGO is suitable for implementing networked physics-based interactions in XR applications but is not ideal for fundamental networked physics simulations.
2. Offloading physics computations to the server is a viable approach that can significantly alleviate the computational burden on mobile HMDs
3. Existing SDKs are `MonoBehaviour`-based, making pure use of NCE for networked physics interactions impractical without extensive porting to ECS.
4. Integrating NGO with ECS is possible but demands advanced techniques and significant development time to realize practical benefits.

This research identifies a clear gap in the availability of ECS-compatible SDKs, despite the potential for significant performance gains. It offers a detailed overview of both Unity networking libraries within the context of XR projects.Additionally, it facilitates the evaluation of the NGO's bandwidth requirements and illustrates its latency as the number of networked simulated physics objects increases under comparable hardware conditions. Finally, the study provides a foundation for comparing other networking libraries with regard to networked physics-based interactions in XR environments.

The implications of these findings are twofold. First, they inform the development of networked physics-based interactions in XR using NGO, emphasizing the need to address computational and bandwidth constraints effectively. Second, they

serve as a motivation for developing an ECS-native XR SDK within Unity, which could harness the performance advantages of ECS and facilitate more efficient networked physics simulations in XR applications.

## Improvements and Future Work

The present thesis demonstrates that NGO is suited for implementing networked physics-based interactions in XR environments, while NCE is unsuitable due to a lack of compatible SDKs. NGO's networked physics capabilities for Unity XR projects quickly reach their limits, whereas NCE could theoretically offer ten times the performance.

Future research should focus on several areas to enhance and refine these findings. First, measurements of end-to-end latency should be conducted with careful consideration of potential influences from the measurement setup, both before and after testing, to identify and mitigate fluctuations and disturbances. Second, a more detailed investigation into the impact of NGO on server performance is necessary. Third, the expansion capabilities provided by NGO should be examined to assess their effectiveness. Additionally, exploring Unity Transport for potential latency reduction in physics-based interactions could be beneficial. Finally, implementing and evaluating a hybrid approach that integrates NGO with ECS may provide insights into its usability for networked physics-based interactions.

While this bachelor thesis provides valuable insights into the utilization of NGO and the potential of ECS, several questions remain for future research:

- Can Unity Transport be utilized to optimize throughput and improve latency in NGO?
- How can client-side prediction and lag compensation be implemented in NGO and what is their effectiveness?
- How viable is a hybrid approach combining NGO with ECS for networked physics-based interactions in XR?
- How does Havok compare to the Unity Physics Package in terms of performance and suitability?
- What is the effort involved in developing a data-oriented, component-based XR SDK compatible with Unity's DOTS stack?

By addressing these questions and continuing to explore the potential of both NGO and ECS, developers can create more immersive, interactive, and realistic XR experiences that push the boundaries of what is possible in networked physics simulations.

# Appendix A

# RTTs of RPCs, NetworkVariables and the Unity Transport under Varying Conditions

The diagrams in this appendix present RTT measurements for RPCs, Network Variables, and the Unity Transport, each with variations of a fixed baseline consisting of the server tick rate, server update rate, and client update rate. Each implementation of the RTT is represented by a separate section. Within each section, there are three diagrams. The first diagram shows the RTT under varying network tick rates of 30, 60, 90, and 120 Hz. The second diagram shows the RTT under varying server update rates of 30, 60, 90, and 120 Hz. The third diagram presents the RTT under varying client update rates, which are aligned with the refresh rates of Meta Quest 3 at 72, 80, 90, and 120 Hz.

## A.1 RTT of a RPC under Varying Conditions

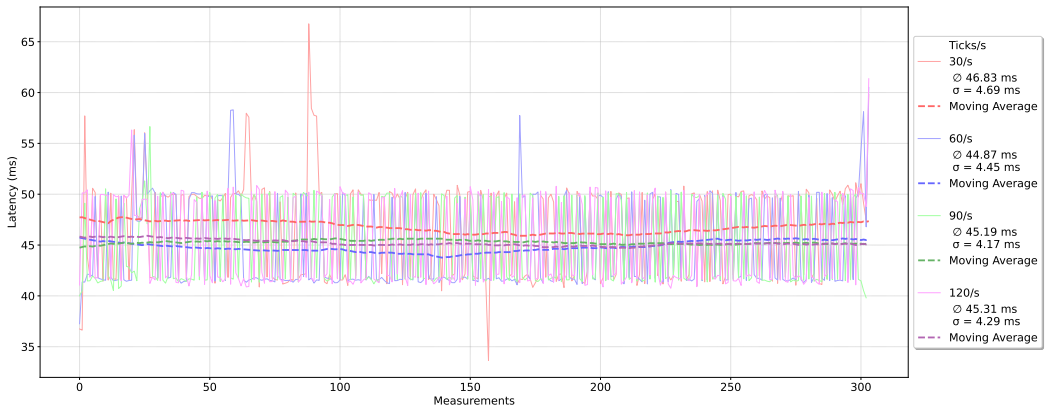



FIGURE A.1: RTT of a RPC under Varying Network Tick Rates

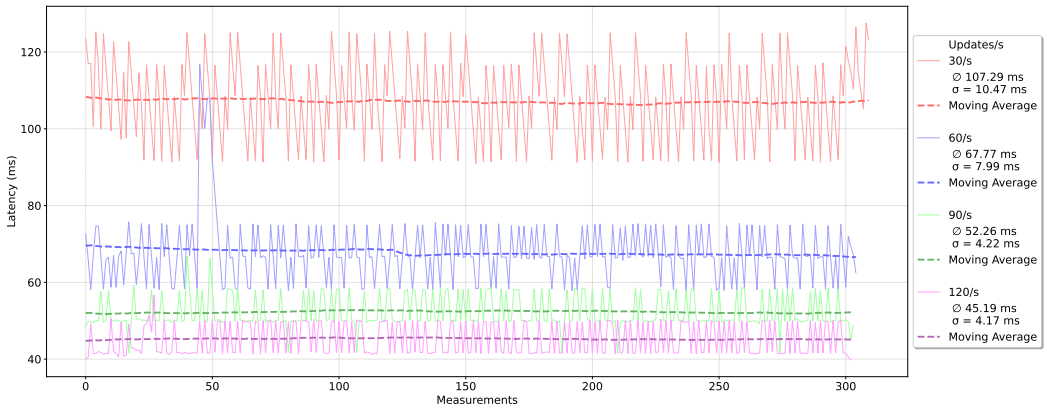



FIGURE A.2: RTT of a RPC under Varying Server Update Rates

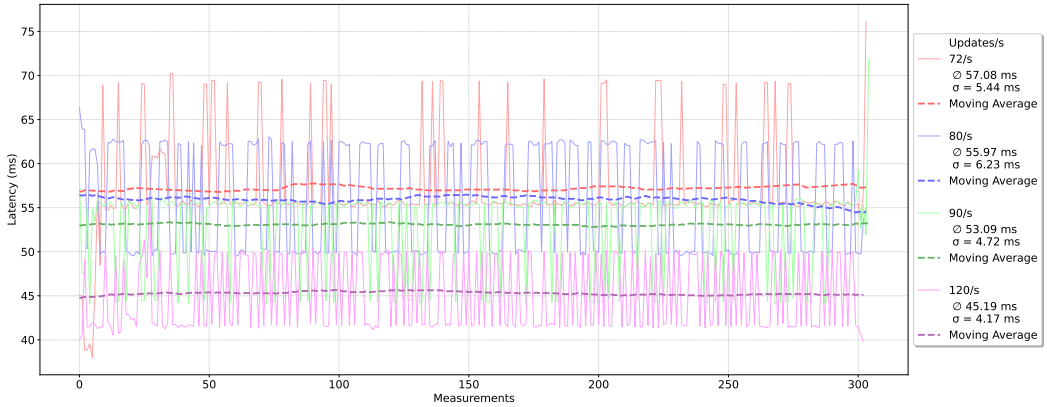



FIGURE A.3: RTT of a RPC under Varying Client Update Rates

## A.2 RTT of a NetworkVariable under Varying Conditions

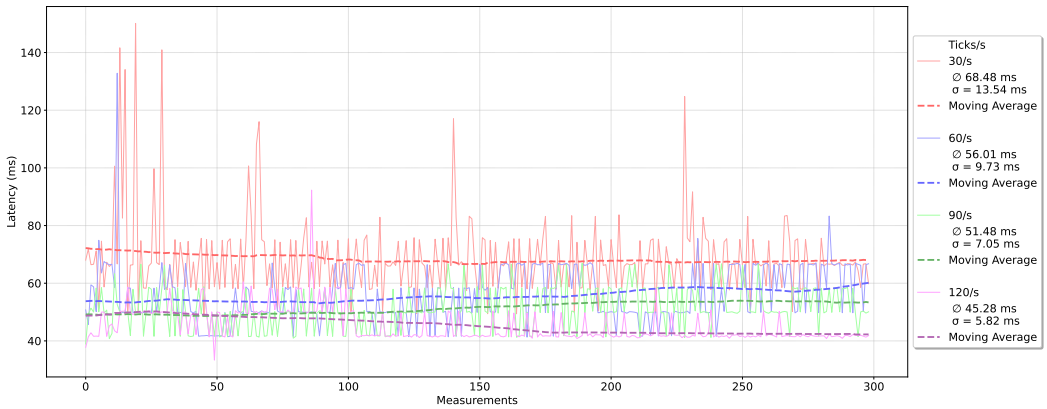



FIGURE A.4: RTT of a NetworkVariable under Varying Network Tick Rates

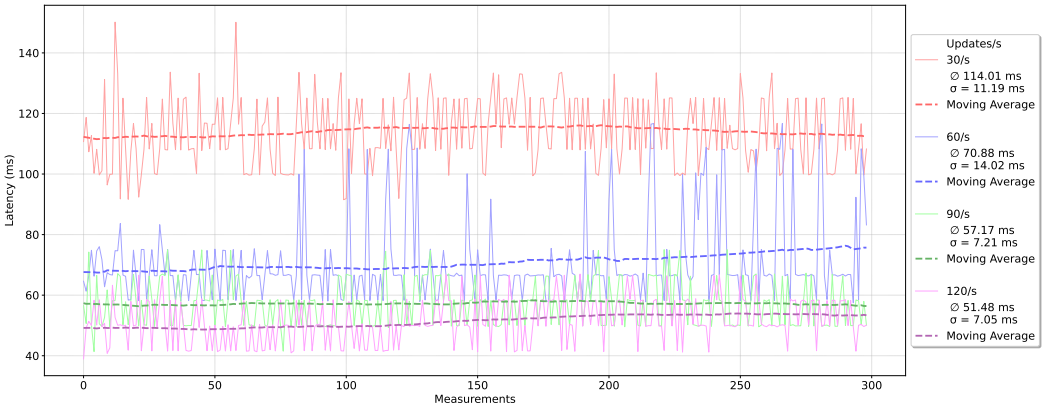



FIGURE A.5: RTT of a NetworkVariable under Varying Server Update Rates

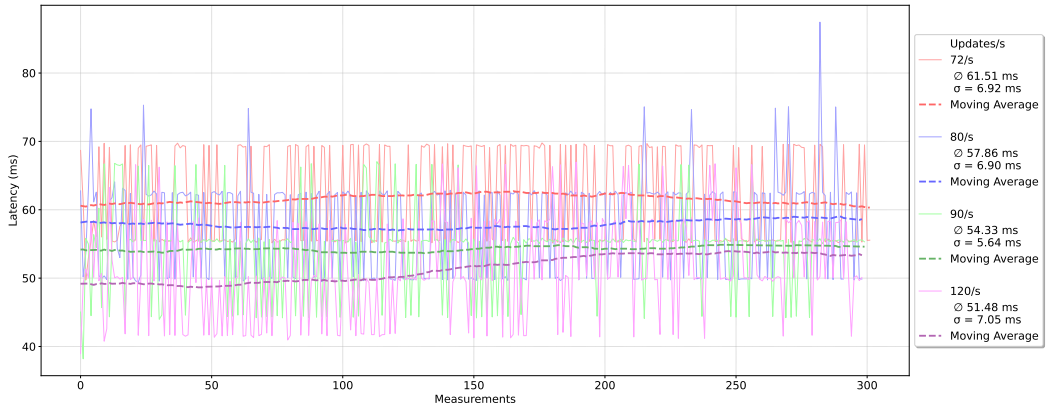



FIGURE A.6: RTT of a NetworkVariable under Varying Client Update Rates

## A.3 RTT of a the Unity Transport under Varying Conditions

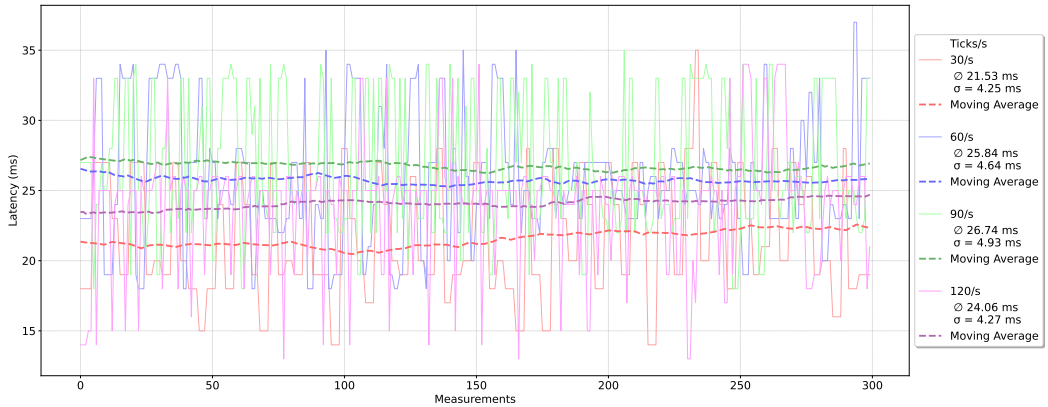



FIGURE A.7: RTT of a the Unity Transport under Varying Network Tick Rates

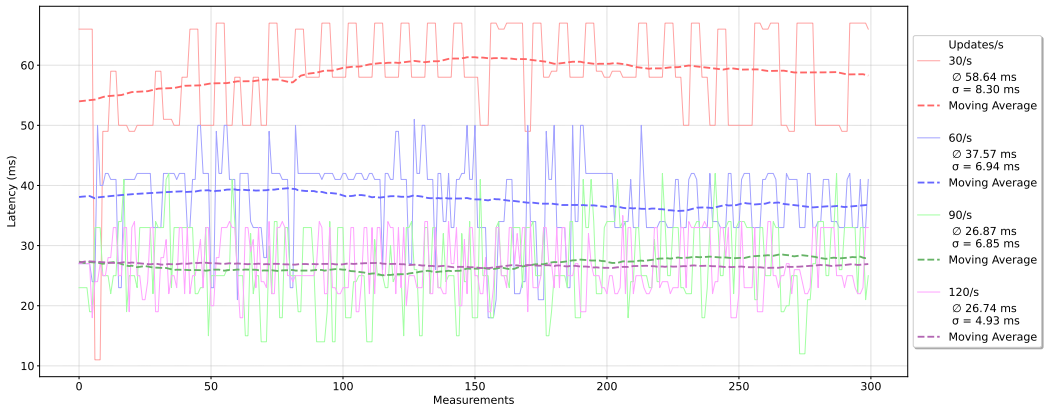



FIGURE A.8: RTT of a the Unity Transport under Varying Server Update Rates

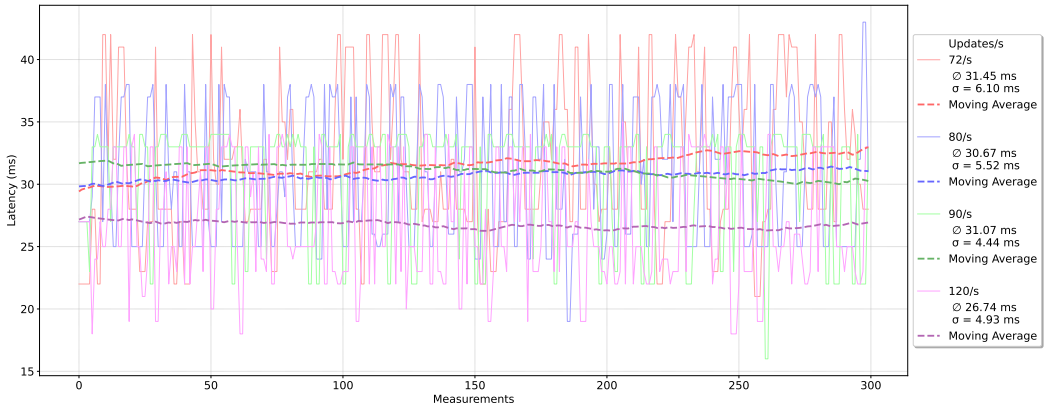


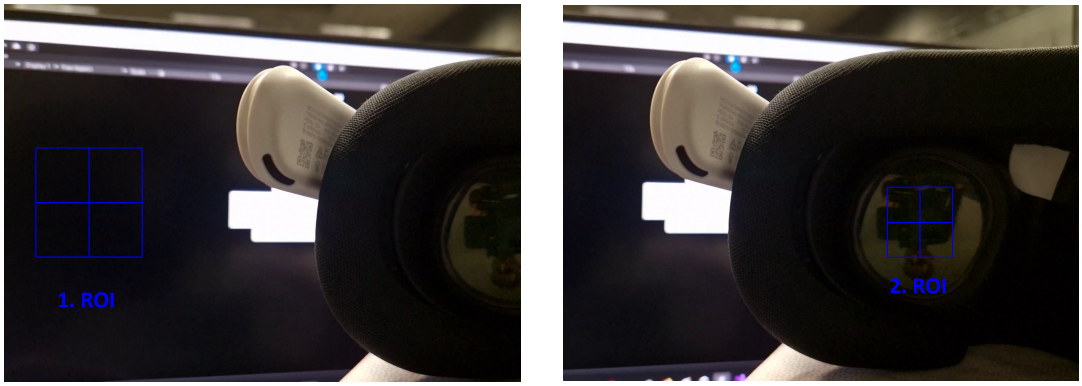

FIGURE A.9: RTT of a the Unity Transport under Varying Client Update Rates

# Appendix B

# Visualizations of MTP Latencies of a Networked Simulated Physics Object

The diagrams in this appendix visualize the MTP latency of a networked simulated physics object, as previously described in the methodology. This object utilizes a Network Rigidbody and a Network Transform component. The presented diagrams illustrate specific steps of the video data processing to enhance the understanding of the motion-to-photon latency calculation. Since the methodology remains consistent between the two measurement scenarios (with and without additional server load), no distinction is made here. The diagrams are arranged according to the data processing sequence. This begins with the selection of ROIs, as shown in Figure chapter B. Selection is performed by manually drawing a blue rectangular region with the mouse to define the ROI. For the measurement method, the selected region is reduced to a single pixel, specifically the central pixel where the two blue lines intersect.

End-to-end latencies were measured from a 240 FPS video within a system configured with a 90 Hz server network tick rate and 120 Hz server and client update rates.




(A) Selecting the First ROI (B) Selecting the Second ROI

FIGURE B.1: Demonstration of Selecting a Region of Interest

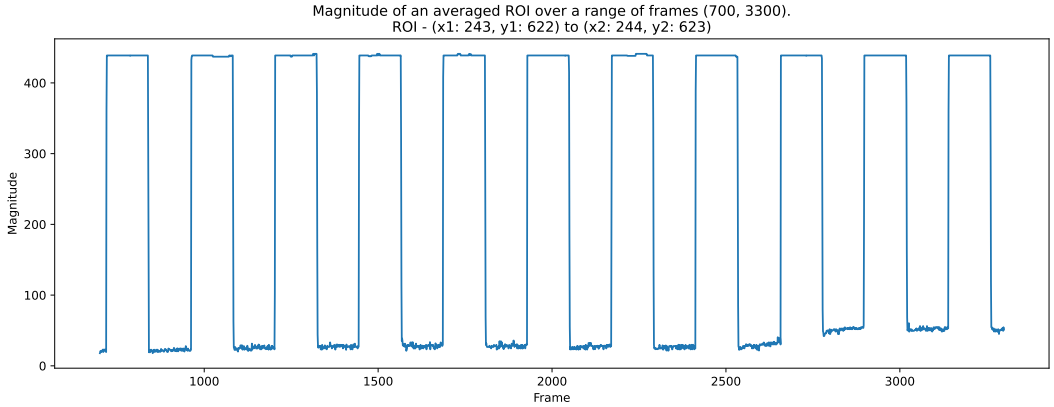



FIGURE B.2: Pixel Color Magnitudes of the First ROI (Server) over Frames

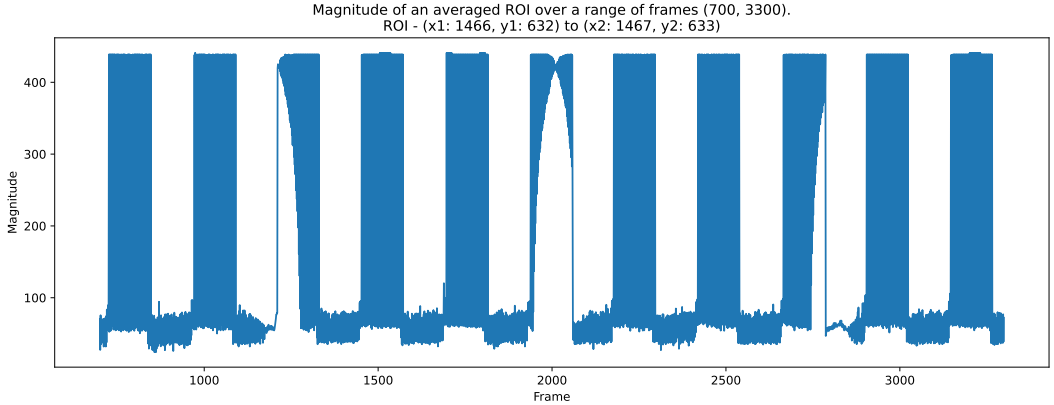



FIGURE B.3: Pixel Color Magnitudes of the Second ROI (HMD) over Frames

The increases in Figure fig. B.3 appear to be solid. However, upon closer inspection in Figure fig. B.4, it becomes evident that a black frame is inserted between each frame.

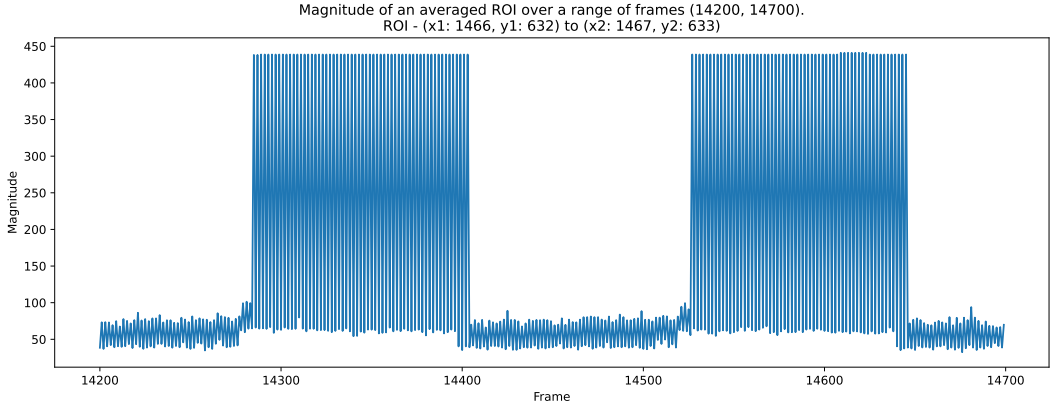



FIGURE B.4: Demonstration of the "Black Frame Insertion" of the Meta Quest 3

This technique is known as "Black Frame Insertion" (BFI). By intentionally inserting black frames between regular frames, blurring is reduced and the perception

of motion is enhanced. Furthermore, BFI can contribute to reducing motion sickness due to increased visual clarity. Thsi yields, that the Meta Quest 3 is using such technique.

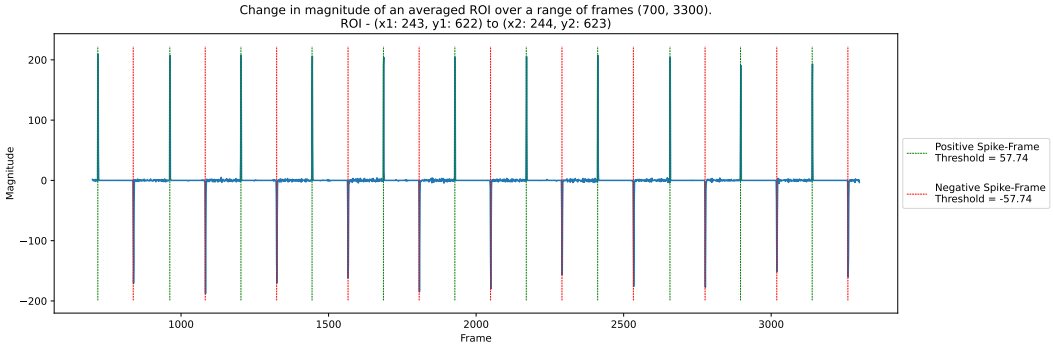



FIGURE B.5: Change in Pixel Color Magnitudes over Frames of the First ROI (Server)

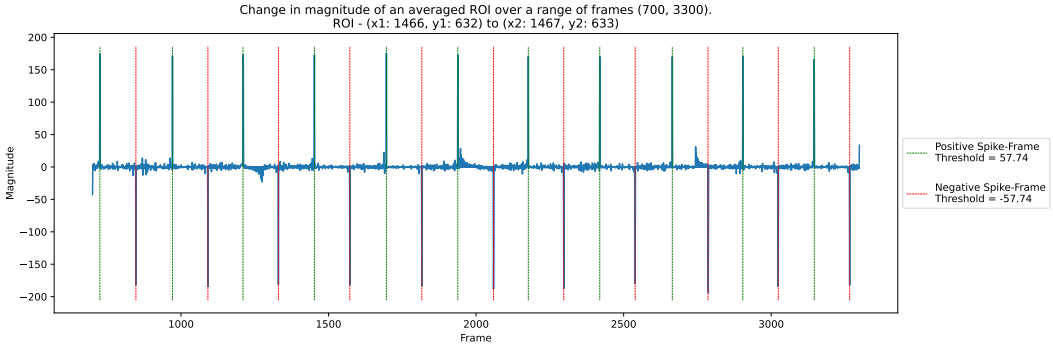



FIGURE B.6: Change in Pixel Color Magnitudes over Frames of the Second ROI (HMD)

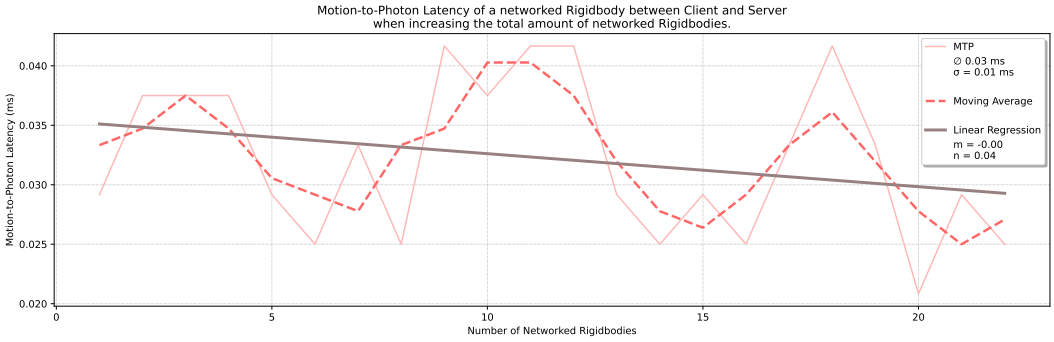



FIGURE B.7: Resulting End-to-End Latency between ROI 1 (Server) and ROI 2 (HMD)

# Appendix C

# Table of Network Packet Sizes

In table C.1, the columns are organized according to the unmanaged types of the respective transmission method, with the exception of an empty RPC sent to the server and another sent to all clients. Each row is divided into the sizes measured and calculated in the Unity Profiler (highlighted with a green header) and those measured and calculated in Wireshark. The terms "Objects," "Messages" correspond to the module names in the Unity Profiler: "NGO Objects" and "NGO Messages," with the total number of bytes sent also listed under "Overall".

The last column in the Unity Profiler section indicates the difference between the message size (i.e., the actual payload) and the overall sent size. In the Wireshark section (with a blue header), the final two columns present the payload size and the total packet size, as measured by Wireshark. The "UDP Overhead" column highlights the general overhead associated with UDP, which was used to determine Unity's overhead per packet. The last two rows display the network packet to synchronize the time with the server when sent and received.

The **header** consists of fixed and optional fields.

The **payload** is the data that is carried on behalf of an application

| | | UNITY PROFILER | | | | | Wireshark | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| **Communication** | **ValueType** | **Object** | **Message** | **Overall** | **DIFFERENCE** Message - Object | **DIFFERENCE** Overall - Message | **Unity** Overhead | **UDP** Overhead | **Payload** | **Packetsize** |
| **RPC** | (Empty) ToServer | 10 | 10 | 32 | 0 | 22 | 31 | 32 | 63 | 95 |
| | (Empty) ToEveryone | 10 | 25 | 48 | 15 | 23 | 31 | 32 | 79 | 111 |
| | byte | 10 | 10 | 32 | 0 | 22 | 31 | 32 | 63 | 95 |
| | short | 13 | 13 | 32 | 0 | 19 | 31 | 32 | 63 | 95 |
| | int | 14 | 14 | 32 | 0 | 18 | 31 | 32 | 63 | 95 |
| | long | 18 | 18 | 40 | 0 | 22 | 31 | 32 | 71 | 103 |
| | float | 14 | 14 | 32 | 0 | 18 | 31 | 32 | 63 | 95 |
| | double | 17 | 17 | 40 | 0 | 23 | 31 | 32 | 71 | 103 |
| | decimal | 26 | 26 | 48 | 0 | 22 | 31 | 32 | 79 | 111 |
| | char | 12 | 12 | 32 | 0 | 20 | 31 | 32 | 63 | 95 |
| | FixedString512 | 0.5 KB | 0.5 KB | . | . | . | . | 32 | 575 | 607 |
| **Network Variable** | byte | 2 | 14 | 32 | 12 | 18 | 31 | 32 | 63 | 95 |
| | short | 4 | 16 | 32 | 12 | 16 | 31 | 32 | 63 | 95 |
| | int | 6 | 18 | 40 | 12 | 22 | 31 | 32 | 71 | 103 |
| | long | 10 | 22 | 40 | 12 | 18 | 31 | 32 | 71 | 103 |
| | float | 5 | 17 | 40 | 12 | 23 | 31 | 32 | 71 | 103 |
| | double | 9 | 21 | 40 | 12 | 19 | 31 | 32 | 71 | 103 |
| | decimal | 17 | 29 | 48 | 12 | 19 | 31 | 32 | 79 | 111 |
| | char | 3 | 15 | 32 | 12 | 17 | 31 | 32 | 63 | 95 |
| | FixedString512 | 0.5 KB | 0.5 KB | . | . | . | . | 32 | 575 | 607 |
| **System Time Synchronization** | Send | | 4 | | | 24 | 8 | 32 | 41 | 73 |
| | Receive | | 11 | | | 24 | 8 | 32 | 41 | 73 |

TABLE C.1: Overview of Network Packet Sizes of Unmanaged Types by RPC and Network Variable

# Bibliography


[1] 5G Initiative Team. "NGMN 5G White Paper (1.0)". In: (Mar. 3, 2015). URL: https://www.ngmn.org/publications/ngmn-5g-white-paper.html (visited on 03/19/2024).

[2] Christoph Anthes et al. "State of the art of virtual reality technology". In: *2016 IEEE Aerospace Conference*. 2016 IEEE Aerospace Conference. Mar. 2016, pp. 1–19. DOI: 10.1109/AERO.2016.7500674. URL: https://ieeexplore.ieee.org/abstract/document/7500674 (visited on 03/07/2024).

[3] Christan Versteeg. *An analytical performance comparison between DOTS and Monobehaviours, ease of use & utilization – Gameplay Engineering*. URL: https://summit-2324-sem1.game-lab.nl/2023/09/25/me-and-the-bois-dots-ting-around/ (visited on 09/07/2024).

[4] Giovanna Culot et al. "Behind the definition of Industry 4.0: Analysis and open questions". In: *International Journal of Production Economics* 226 (Aug. 1, 2020), p. 107617. ISSN: 0925-5273. DOI: 10.1016/j.ijpe.2020.107617. URL: https://www.sciencedirect.com/science/article/pii/S0925527320300050 (visited on 03/11/2024).

[5] Cedric Westphal Huawei. "Challenges in Networking to Support Augmented Reality and Virtual Reality". In: 2016. URL: https://www.semanticscholar.org/paper/Challenges-in-Networking-to-Support-Augmented-and-Huawei/725e69bd0093e76505161b8dc6d68ad0a50c47da (visited on 03/07/2024).

[6] "IEEE Standard for Ethernet". In: *IEEE Std 802.3-2018 (Revision of IEEE Std 802.3-2015)* (Aug. 2018). Conference Name: IEEE Std 802.3-2018 (Revision of IEEE Std 802.3-2015), pp. 1–5600. DOI: 10.1109/IEEESTD.2018.8457469. URL: https://ieeexplore.ieee.org/document/8457469 (visited on 08/27/2024).

[7] George Kokiadis et al. *Decoupled Edge Physics algorithms for collaborative XR simulations*. July 17, 2024. DOI: 10.48550/arXiv.2407.12486. arXiv: 2407.12486[cs]. URL: http://arxiv.org/abs/2407.12486 (visited on 09/02/2024).

[8] V. Krstić and I. Mekterović. "Unity as a Physics Simulator: Calculating Mean Free Path for Hard Disk Gas". In: *2021 44th International Convention on Information, Communication and Electronic Technology (MIPRO)*. 2021 44th International Convention on Information, Communication and Electronic Technology (MIPRO). ISSN: 2623-8764. Sept. 2021, pp. 613–616. DOI: 10.23919/MIPRO52101.2021.9596893. URL: https://ieeexplore.ieee.org/abstract/document/9596893/authors#authors (visited on 09/02/2024).

[9] Yushin Lee, Donggun Park, and Yong Min Kim. "The effect of wearing a head-mounted display on the boundaries of the cervical range of motion based on perceived comfort in a static posture". In: *Virtual Reality* 27.2 (June 1, 2023), pp. 815–828. ISSN: 1434-9957. DOI: 10.1007/s10055-022-00684-w. URL: https://doi.org/10.1007/s10055-022-00684-w (visited on 09/06/2024).

[10] Michael Macedonia et al. “Exploiting reality with multicast groups: a network architecturefor large-scale virtual environments”. In: Apr. 11, 1995, pp. 2–10. ISBN: 978-0-8186-7084-8. DOI: 10.1109/VRAIS.1995.512473.

[11] Tariq Masood and Johannes Egger. “Augmented reality in support of Industry 4.0—Implementation challenges and success factors”. In: *Robotics and Computer-Integrated Manufacturing* 58 (Aug. 1, 2019), pp. 181–195. ISSN: 0736-5845. DOI: 10.1016/j.rcim.2019.02.003. URL: https://www.sciencedirect.com/science/article/pii/S0736584518304101 (visited on 03/11/2024).

[12] Prof. Matthew McGinity. “Lecture: "Foundations of Virtual Reality"”. TU Dresden. Winter semester 2022-23.

[13] Niels Christian Nilsson, Rolf Nordahl, and Stefania Serafin. “Immersion Revisited: A review of existing definitions of immersion and their relation to different theories of presence”. In: *Human Technology* 12.2 (Nov. 30, 2016), pp. 108–134. ISSN: 17956889. DOI: 10.17011/ht/urn.201611174652. URL: http://humantechnology.jyu.fi/archive/vol-12/issue-2/immersion-revisited (visited on 03/11/2024).

[14] Jason Orlosky, Kiyoshi Kiyokawa, and Haruo Takemura. “Virtual and Augmented Reality on the 5G Highway”. In: *Journal of Information Processing* 25 (2017), pp. 133–141. DOI: 10.2197/ipsjjip.25.133.

[15] Christer Qvarfordt, Henrik Lundqvist, and Georgios P. Koudouridis. “High Quality Mobile XR: Requirements and Feasibility”. In: *2018 IEEE 23rd International Workshop on Computer Aided Modeling and Design of Communication Links and Networks (CAMAD)*. 2018 IEEE 23rd International Workshop on Computer Aided Modeling and Design of Communication Links and Networks (CAMAD). ISSN: 2378-4873. Sept. 2018, pp. 1–6. DOI: 10.1109/CAMAD.2018.8514957. URL: https://ieeexplore.ieee.org/abstract/document/8514957 (visited on 09/08/2024).

[16] John-Ross Rizzo et al. “The Intersection between Ocular and Manual Motor Control: Eye–Hand Coordination in Acquired Brain Injury”. In: *Frontiers in Neurology* 8 (June 1, 2017). Publisher: Frontiers. ISSN: 1664-2295. DOI: 10.3389/fneur.2017.00227. URL: https://www.frontiersin.org/journals/neurology/articles/10.3389/fneur.2017.00227/full (visited on 08/31/2024).

[17] Jinjia Ruan and Dongliang Xie. “Networked VR: State of the Art, Solutions, and Challenges”. In: *Electronics* 10.2 (Jan. 2021). Number: 2 Publisher: Multidisciplinary Digital Publishing Institute, p. 166. ISSN: 2079-9292. DOI: 10.3390/electronics10020166. URL: https://www.mdpi.com/2079-9292/10/2/166 (visited on 03/09/2024).

[18] Ayoung Suh and Jane Prophet. “The state of immersive technology research: A literature analysis”. In: *Computers in Human Behavior* 86 (Sept. 1, 2018), pp. 77–90. ISSN: 0747-5632. DOI: 10.1016/j.chb.2018.04.019. URL: https://www.sciencedirect.com/science/article/pii/S0747563218301857 (visited on 08/31/2024).

[19] Nak-Jun Sung et al. “Real-Time Augmented Reality Physics Simulator for Education”. In: *Applied Sciences* 9.19 (Jan. 2019). Number: 19 Publisher: Multidisciplinary Digital Publishing Institute, p. 4019. ISSN: 2076-3417. DOI: 10.3390/app9194019. URL: https://www.mdpi.com/2076-3417/9/19/4019 (visited on 09/02/2024).

[20] Elliott Wen et al. "VR.net: A Real-world Dataset for Virtual Reality Motion Sickness Research". In: *IEEE Transactions on Visualization and Computer Graphics* (2024). Conference Name: IEEE Transactions on Visualization and Computer Graphics, pp. 1–7. ISSN: 1941-0506. DOI: 10.1109/TVCG.2024.3372044. URL: https://ieeexplore.ieee.org/document/10458372 (visited on 03/19/2024).

[21] *Wi-Fi 6 and Wi-Fi 6E drive global market opportunities | Wi-Fi Alliance*. URL: https://www.wi-fi.org/news-events/newsroom/wi-fi-6-and-wi-fi-6e-drive-global-market-opportunities (visited on 08/27/2024).

[22] *Wi-Fi Alliance® brings Wi-Fi 6 into 6 GHz | Wi-Fi Alliance*. URL: https://www.wi-fi.org/news-events/newsroom/wi-fi-alliance-brings-wi-fi-6-into-6-ghz (visited on 08/27/2024).

[23] *Wi-Fi CERTIFIED 6 | Wi-Fi Alliance*. URL: https://www.wi-fi.org/discover-wi-fi/wi-fi-certified-6 (visited on 08/27/2024).