EdgeVerse: Multi-User Virtual Reality via Edge Computing and eBPF

Okwudilichukwu Okafor
Computer Science
Saint Louis University, USA

Flavio Esposito
Computer Science
Saint Louis University, USA

Tommaso Pecorella
Information Engineering
University of Florence, Italy

***Abstract*— The adoption of Extended Reality (XR) technology has been hindered by the need for high-bandwidth and low-latency networks to provide immersive experiences. Head-mounted devices used in XR are still heavy and not portable, limiting the potential of XR applications in various contexts. In this paper, we propose *EdgeVerse*, a multi-user XR system that leverages edge computing and extended Berkeley Packet Filter (eBPF) to offload computation, thereby reducing the dependency on high-bandwidth networks in support of XR applications. Our approach enables lightweight clients, such as devices with an edge browser, making XR more accessible to users. The key design of *EdgeVerse* focuses on offloading the connection and network synchronization of multiple XR users at the edge. We used an XDP bidirectional router that processes XR traffic faster to enhance user interaction, responsiveness, and immersive experience. To establish the practicality of our approach, we evaluate our results on a prototype that indicates improved response times and reduced latency with respect to baseline solutions.**

***Index Terms*— Virtual Reality, Augmented Reality, network management, edge Computing, eBPF, eXpress Data Path.**

I. Introduction

XR (Extended Reality) is an umbrella term that encompasses Augmented Reality (AR), Virtual Reality (VR), and Mixed Reality (MR). It represents all forms of immersive digital experiences that blend or replace real-world environments with virtual elements. XR systems have rapidly gained popularity across various industries, including commercial applications such as gaming, and academic sectors for training, e.g., for continuing medical education or aerospace engineering. McKinsey has estimated that the Metaverse will generate up to $5 trillion in value by 2030 [1]. A portion of the research community believes that they will enable immersive virtual experiences that will transform how we learn, communicate, and perceive reality. Their transformational experiences and potential to increase productivity has been well documented, see e.g., [2], [3]. Despite their widespread, some pressing challenges still exist today in XR systems, including mobility and portability of the head-mounted devices [4]. In particular, current (metaverse) technologies fall short of seamlessly connecting the physical and virtual worlds.

Experts emphasize that virtual reality (VR) systems must achieve at least 8K resolution and a 120 Hz frame rate to prevent pixelation and motion sickness. To achieve these goals, several solutions have been proposed to improve user experience and provide more mobility [4]–[6]. A practical strategy proposed is to push the rendering to a cloud or edge server. By rendering at a remote location, the graphical elements and visual components of a VR environment are generated in real-time at the remote server, and the VR video is transmitted to thin VR glasses or a display device of the end user. While this strategy promises portability and high mobility, VR rendering needs to be done at a high frame rate and with low latency to ensure a smooth experience as well as minimize the discomforts that could arise from inconsistencies or delays in the visual representation. Prior solutions [2], [5], [7] have also proposed ways to stream the entire or part of the application from the edge, while others [4], [8] proposed approaches to cache and reuse the VR application frames in a way that it minimizes the resources expended. Some other work [8], [9] considered a multi-user VR scenario and developed optimization strategies from a network perspective. While sound and addressing relevant problems, none of these formerly published approaches considered the synchronization of multiple VR users nor has any optimization been done to speed up VR packet processing at the network layer using eBPF. In addition, none of the work emphasized the need to have an open VR system that utilizes a non-propietary XR runtime. To tackle these challenges, we propose an architecture that decouples the components that constitute a multi-user VR environment (the application and network layer) and offloads key roles such as the multi-user networking service. We implemented an eBPF/XDP program within the intermediate network router that processes the VR packets faster to and from the VR users and the VR server. This approach enhances user experience, reducing latency and improving bandwidth.

In particular, in this paper we introduce *EdgeVerse*, a multi-user VR application that has the multi-user networking and synchronization offloaded to an edge server. Our design is based on a decoupling of the multi-user network operation (such as the connection processing and synchronization) from the scene rendering. *EdgeVerse* offloads the connection and synchronization tasks of the VR users to a separate dedicated system, a connection server, and consistently maintains the users' states across the local devices of the connected VR users. Such *EdgeVerse* connection server runs at the edge of the network and leverages the PUN SDK [10] to maintain user states and connection. *A key novelty of our architecture design is that it should allow VR users' devices to only process scene rendering while receiving the updated*

[1]Okwudilichukwu Okafor is a graduate student in the department of Computer Science, Saint Louis University, MO 1 N, 63103, USA.

[2]Flavio Esposito is an Associate Professor with the Department of Computer Science, Saint Louis University, USA.

[3]Tommaso Pecorella is an Associate Professor with the Department of Information Engineering, University of Florence, Italy.

978-3-903176-66-9 ©2024 IFIP

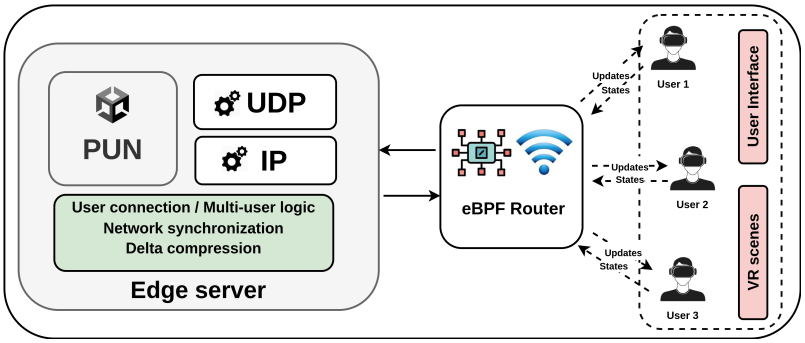


Fig. 1: High-level system architecture of *EdgeVerse*. VR clients connect through the PUN SDK) which runs at the edge.

positional states of other connected VR users that are in the same virtual space.

We implemented *EdgeVerse* and tested our application using two network servers for multi-user connection. One of the network servers runs as the edge server (*EdgeVerse*'s connection server) within the users' LAN, and the other runs as the cloud server in a public cloud. *EdgeVerse* is developed using C# programming language, Unity Engine, and scripting APIs, Photon Unity Networking APIs and on-premise server SDK. The XDP programs are written in C/C++ and attached to an intermediate eBPF router. Our evaluation of *EdgeVerse* showed good improvement in response time and higher frame rate compared to using the cloud server thereby enhancing the responsiveness of the VR application. Other evaluation results on bandwidth show an improved data speed to *EdgeVerse*'s VR users as the connection and network synchronization is done within the edge server. Our system architecture is separated into three different components as shown in Figure 1.

II. Design Challenges

To achieve our design goals, we tackled three primary challenges: synchronized user interaction, proprietary XR runtimes, and network optimization Using eBPF.

Synchronized User Interaction. In a multi-user VR environment, ensuring consistent updates for all users interacting with shared objects is crucial to maintaining immersion. We introduced a dedicated monitoring routine that continuously checks for any inconsistencies in the state of shared objects across users. This routine handles incoming updates from remote users and resolves discrepancies in real-time, guaranteeing that every user sees the same interactions. This persistent synchronization ensures a seamless experience, reducing the risk of misaligned views and fostering a cohesive virtual environment.

(2) Proprietary XR Runtimes. Most VR platforms rely on proprietary XR runtimes, limiting flexibility and compatibility, especially for projects requiring open standards for security and compliance, such as in government settings. To overcome these constraints, *EdgeVerse* utilizes the OpenXR runtime, which supports cross-platform, high-performance access to a broad range of XR devices. OpenXR allows *EdgeVerse* to run on multiple hardware configurations without being tied to a specific vendor, enhancing adaptability and ensuring that the application can be deployed in diverse environments without sacrificing compatibility or performance.

Network Optimization Using eBPF. Photon SDK's integration with Windows and the .NET Framework posed limitations in optimizing VR packet handling, impacting latency and throughput. To address this, we implemented a Linux-based eBPF router with custom XDP programs for packet processing. The eBPF router efficiently manages network traffic between VR clients and servers, reducing latency and increasing data throughput. By offloading packet processing to the network layer, *EdgeVerse* achieves better performance, enabling a responsive and scalable multi-user VR environment that can handle complex interactions without degradation in user experience.

III. EdgeVerse Architecture

User Interface. This layer consist of all the end user devices that are used to interact with a VR application. Most of the devices in this layer are headsets and controllers. However, with recent advancements, many VR applications can now be experienced using phones, tablets, directly on PCs and browsers using mouse and keyboard as controllers. For standalone untethered VR applications, the application is built to the headset and directly rendered. Whereas if the headset requires tethering, then the application can be rendered directly on a PC while the headset act as a display device.

Network Logic. This layer encodes the multi-user functionality and the network optimization strategy of *EdgeVerse*. This layer is completely decoupled from the user interface implementation and devices. In the event that multiple users are to participate in a shared environment, this layer handles the connection to the edge server for connectivity and synchronization among the users. The logic implements three algorithms: The first, checks the requested state of the application depending on the connection status of the user. The connection status returned indicates if the user device is allowed to connect to the network server or not based on the network configuration of the user and internet protocol (IP) address. The second algorithm offloads the multi-user network synchronization to the edge server. Offloading of the network synchronization to the edge server ensures that

Algorithm 1: Connection to the Edge Server

Data: The connection object to the server
Result: The connection status and network mode
1 $connectionStatus \leftarrow False$
2 **if** *connectionObject is True* **then**
3 $\quad connectionStatus \leftarrow True$
4 $\quad multiUserMode \leftarrow Activated$
5 $\quad networkUser \leftarrow UserSpawn()$
6 $\quad UserSync() \leftarrow connection.networkServer()$
7 $\quad Updates() \leftarrow consistentWithServer$
8 **else**
9 $\quad connectionToServer \leftarrow False$
10 $\quad singleUserMode \leftarrow Activated$
11 $\quad rendering.Set.Local() \leftarrow True$
12 **return** $connectionStatus, UserMode$

Algorithm 2: Network Synchronization

Require: Object position and rotation
Ensure: Updated object position and rotation
1: **if** isLocalUser **then**
2: {Client is the local user, send states}
3: RunPhysicsSimulation(PhysicsData)
4: UpdateObjectPositionAndRotation()
5: ComputePositionAndRotationDeltas()
6: SendDeltaUpdatesToServer()
7: **else**
8: {Client is not the local user, receives updates}
9: ReceiveDeltaUpdatesFromServer()
10: InterpolateObjectPositionAndRotation()
11: **end if**

Algorithm 3: VR Packet Processing

Data: Ingress packet
Result: Packet is properly directed
1 **for** *Every Ingress packet* **do**
2 $\quad pktProcessed \leftarrow False$
3 $\quad$ **if** pkt *is VR* **then**
4 $\quad\quad$ **if** $pkt.direction$ *is to_VR_server* **then**
5 $\quad\quad\quad XDP\_redirect\_to\_VR\_server\_IF()$
6 $\quad\quad$ **else**
7 $\quad\quad\quad XDP\_redirect\_to\_VR\_client\_IF()$
8 $\quad\quad pktProcessed \leftarrow True$
9 $\quad$ **else**
10 $\quad\quad pktProcessed \leftarrow$
$\quad\quad\quad Linux\_NW\_pkt\_proc\_stack()$
11 **return** $pktProcessed$

end user devices only focus on rendering local contents to the users connected while being constantly updated of the behavior of other network users in the same environment. The third algorithm implements the eBPF/XDP program that optimizes the VR packet processing to and from the VR users and the VR network server. The first two algorithms freed up resources that would have been required if the multi-user network functionality were directly handled by the end user devices. The third algorithm improve the VR packet processing times as it traverses the network.

Connection Algorithm. As shown in Figure 1, connections from multiple users are handled by the edge server, which manages user interactions in the shared VR environment. The connection algorithm (Algorithm 1) initializes *connectionStatus* as **False** and determines if the client is in single or multi-user mode. If a UDP connection is established (Line 2), *connectionStatus* is set to **True**, activating multi-user mode (Line 4), creating users as networked objects, and synchronizing their state with the server (Lines 5-6). The server continuously updates user states, isolating end-user devices to focus on local rendering.

If the connection is unsuccessful, as shown in Line 8, the connection status is set to **False** in Line 9 and the single-user mode is activated in Line 10. In the single-user mode, the rendering is set to local in Line 11, meaning that the client is only rendering what is on their machine. Finally the algorithm returns the connection status and the user mode.

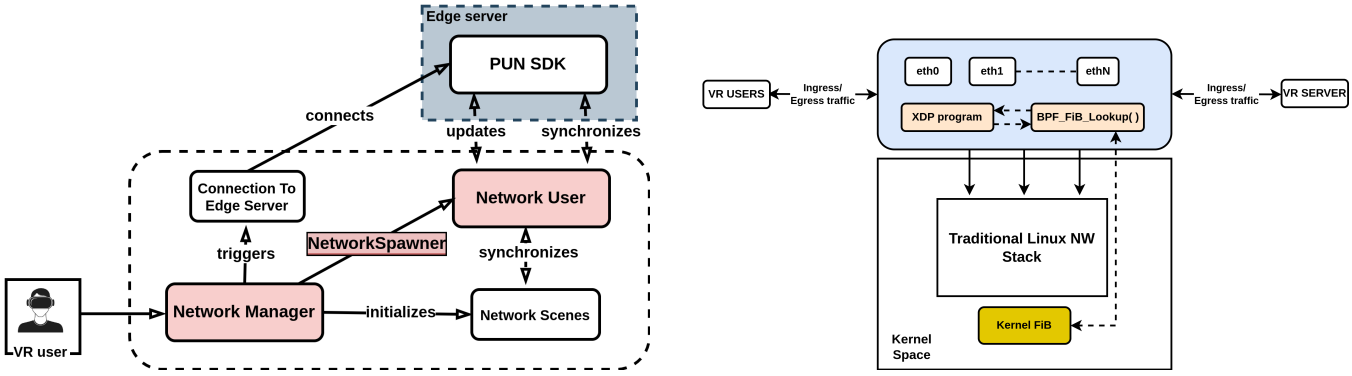
Network Synchronization Algorithm. The network synchronization algorithm ensures all connected clients have a consistent view by updating the state using delta values between the current and prior states. Local users compute position and rotation deltas, sending updates to the edge server. Remote users receive these updates and apply interpolation. This delta encoding minimizes bandwidth usage, reducing latency and maintaining consistency. The algorithm is represented as

$$P_i(t) = f(P_i(t-1), U_i(t), S(t)),$$

where $P_i(t)$ is the position of object i at time t, $U_i(t)$ is the user input, $S(t)$ is the network state, and f is the function computing the new position.

The equation models non-simultaneous user interactions, where the new position $P_i(t)$ of an object depends on its previous position $P_i(t-1)$, the user input $U_i(t)$ (e.g., movement/rotation commands), and the network state $S(t)$. The function f computes the new position using these inputs, ensuring synchronization across all connected clients.

VR Packet Processing Algorithm. The algorithm processes incoming packets at the eBPF router. If a packet is identified as a VR packet, it is redirected using either *XDP_redirect_to_VR_server_IF* or *XDP_redirect_to_VR_client_IF* based on its direction, and marked as processed. Non-VR packets are handled by *Linux_NW_pkt_proc_stack*. The algorithm then returns the processing status.



(a) *EdgeVerse* consistent state sharing and synchronization.

(b) *EdgeVerse* eBPF GW. packets are redirected at the interfaces.

Fig. 2: (a) The software architecture of *EdgeVerse*, showing consistent state sharing and synchronization. (b) The network architecture of *EdgeVerse*, where VR packets are redirected through the eBPF gateway at the network interfaces.

IV. System Implementation

In this section, we describe the design and implementation of *EdgeVerse*, developed using the Unity3D platform and Photon Unity Networking (PUN) SDK.

End-User devices used include HP Reverb, Samsung Odyssey HMDs, and Windows PCs with NVIDIA GeForce RTX GPUs. To avoid reliance on Oculus (Meta), we used OpenXR as the default runtime, allowing builds for Oculus Quest 2. A headless version was created using Unity XR simulator, enabling interaction via keyboard and mouse. The multi-user functionality is managed by two servers: a cloud server in Azure and an edge server in the user's LAN, both running PUN SDK. The edge server has 128GB RAM and an NVIDIA RTX GPU, while the cloud server has 16GB RAM. Rendering is performed on user devices.

EdgeVerse consists of three main software modules namely `NetworkManager`, `NetworkUser` and the `NetworkSpawner` as seen in Figure 2a. These software modules interact with each other to connect, instantiate and synchronize the networked VR users. User connections and spawning go through the `NetworkManager` module. On the successful connection to a network server, the `NetworkManager` places the connection on a multi-user mode interacts with the `NetworkSpawner` module to create the network user. The VR user is then placed in the shared VR scene where all connected users are synchronized. Other software modules used are Unity XR Interaction Toolkit [11] and several Unity Prefabs [12]. Unity XR Interaction Toolkit is an interaction solution for building VR and AR applications. It offers a framework that enables Unity input events to be used for 3D and UI interactions. The Interaction Manager, which connects these two types of components, and a set of basic Interactor and Interactable components make up the system's core. Additionally, it has elements that you can utilize to move about and create visuals.

V. Conclusion

In this paper, we discussed how to enable a responsive multi-user VR adoption with edge computing and eBPF. We developed a multi-user VR application, *EdgeVerse*, explained the system architecture and design implementation. We offloaded the multi-user logic to an edge sever that runs on PUN SDK with the VR traffic traversing through an eBPF router for optimized packet processing.

VI. Acknowledgement

This work has been supported by NSF Awards #1908574 and #2201536, and partially by the EU under the Italian National Recovery and Resilience Plan (NRRP) of NextGenerationEU, partnership on "Telecommunications of the Future" (PE0000001 - program "RESTART").

References

[1] McKinsey and Company, "Value creation in the metaverse," https://www.mckinsey.com/capabilities/growth-marketing-and-sales/our-insights/value-creation-in-the-metaverse, 2024.

[2] X. Hou, S. Dey, J. Zhang, and M. Budagavi, "Predictive adaptive streaming to enable mobile 360-degree and vr experiences," *IEEE Transactions on Multimedia*, vol. 23, pp. 716–731, 2021.

[3] A. Nichols. Top 5 tech trends for 2022: Virtual reality offers exciting new avenue. [Online]. Available: https://www.cepro.com/news/top-5-tech-trends-2022-virtual-reality-offers-exciting-new-avenue/

[4] X. Hou, Y. Lu, and S. Dey, "Wireless vr/ar with edge/cloud computing," in *2017 26th International Conference on Computer Communication and Networks (ICCCN)*. IEEE, 2017, pp. 1–8.

[5] X. Hou and S. Dey, "Motion prediction and pre-rendering at the edge to enable ultra-low latency mobile 6dof experiences," *IEEE Open Journal of the Communications Society*, vol. 1, pp. 1674–1690, 2020.

[6] S. Shi, V. Gupta, M. Hwang, and R. Jana, "Mobile vr on edge cloud: a latency-driven design," in *Proceedings of the 10th ACM multimedia systems conference*, 2019, pp. 222–231.

[7] X. Hou, S. Dey, J. Zhang, and M. Budagavi, "Predictive view generation to enable mobile 360-degree and vr experiences," in *Proceedings of the 2018 Morning Workshop on Virtual Reality and Augmented Reality Network*, 2018, pp. 20–26.

[8] Y. Li and W. Gao, "Muvr: Supporting multi-user mobile virtual reality with resource constrained edge cloud," in *2018 IEEE/ACM Symposium on Edge Computing (SEC)*. IEEE, 2018, pp. 1–16.

[9] S. N. Gunkel, "[dc] multi-user (social) virtual reality commnunication," in *2019 IEEE Conference on Virtual Reality and 3D User Interfaces (VR)*, 2019, pp. 1359–1360.

[10] Photon. Demos and turials. [Online]. Available: https://doc.photonengine.com/pun/current/getting-started/pun-intro

[11] Unity. (2023) Xr interaction toolkit. [Online]. Available: https://docs.unity3d.com/Packages/com.unity.xr.interaction.toolkit

[12] Unity3D. (2023) Prefabs. [Online]. Available: https://docs.unity3d.com/Manual/Prefabs.html