MRNetViz: Immersive Network Visualization and Monitoring with Mixed Reality

Authors: Sri Nithya Anne, Sneha Sugilal, Govind Iyer

GitHub Link: MR Net viz

1. Abstract

As the complexity and volume of internet traffic increase, real-time network visualization is more essential for network administrators and IoT-enabled environments. Traditional network monitoring tools often fail to provide spatial context for understanding activity. This project leverages Mixed Reality technologies and packet localization to develop a system that visualizes network traffic in real-time, with both network topology and dynamic packet exchanges displayed across physical spaces. We propose a basic model, MRVizNet, that uses Unity 6, an interactive game engine, with MQTT protocol, for real-time communication and Scapy, a packet sniffer. MRVizNet presents an interactive and dynamic, 3D visualization of network traffic that provides spatial context for data transfer, helping users analyze network activity intuitively.

2. Objectives

The MRVizNet system integrates various hardware and software components to capture, process, and visualize network traffic in real time. The objectives are followed by a review of the hardware and software tools utilized.

- Develop a real-time network traffic visualization system using XR.
- Integrate MQTT for real-time packet transmission.
- Provide intuitive tools for filtering, analyzing, and visualizing packet flow.
- Enhance user interaction with network data.

3. Existing Work and Their Limitations

In a similar paper *XRShark: Mixed Reality Network Introspection*, the authors Meghan Clark, Mark W. Newman, Prabal Dutta, discuss several existing tools and frameworks for network introspection, MR applications, and device localization. These tools include **X-Trace**, which captures relationships and data flow across multiple network layers, and **WiFröst**, which links code execution to network behavior in embedded systems. Additionally, **EyeSec** provides augmented reality-based network visualization for wireless sensor networks.

One limitation discussed in the paper is the **reliance on physical presence for visualization in augmented reality (e.g., EyeSec)**, which limits remote monitoring and requires the user to be physically near the network nodes.

In our project, we overcome this limitation by implementing a **virtual reality-based visualization system** via Unity 6 that allows users to monitor and interact with network traffic remotely. This enables a more flexible and accessible approach to understanding network activity without requiring physical proximity to the access points or network devices.

4. Requirements

To successfully implement the real-time network traffic visualization project, the following describes the tools, packages, and libraries needed for its implementation on a high-level:

Libraries required:

- **Python (Scapy, Paho-MQTT)**: Scapy is used to capture Wi-Fi packets, while Paho-MQTT sends the data to the MQTT broker in JSON format.
- **MQTTnet Library**: Enables Unity to subscribe to network data topics from the MQTT broker, allowing real-time packet visualization.
- **Newtonsoft.Json**: For parsing incoming JSON data representing network packets.
- **TextMeshPro**: For displaying detailed packet information in the UI.
- **XR Interaction Toolkit**: For enabling XR-based interactions (optional for VR headset support).
- **Meta XR Plugin**: For integrating the Meta Quest 3 headset (optional for immersive visualization).

Software tools required:

- **Packet Sniffers**: Devices such as Wi-Fi analyzers capture network traffic at the packet level, providing granular data on traffic flow.
- **Unity 6**: The primary development environment for building the visualization UI. It supports AR/VR integration, allowing for interactive and immersive visualization.
- **MQTT Broker (Mosquitto)**: Acts as an intermediary to route network data from Python scripts to Unity 6. This ensures low-latency data transmission for real-time visualization.

Hardware tools required:

- **Wireless Access Points**: Wi-Fi, ZigBee, and Bluetooth access points facilitate communication across different protocols, generating real-world network traffic data.
- **Quest 3 Headset**: For future scope, the integration of VR headsets for immersive network visualization will provide a 3D perspective to enhance the spatial interpretation of network traffic.

5. Implementation

This project consists of several core scripts working together to achieve real-time network traffic visualization. Let's dive into it on a low-level basis:

- The **MQTTReceiver.cs** script is responsible for connecting to the MQTT broker, subscribing to a specific topic, and listening for incoming packet data. Upon receiving a packet, it parses the data and triggers the visualization process.
- The **PacketVisualizer.cs** script handles creating packet instances, adjusting their size and color based on the packet type (TCP, UDP, HTTP, or FTP), and animating them toward the designated access points. It also draws lines to visualize the packet's path.
- The **PacketInfo.cs** script stores essential details like packet type, size, source IP, and destination IP, allowing this information to be displayed when the user interacts with a packet.
- For user interaction, the **Raycasting.cs** script enables detecting clicks on packets and displaying their details using the PacketInfo script.
- The **UIManager.cs** script manages the display and clearing of packet details on the UI using TextMeshPro components.
- The **mqttsender.py** script, additionally, acts as the data generator, sending random packet data with varying types, sizes, and IP addresses to the MQTT broker for visualization in Unity 6.

Together, these files ensure seamless packet generation, visualization, interaction, and display of relevant details. Furthermore, there is creation of various assets that support this visualization, they are as discussed

- A **Packet Prefab** is created using a simple cube or sphere, which includes components like PacketInfo, BoxCollider, and LineRenderer to facilitate interaction and visualization.
- **Access Point Prefabs** are represented using cylinder models to denote different access points (AP1, AP2, AP3) in the scene. To enhance visual distinction, materials are created for each packet type, including red for TCP, blue for UDP, green for HTTP, and yellow for FTP. These materials are applied dynamically during visualization.

- Additionally, 3D models for the *laptop* (representing the source) and APs are imported to provide a clear and engaging representation within the Unity 6 scene. These assets, combined with the scripts, enable a dynamic and interactive network visualization experience.

6. Results & Key Findings:

Basic Scenario Setup: We have one primary user equipment, i.e. our personal laptop through which we are transmitting data packets to 3 different Wi-Fi Access Points (APs), with the respective device names and IP Addresses as shown in the figure below. The packets are scheduled to be sent every 30 seconds to the different APs, and based on the packet size and type, the characteristics of the packet vary visually.

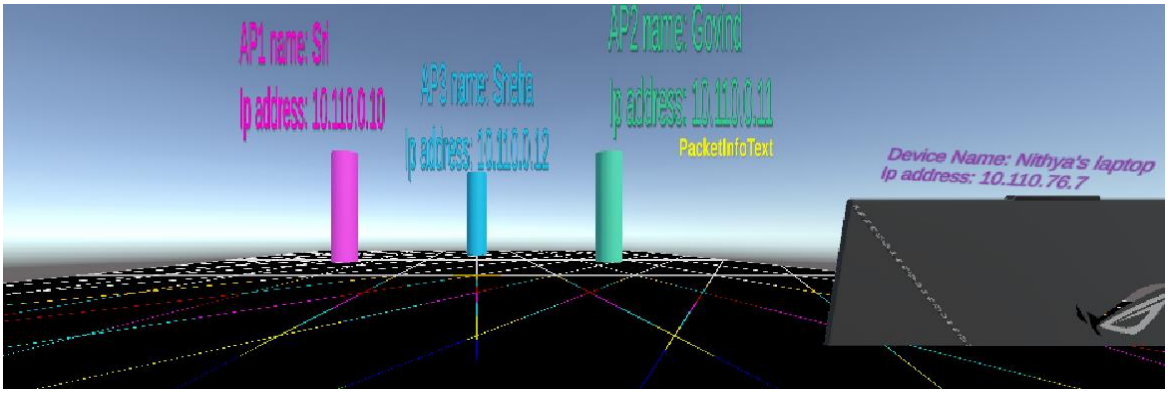

Fig.1. Basic Scenario Setup

The different packets are also color-coded: **TCP packets** **UDP packets** **HTTP packets** **FTP packets**

Unity 6 provides us with two distinct point-of-views, Scene vs Game:

- **Scene**: It resembles a *movie set* where we can edit the objects. It's merely a static image.
- **Game**: This is like the *final film* seen by the audience. This mode is more dynamic & interactive! It is nothing but what "gamers" see when they play, a real-time execution.

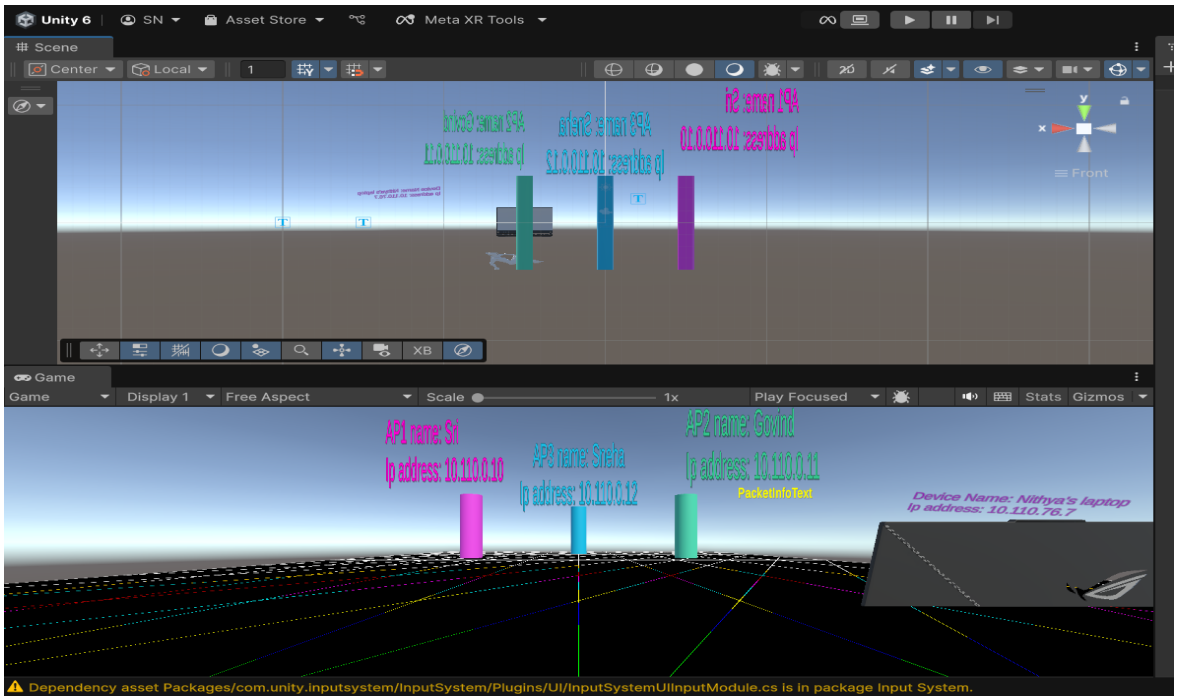

Fig.2. Scene vs Game

When we run the **mqttsender.py**, the output terminal is observed as below, which confirms the transmission of the different packets to different APs.

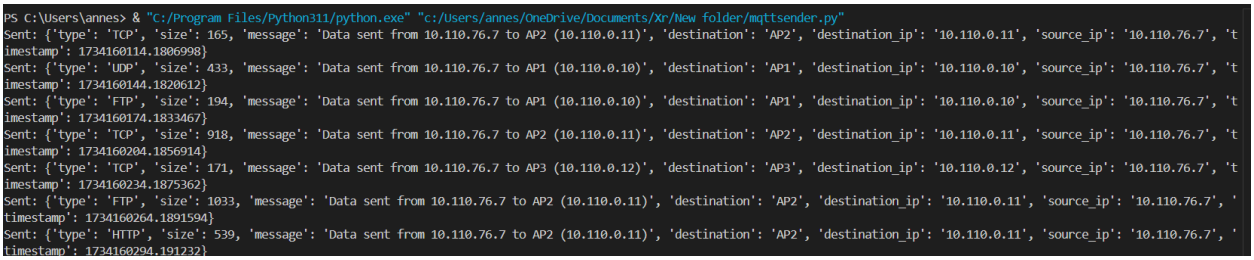

Fig.3. Transmission of different packets

Next, when we press play in Unity 6, the console shows the following output. This confirms that Unity 6 is connected to the MQTT broker successfully! We can also note that it shows the implementation of the sequence of scripts that we run, once the packets are sent. On the Unity 6 platform, we observe that:

- the packets are received via the *MQTT_Receiver.cs* file, which can be viewed in the Game mode along with the corresponding visualization via the *Packet_Visualizer.cs*.
- Furthermore, it displays the packet info. In the Game mode, when we hover over the packets it displays the packet info using *PacketInfo.cs* and *UIManager.cs*.

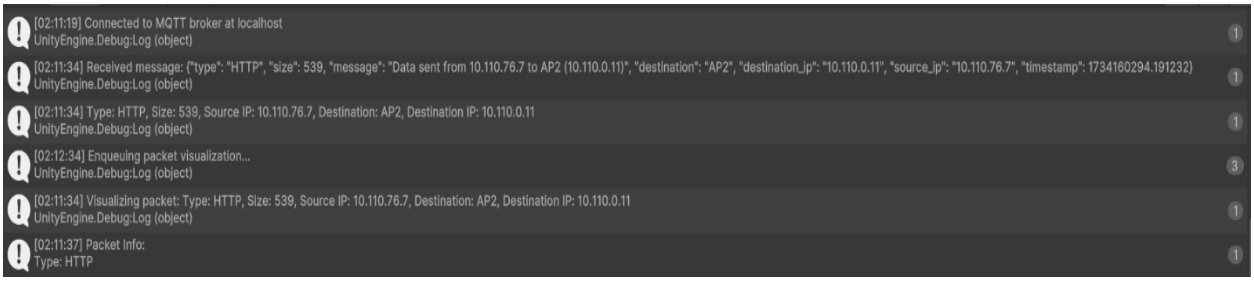

Fig.4. Unity 6 connected to MQTT Broker

As is shown in the snapshot below, when the mouse pointer hovers over the “green” packet, it displays all the info on the packet, highlighted in yellow text:

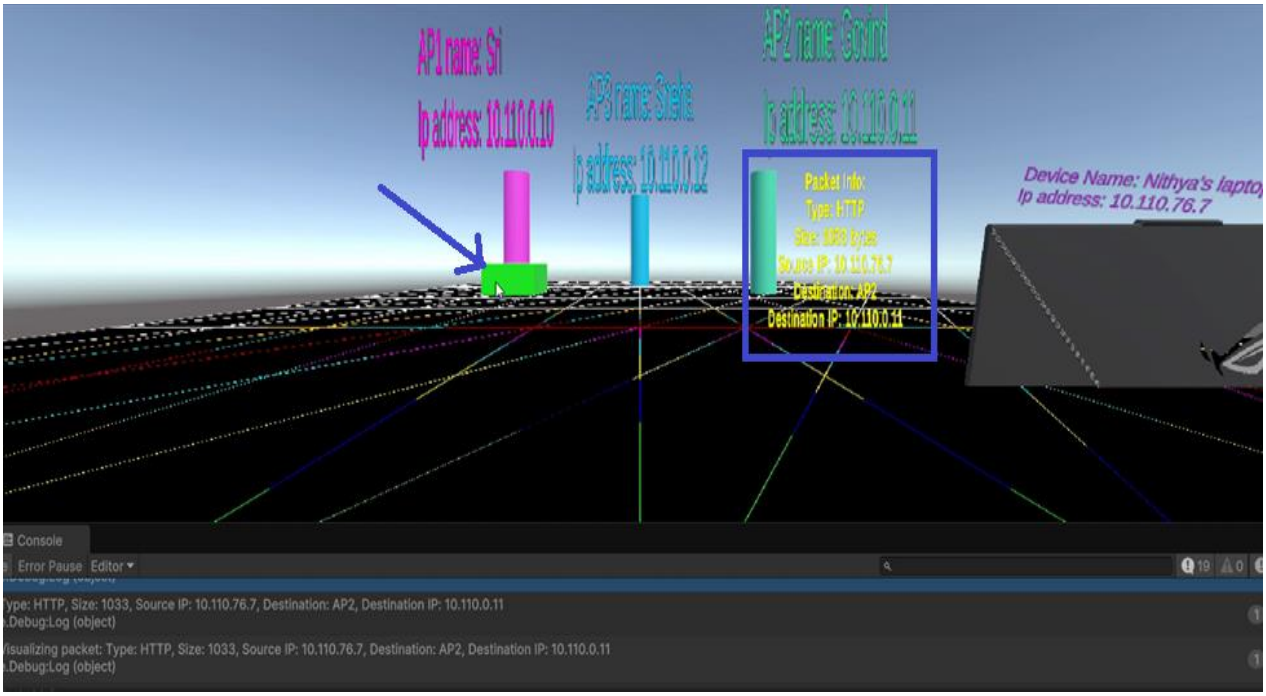

Fig.5. HTTP Packet flow from Laptop to AP 1

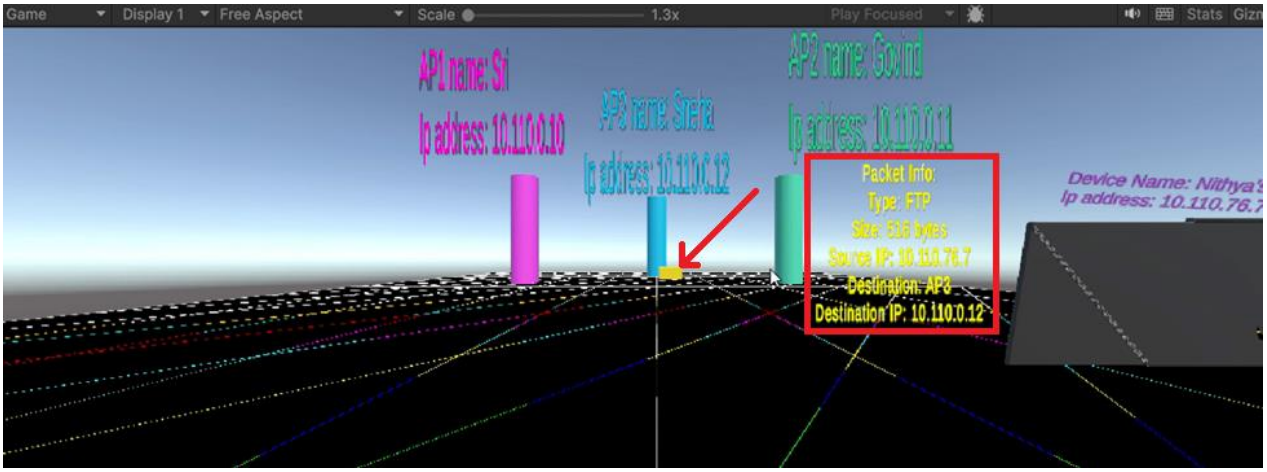

Fig.6. FTP packet flow from laptop to AP 3

All in all, the key outcomes on Unity 6 with the scenario as described above are:

- **Packet Capture and Transmission**: The system successfully captures Wi-Fi packets and transmits them to Unity 6 via MQTT. This seamless data flow ensures real-time monitoring with minimal latency, crucial for immersive MR applications.
- **Visualization**: The packets are visualized as 3D objects in Unity 6, with distinct colors representing TCP and UDP packets. This early visualization prototype provides a clear and interactive overview of network traffic.

7. Challenges

Several challenges were encountered during the implementation of the project, as briefly discussed below:

The major challenge encountered during the integration of real-time MQTT communication for packet visualization was ensuring that packet data was accurately assigned and synchronized across the system. This involved multiple components working together seamlessly: the MQTT sender script generating and publishing packets, the MQTT receiver script subscribing to these messages, and the packet visualization system in Unity 6 displaying the data in real-time.

Debugging issues such as missing script references, compile errors, and namespace conflicts required meticulous troubleshooting by verifying script attachments, resolving syntax errors, and managing dependencies between assemblies. Ensuring correct file names, using directives, and consistent references in the Unity Inspector helped resolve these challenges efficiently.

Moreover, ensuring that packets were destroyed or updated appropriately after visualization was crucial to prevent clutter or stale data in the scene. Proper handling of object instantiation and cleanup had to be carefully managed to maintain system performance and provide an accurate, real-time representation of the network activity.

The biggest challenge, in fact, was that Unity 6 often kept crashing unexpectedly! This hampered the workflow countless times, and the project had to be restarted from scratch. Despite these obstacles, the project successfully achieved interactive real-time network traffic visualization in Unity 6.

8. Future Scope

The future scope of this project is briefly summarized as follows:

a) Enhancing the visualization capabilities to incorporate advanced XR features by using devices like the Meta Quest 3 headset.

b) Real-time packet analysis could be expanded with detailed metrics, such as latency, throughput, and error rates, to provide more comprehensive network diagnostics.
c) Implementing support for additional network protocols and integrating real-world network data sources would increase the project's versatility.
d) Adding a data logging feature to retain changes made during Play Mode would prevent data loss. Performance optimization for high traffic volumes, interactive UI dashboards for better data visualization, and AI-driven analytics for anomaly detection and predictive insights could further enhance the project's utility for network security and management.

9. Conclusion

The MRNetViz project successfully showcases how Mixed Reality can revolutionize network monitoring by offering immersive, real-time visualization of network traffic. This provides spatial awareness and interactive insights that traditional monitoring tools cannot achieve, enabling users to intuitively understand traffic flow and anomalies. By visualizing packets in 3D space, the project bridges the gap between abstract network data and tangible representations, enhancing situational awareness and decision-making.

References

[1] XRShark https://meghanclark.com/publication/21xrshark/

[2] **Unity Technologies**, "Unity Development for Meta Quest 3," *Unity Documentation*. https://docs.unity3d.com/.

[3] MQTT. 2019. MQTT. http://mqtt.org.

[4] Scapy. "Scapy Documentation." https://scapy.readthedocs.io/en/latest/

[5] Unity. 2020. Unity Real-Time Development Platform | 3D, 2D, VR & AR Visualizations. https://unity.com.

[6] The Tcpdump Group. 2019. TCPDUMP/LIBPCAP public repository. https://www.tcpdump.org.

[7] Wireshark Foundation. 2020. Wireshark: Go Deep. https://www.wireshark.org./

[8] **Microsoft**, "Mixed Reality Toolkit (MRTK) for Quest Deployment," *Microsoft Documentation*. https://learn.microsoft.com/en-us/windows/mixed-reality/mrtk-unity/.

[9] **GeeksforGeeks**, "Packet Sniffing Using Scapy," *GeeksforGeeks*. https://www.geeksforgeeks.org/packet-sniffing-using-scapy/.

[10] **Python Scapy Guide**, "Building a Real-time Packet Sniffer with Python and Scapy," *Scapy Documentation* https://scapy.readthedocs.io/

[11] **Meta**, "Developing a PC Mixed Reality App with Quest 3 Passthrough," *Meta Developer Blog* https://developer.oculus.com/blog