Protocol for Dynamic Load Distribution in Web-Based AR

Sahil Athrij* Akul Santhosh* Rajath Jayashankar* Arun Padmanabhan† Sheena Mathew‡

Cochin University of Science and Technology

Abstract

In a Web-based Augmented Reality (AR) application, to achieve an immersive experience we require precise object detection, realistic model rendering, and smooth occlusion in real-time. To achieve these objectives on a device requires heavy computation capabilities unavailable on most mobile devices, this can be solved by using cloud computing but it introduces network latency issues. In this work, we propose a new network protocol named DLDAR (Dynamic Load Distribution in Web-based AR) that facilitates and standardizes methods for dynamic division of compute between client and server based on device and network condition to min-max latency and quality.

Index Terms: Computing methodologies—Computer graphics—Graphics systems and interfaces—Mixed / augmented reality; Computer systems organization—Real-time systems—Real-time system architecture

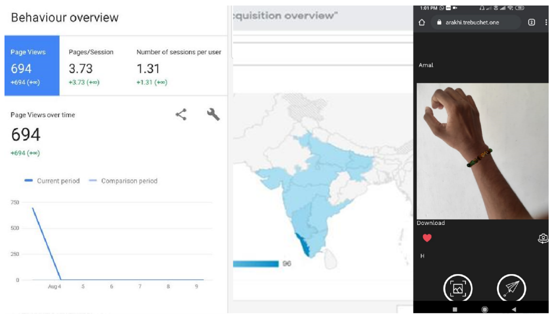

Figure 1: (Left) Usage metrics during the testing. (Middle) The spread of users during the testing of Web AR Application. (Right) Screenshot of Web App where the user tries on AR Wristband (Rakhi).

1 Introduction

Making AR accessible through the Web will make it reach out to more users by overcoming the lack of computational resources on mobile devices and the ease of using a browser over applications. Existing Web-based AR solutions have many limitations due to the lack of computational capabilities on the client-side and make human interaction with the virtual environment limited.

The limited computing and rendering capabilities of mobile device browsers make it challenging to enable high-performance Web-based AR. State-of-the-art computer vision algorithms such as YOLO, Faster R-CNN, SSD have very high computational complexity to run on mobile device browsers on the client-side. Computer vision algorithms with lesser complexity such as Tiny YOLO, DeepMon can perform object detection but with much less precision and significantly lower performance compared to the state-of-the-art algorithms. Better augmentation can be achieved by offloading the computation [4] to the server-side.

In this paper, we propose a hybrid model (DLDAR) that facilitates and standardizes compute to be dynamically divided between the client-side and server-side which prioritizes real-time performance and low data consumption. The intuitive basis of (DLDAR) is to min-max latency and quality while having real-time performance and low data consumption by selectively dividing the detection and rendering methods to the server and client-side.

2 Related Work

Mobile devices currently include a set of API's in-order to handle real-time mapping and detection through Google's ARcore, Apple's ARKit, Vuforia, etc. Although these libraries help create immersive AR experiences, they are constrained by their usability only to mobile applications. This forces users to download applications [2] onto their devices to experience high-quality AR.

Implementing Web-based AR traditionally includes a rendering library, such as WebGL or THREE.js, an AR marker tracking library like AR.js, and additionally use API's like WebXR to manage an AR session. The browsers running these libraries have limited access to device resources and hence cannot handle computational workloads for tasks such as markerless detection and photo-realistic rendering.

In cloud-based gaming, mobile devices send user inputs to the cloud which renders and executes a video game. These rendered frames are streamed back to the client's mobile device. This allows the users to experience high-quality gaming on a wide range of devices without having to download any of the games. This technology makes gaming device-independent and allows performance to scale based on need, thereby making it possible to run next-gen games without changing existing hardware [1]. It is possible to take inspiration from this method to develop a protocol for Web-based AR which dynamically provisions compute between client and server [3]. Fig. 1: we demonstrate our approach by developing a web application that enables users to share wristbands "rakhis" that can be augmented on your wrist.

3 System Architecture

We describe the system architecture for DLDAR, a client-server protocol, which optimizes for minimal latency and bandwidth usage on the client-side by performing compression and evaluating latency and compute parameters. DLDAR dynamically sets appropriate flags for availing server-side rendering, frame reconstruction, and object detection. For achieving realistic augmentation DLDAR protocol communicates between client and server for dynamically optimizing resource utilization and minimizing latency.

3.1 Client Architecture

The input video captured is scaled down to meet the protocol's upper bound threshold of 1280x720p resolution and 60 fps to maintain optimal server-side computational performance and reduce network latency.

The type of processing on the input video frames depends on the mode of operation:

*These authors contributed equally to this work.
E-mail: {sahilathrij,akul753,jayashankar.rajath}@gmail.com
†E-mail: 17cs045jois@ug.cusat.ac.in
‡E-mail: sheenamathew@cusat.ac.in

- **Low Compute High Network (LC)**
  The protocol sends reconstruct-able frames to the server, where it can be processed, and the rendering arguments can be computed. The current frame is encoded into a compressed representation called the difference frame, which is the absolute threshold difference between the pixel values taken at the same location between two different frames. Further, we find the minimum difference frame, which is the difference frame with the minimum number of changes.

- **High Compute Low Network (HC)**
  The protocol processes the frames in the client device and only sends the rendering arguments to the server for rendering. Since initializing the detection model can be compute-intensive on the client, it can be offloaded to the server. The server can send the trained weights or the region of interest to the client-side to be used for fast detection. The server-side will send correction arguments at set intervals of time, which is the default operating method.

Depending on the latency and bandwidth of the network (network availability) and the compute latency of the user device, the protocol will switch between the two modes of operation HC and LC, signaled by the MODE flag, based on the four conditions:

- **Case 1:** Compute latency is below a threshold λ_c then the protocol will default to HC operation mode.

- **Case 2:** Compute latency is above a threshold λ_c then the protocol will dynamically switch between HC and LC modes of operation.

- **Case 3:** Network latency is above a threshold λ_n then the protocol will switch between LC and HC preferring the HC mode when possible until network conditions improve.

- **Case 4:** Network latency is below a threshold λ_n then the protocol is free to use LC or HC mode, but the compute latency is the deciding factor on which mode is preferred.

The compute latency determines the frame rate of the final video output while the network latency and bandwidth determine the delay in display of the output. In LC mode large part of the compute happens on the server, which can reduce the effect of compute latency on the output. In HC mode the protocol transmits only the rendering arguments to the server, which will reduce the effect of network latency on the output. The protocol switches between the two modes to optimize both compute and network latency.

The DLDAR protocol works with either a given client-side detection algorithm or there is a default detection algorithm, any custom algorithm given must produce rendering parameters as output.

Based on HC or LC mode of operation, there are two modes of operation for the client-side algorithm:

- **Feature mode:** The server-side detection algorithm sends back the modified weights so that the client-side detection algorithm can now use the weights to perform detection on the client device. This is the detection algorithm performed in HC mode.

- **Region of Interest mode:** The detection algorithm from the server sends back a region of interest and the region of interest is tracked over the frame using the velocity and direction of motion. The pixels outside the region of interest is not considered and the new difference image is sent to the server.

In HC mode the anchor positions and rendering arguments are calculated by the client-side detection algorithm, these rendering arguments are then sent to the server. The server returns the rendered image for augmentation. In LC mode the server sends back the rendered image along with the anchor positions for augmentation.

4 Server Architecture

Reconstruction is only in LC mode, the minimum difference frame sent by the client is added with the frame which was used to create the minimum difference frame in the client, to get the current frame. This reconstructed frame is then added to the chunk of frames and the oldest frame from the chunk is removed, the frame is then sent for detection to find the anchor points and rendering parameters.

When the motion between the frames increases to 5%, the size of the difference frame would be larger than the JPEG frame. This JPEG frame is sent instead of the difference frame which is then added to the chunk of frames. A hard reset frame (LC PNG Frame) is sent at set intervals of time, this frame will overwrite the base frame of the chunk. When and if the client switches to HC mode, all the frames except the base frame is dropped.

The protocol specifies either a 3D hand pose detection algorithm [5] as the built-in default detection algorithm or a custom server-side detection algorithm that can be set by the developer using the appropriate flags.

The default detection algorithm will detect the location of the hand and create the anchor points at the center of the palm. If the developer requires he can override the anchor points by providing a function that will take the return values of the detection algorithm as input and generate rendering arguments and anchor points. The rendering argument is sent to the rendering service, and the anchor points are sent to the client.

Table 1: Comparison of different methods of transmission

Method of Transmission	Size	Quality
RAW	240 MB	Very High
PNG	60 MB	Very High
JPEG	4.8 MB	Low
DLDAR (with noise)	5.5 MB	Medium
DLDAR (without noise)	2.2 MB	High

5 Conclusion

In this paper, we have designed the DLDAR protocol to provide a method for dynamic division of compute between client and server for achieving real-time performance in a comprehensive AR web application. We prioritized cross-platform usability, the flexibility of adding custom functionality, real-time performance across devices, and low bandwidth usage to maximize the AR experience (see Table 1). We have developed an optimization mechanism based on network latency and device compute to min-max latency and quality.

References

[1] D.-Y. Chen and M. El-Zarki. Impact of information buffering on a flexible cloud gaming system. In *Proceedings of the 15th Annual Workshop on Network and Systems Support for Games*, NetGames '17, page 19–24. IEEE Press, 2017.

[2] T. Olsson and M. Salo. Online user survey on current mobile augmented reality applications. pages 75 – 84, 11 2011.

[3] Ta Nguyen Binh Duong and Suiping Zhou. A dynamic load sharing algorithm for massively multiplayer online games. In *The 11th IEEE International Conference on Networks, 2003. ICON2003.*, pages 131–136, 2003.

[4] Y. Tao, Y. Zhang, and Y. Ji. Efficient computation offloading strategies for mobile cloud computing. In *2015 IEEE 29th International Conference on Advanced Information Networking and Applications*, pages 626–633, 2015.

[5] C. Zimmermann and T. Brox. Learning to estimate 3d hand pose from single rgb images. In *IEEE International Conference on Computer Vision (ICCV)*, 2017. https://arxiv.org/abs/1705.01389.