Dynamic Load Distribution in web-based AR

SAHIL ATHRIJ*, Cisco Systems Inc, India
AKUL SANTHOSH*, New York University, USA
RAJATH JAYASHANKAR*, Rutgers University, USA
ARUN PADMANABHAN and JYOTHIS P, Cochin University of Science and Technology, India

In a web-based Augmented Reality (AR) application, to achieve an immersive experience we require precise object detection, realistic model rendering, smooth occlusion, and accurate augmentation 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 for achieving real-time performance in a comprehensive AR web application. The protocol will manage computation and network latency by dynamically provisioning compute between client and server based on device and network condition to min-max latency and quality. We demonstrate our approach by developing a web application that enables users to celebrate the regional festival "Raksha Bandan" by sharing wristbands "rakhis" that can be augmented on your wrist. This protocol can be used to create complex web AR applications with more realistic rendering, animations, and interactive environments.

CCS Concepts: • **Computing methodologies** → **Mixed / augmented reality**; • **Computer systems organization** → *Real-time system architecture.*

Additional Key Words and Phrases: network protocol, realtime AR, web technology

ACM Reference Format:
Sahil Athrij, Akul Santhosh, Rajath Jayashankar, Arun Padmanabhan, and Jyothis P. 2021. Dynamic Load Distribution in web-based AR. In *2021 5th International Conference on Artificial Intelligence and Virtual Reality (AIVR) (AIVR 2021), July 23–25, 2021, Kumamoto, Japan.* ACM, New York, NY, USA, 14 pages. https://doi.org/10.1145/3480433.3480440

1 INTRODUCTION

Augmented reality (AR) can embed virtual information seamlessly into our physical environment and provide new kinds of experiences where the world is improved by virtual content blending with the real world [9],[3]. Augmented Reality provides many tangible benefits for use cases in many industries such as tourism, entertainment, advertisement, education, manufacture etc. 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. A key component of AR is fast detection and positioning to enable seamless rendering to blend in with the real world. Existing Web-based AR solutions have many limitations due to the lack of computational capabilities in the client-side and make human interaction with the virtual environment limited.

*Indicates equal contribution to this research.

Manuscript submitted to ACM

Web-based AR provisions for native cross platform and lightweight applications on mobile devices. Mobile- applications based AR lacks flexibility and requires users to download and install applications in advance. These applications are also not convenient for cross-platform deployment. Web-based AR provides a pervasive platform that is independent of these drawbacks.

Although web-based AR provides for a promising approach to a cross-platform, lightweight implementation on mobile devices. There are still many challenges that arise while implementing web-based AR in real cases. The limited computing and rendering capabilities of mobile device browsers make it challenging to enable high-performance web-based AR. The image data from sensors present on the mobile devices are processed using computer vision algorithms to detect objects in the physical environment [29]. State-of-the-art computer vision algorithms such as YOLO [23], Faster R-CNN [26], SSD [15] 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 [23], DeepMon [16] that can perform object detection but with much less precision and significantly lower performance compared to the state-of-the-art algorithms. With the significant overhead of computation on the client-side, interaction with the user becomes limited. Better augmentation can be achieved by offloading the computation [28] to the server-side. However, computation offloading may bring an added communication delay, which can hinder user experience.

The different implementation mechanism for computation offloading to server-side depends mainly on the following two factors.

- `Prioritising real-time performance`: Mobile devices that have access to high speed internet can send uncompressed raw images to server for processing. However, sending raw images can lead to high data consumption but while retaining high real-time performance.
- `Prioritising low data consumption`: In case the mobile devices do not have access to high speed internet or have low data bandwidth, raw images can be compressed to reduce data usage. However running compression and decompression algorithms on client and server side respectively introduces a considerable amount of delay thereby hindering real-time performance.

In this paper, we propose a hybrid model (DLDAR) that facilitates and standardises compute to be dynamically divided between the client-side and server-side which prioritises 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 server and client side.

2 RELATED WORK

Our work relates to mobile augmented reality, web-based augmented reality and cloud gaming.

2.1 Mobile Augmented Reality

In recent years, the number of applications that run Mobile Augmented Reality (MAR) has drastically increased thanks to increased computational resources available on Mobile Devices[25],[21]. Most MAR applications can run on current high-end mobile devices, but in order to achieve an immersive experience processes such as precise object detection, realistic model rendering, smooth occlusion [10] and accurate augmentation in real-time cannot be handled by mobile devices. Further, most mid-range and low-end mobile devices are not capable of achieving admissible quality of augmentation due to limited resources and battery capacity.

Mobile devices currently include a set of API's in-order to handle real-time mapping and detection through Google's ARcore [19], Apple's ARKit [19], PTC's Vuforia [20] 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 [18] onto their devices to experience high quality AR. Therefore, extensive research is being conducted to develop Web based AR systems which can deliver similar immersive experience.

2.2 Web Based Augmented Reality

Implementing Web based AR [22] traditionally includes a rendering library, such as WebGL [24] 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 marker less detection and photo-realistic rendering. Further, technologies such as WebXR are still in their infancy with limited capabilities and does not include provisions to add custom detection algorithms.

To bring native application performance to Web browsers, technologies like SceneViewer by Google and ARQuickLook by Apple handles the task of rendering models, light conditions, surface detection and placement in the browser[14]. These technologies are browser specific, and therefore not inter-operable [11], [17]. Further, these technologies limit developer options such as adding custom detection algorithms and adding interaction [12]. For these reasons, server-based AR systems are being developed as alternatives to client only AR systems.

2.3 Cloud Based Gaming Technology

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 [13]. 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 [4].

Google Stadia[2], Microsoft xCloud, Amazon Luna, Nvidia GeForce Now and PlayStation Now are all cloud-based gaming technologies that allow users to streams games on their devices. In these papers, reducing network latency of computational offloading is at the core of mobile cloud gaming. 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 [27].

3 SYSTEM OVERVIEW

DLDAR allows the developer to create seamless AR applications on the web that have server-side object rendering and dynamic client and server-side detection which allows the AR application to perform with maximum rendering quality and minimum use of network bandwidth along with automatically adjusting for the user's device performance.

We implement the DLDAR protocol to provide the following features:

- The protocol takes into consideration individual user device compute latency to determine client-side utilisation.
- The protocol creates a persistent communication channel to maintain a consistent state for the detection algorithms.
- The protocol implements accurate and precise but computationally expensive object detection algorithm in the server-side.

- The protocol implements computationally inexpensive but less precise object detection algorithm in the client-side.
- The protocol dynamically switches between server and client-side object detection for optimising compute and network latency.
- The protocol detects and recovers from detection errors caused by faster but less precise algorithms that run on client-side.
- The protocol provides the anchor points from the detected object to the client-side for augmentation.
- The protocol provides the rendering parameters to the rendering service for object and camera placement.

Based on the compute and network latency, the compression ratio and method will be dynamically updated. Rendering will always take place in the server-side and the rendered result will be sent to the client-side; the quality of render will be updated as per network latency.

To the best of our knowledge, we are the first to develop a dynamic protocol for web-based AR that enables the system to adapt its performance based on network latency and available computational resources.

3.1 Desired User Experience

Our ideal end goal is as follows. As a user opens the link to the AR app in a web browser, an interactive augmented reality experience will start.

The experience will remain smooth even if the network conditions of the user's device start to degrade or even if the user opens another task that consumes lots of bandwidth or CPU usage.

4 SYSTEM ARCHITECTURE

In this section, we describe the system architecture for DLDAR, a client server protocol, which optimises for minimal latency and bandwidth usage on 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.

4.1 Client Architecture

The client side architecture of the protocol as shown in Figure 1, defines the standard of data that is to be sent to the protocol, the standard in which the rendered output will be produced and the functions that can be plugged in to the client side and the modes of operation of the protocol.

4.1.1 Frame Processing. 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:

- `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.
- `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.

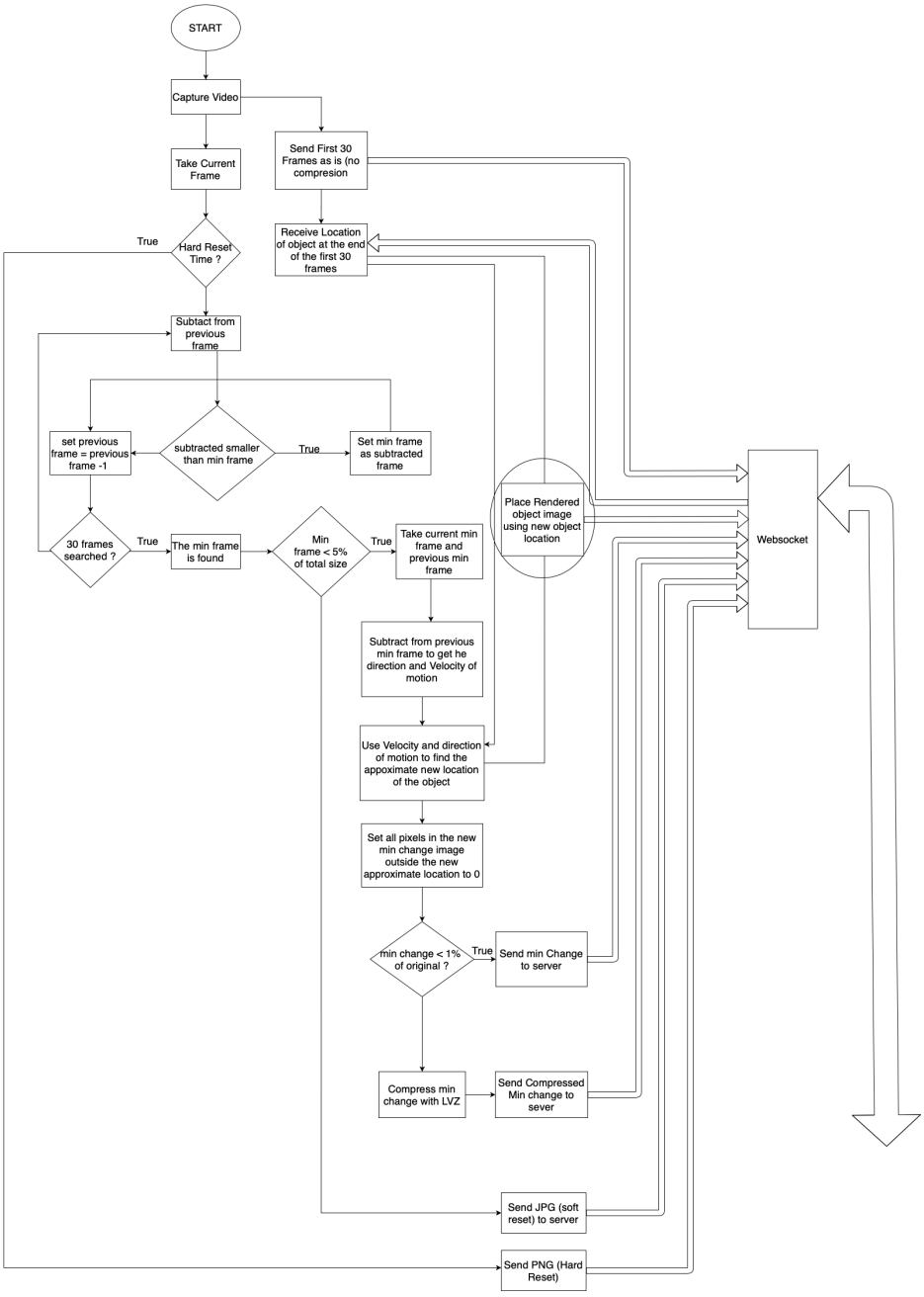


Fig. 1. Client Architecture Overview. The thin arrows represent all the computation happening on the client device browser. The thick arrows represent the flow of data through the web socket to the network.

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, signalled 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. This can be overridden by the developer using the force LC mode flag (FLC Flag).
- `Case 2`: Compute latency is above a threshold λ_c then the protocol will dynamically switch between HC and LC modes of operation. This can be overridden by the developer using FLC flag.
- `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. This cannot be overridden by developer.
- `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 developer can override the preference using FLC flag.

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 optimise both compute and network latency.

The first n frames are sent to the server in LC mode as PNG frames (`LC PNG mode`) in-order to initialise the detection model in the server.

4.1.2 LC Frame Compression. When the DLDAR protocol switches to LC mode, LC frame compression is used to create the reconstruct-able frames that are sent to the server.

The current frame is encoded into a compressed representation called the difference frame f', which is absolute threshold difference between the pixel values taken at the same location between two different frames. The protocol's performs is not affected by ambient lighting changes and can work in real time without additional workload on client side. We find f' as,

$$f' = f_i - f_{i-n} = \begin{cases} 0 \quad \text{if } f_i[i,j,k] - f_{i-n}[i,j,k] < 0 \\ \\ f_i[i,j,k] - f_{i-n}[i,j,k] \text{ otherwise} \end{cases} \tag{1}$$

$$\text{where} \quad i \rightarrow 0 - w$$
$$j \rightarrow 0 - h$$
$$k \rightarrow 0, 1, 2$$

Further we find the minimum difference frame f'_{min}, which is the difference frame with the minimum number of changes given by,

$$f'_{min} = min(\{\forall x : \quad f'_x = f_i - f_{i-x}\}) \tag{2}$$

where min() selects most sparse matrix f'_x as the f'_{min}.

If the size of f'_{min} is more than 5% of the original image, a soft reset JPEG [7] frame is sent, otherwise the minimum difference frame is sent. On every hard reset interval, the hard reset PNG [8] frame is sent instead of the difference frame. The threshold size percentage for soft reset and the interval time for hard reset can be set by the developer.

4.1.3 HC Frame Processing. When DLDAR switches to HC mode, all the processing is done on client-side except for rendering, as photo realistic rendering is computationally taxing on the client device.

After the first n frames are sent via the LC PNG mode, the server-side detection algorithm is initialised. There are two modes of operation which can controlled by the developer using the correction flag (CR flag) :-

- `Correcting mode`: 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.
- `Non-Correcting mode`: In this mode of operation it is the expected the developer will have a custom client-side detection algorithm, the server will not send any correction or weight data to the client-side.

4.1.4 Detection on Client. 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.

Velocity of motion is the amount of travel of the centre of mass between two consecutive difference frames. The centre of mass of a frame $c(f)$ can be calculated by,

$$c(f) = \left(\frac{\Sigma x}{w}, \frac{\Sigma y}{h}\right) \quad \forall (x, y) \in (w, h) \text{ where } 0 < f[x, y, \cdots] \tag{3}$$

The centre of mass for each frame is the average location of the non-zero pixels. The velocity of motion $v(f)$ and direction of motion $\hat{v}(f)$ is calculated using the difference of the centre of mass in two consecutive frames.

$$v(f) = \frac{|c(f_i) - c(f_{i-1})|}{\tau} \tag{4}$$

$$\hat{v}(f) = \frac{c(f_i) - c(f_{i-1})}{\tau v(f)} \tag{5}$$

In-order to reduce computation, approximate velocity of motion can be calculated by taking the centre of mass of the differential frame f'' given by,

$$f'' = f'_i - f'_{i-1} \tag{6}$$

$$v(f) = c(f'') \tag{7}$$

The region of interest is the approximate area containing the object, and the frame region (FR) is the segmented product of ROI and frame, and given by,

$$ROI(f) = \begin{cases} 1 & \text{if ray from i,j intersect through ROI} \\ & \text{odd number of times} \\ 0 & \text{otherwise} \end{cases} \tag{8}$$

$$FR(f, ROI) = f[i, j, k] \ \& \ ROI[i, j, k] \tag{9}$$

$$\text{where} \quad i \rightarrow 0 - w$$
$$j \rightarrow 0 - h$$
$$k \rightarrow 0, 1, 2$$

4.1.5 Augmentation of Rendered Object. In HC mode the anchor positions and rendering arguments are calculated by the client side detection algorithm, these rendering arguments are then send 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. The developer can choose to override the default anchoring with either static or dynamic anchor points.

4.2 Server Architecture

The server side architecture of the protocol as shown in Figure 2 defines the standard of data that is sent by the protocol, the standard in which the rendering service should send the rendered object to the protocol and the method of data transmission between the detection algorithms.

4.2.1 Web Socket Connection. This protocol works on top of the web socket [6] protocol for bidirectional contiguous transmission of data. The control bits are sent as headers and data bit is sent as body of the data chunk.

4.2.2 Reconstruction. 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.

Reconstruction is bypassed in the following three cases:

- `Soft Reset`: 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. This is signalled by the SRS flag (Soft Reset State flag).
- `Hard Reset`: A hard reset frame (LC PNG Frame) is sent at set intervals of time, this frame will overwrite the base frame of the chunk. This is signalled by the HRS flag (Hard Reset State flag).
- `HC mode`: When and if the client switches to HC mode, all the frames except the base frame is dropped. This is controlled by the MODE flag.

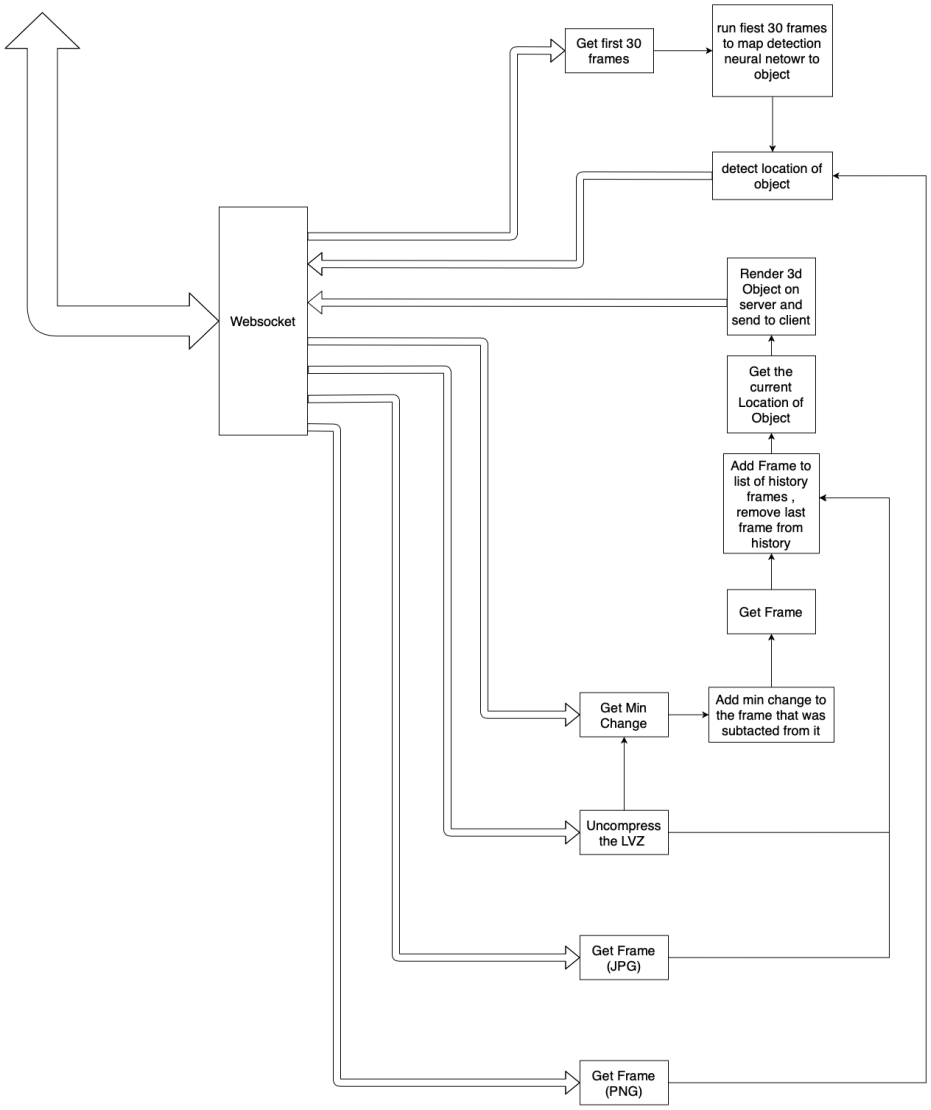


Fig. 2. Server Architecture Overview. The server provides the communication gateway represented by the thick arrow through the Web socket. The thin arrows represent the computation happening on the server.

4.2.3 Detection on Server. The protocol specifies either a 3D hand pose detection algorithm [30] as the built in default detection algorithm, or a custom server side detection algorithm 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 centre of the palm of the hand. 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.

In HC mode there is no detection in the server side, but if the corrections flag is set to true, there is infrequent server side detection to find any errors in the client side detection.

4.2.4 Rendering Service. The protocol specifies no 3D rendering service but the default rendering parameters produced are as follows :-

- `Position of the object`: It specifies the position of the object in 3D space, this is usually at the origin unless there is more than one object to be rendered. If there is more than one object to be rendered the object with the higher triangle count is set at the origin and the all other objects are placed relative to it.
- `Rotation of the object`: By default if there is only one object to be rendered then the rotation of the object is set to zero. If there is more than one object to be rendered, the rotation of all the other objects is set relative to the object with the higher triangle count.
- `Position of the camera`: This is the position of the camera in (x,y,z) axis, this is made to approximately match (x,y,z) position of the camera that has taken the original image.
- `Rotation of the camera`: Rotation of the camera is made so that the camera is always pointing to the origin, this is made for easier compute and rendering. The developer can use custom function to override this and produce rotations that are askew from the origin.
- `Position of lighting`: This specifies the position of lightning as a list with at most three light sources, one behind the camera and two behind the plane of the object. The developer can use a custom function to override this and produce more lights for a more complicated lighting setup.
- `Intensity of the lights`: By default the intensity of the lighting is (1,0.5,0.25), this can also be overridden to produce a more complicated lighting scenario.

The rendering parameters are returned as a JSON object [1] that can be passed to the rendering function. While the protocol does come with a default rendering service that will take rendering parameters and a 3D model to produce a rendered output, the capabilities of the rendering service is limited and it is highly recommended that the developer supply their own rendering service, as 3D rendering is outside of the scope of the protocol.

4.2.5 Render Result Transmission. The rendered image from the rendering service is sent to the client along with the anchor points and the rendering parameters. The transmission of the image is dependent on the network quality, the protocol will try to maintain 30 fps transmission, with PNG being the preferred method of transmission, if the network quality starts to reduce, the protocol will decrease the resolution of the PNG till 256x256, after which it will switch to JPEG.

The lighting parameters are also transmitted so that any modification in the original image can be made to make sure that the rendered image and the original image blend together.

5 PERFORMANCE EVALUATION

In this section, we provide an extensive evaluation of DLDAR protocol by comparing various compression methods and evaluating them based on size, latency and achieved quality of augmentation. We demonstrate our approach by developing a web application which enables users to celebrate the regional festival “Raksha Bandan” by sharing wristbands “rakhis” that can be augmented on your wrist. We hosted the web application on an EC2 Instance (t2.medium configuration) with Intel Xeon 3.3Ghz CPU and 4GB RAM. We initially tested the client on Safari and Chrome Browsers

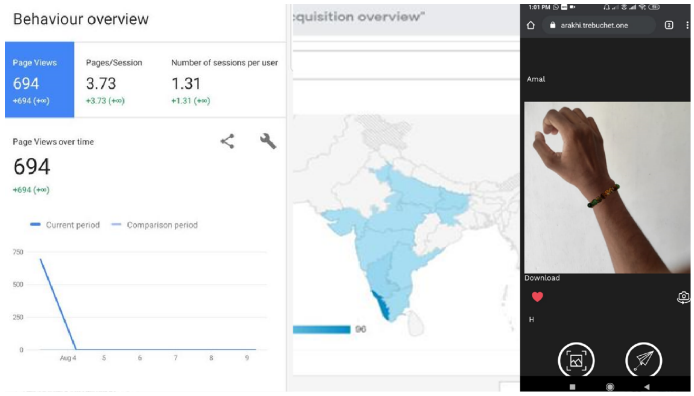
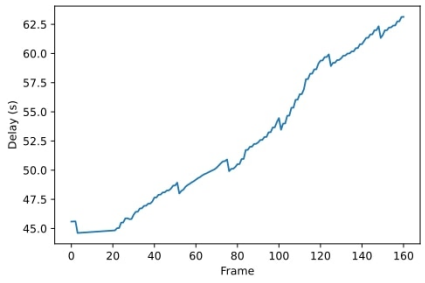
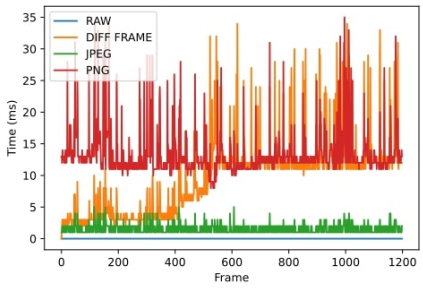


Fig. 3. (Left) Usage metrics during the testing. (Middle) The spread of users during the testing of Web AR Application. (Right) Screenshot of Web App where the user tries on AR Wristband (Rakhi).



(a) Frame Delay of H264 Compression



(b) Frame Delay of Different Formats of Compression. At around 400 frames, frame motion was introduced which reduces performance of difference frame.

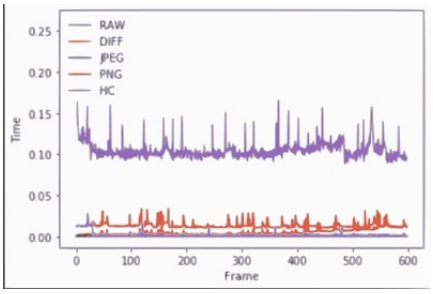
Fig. 4. Comparison between frame delay of different formats

on high end mobile devices like iPhone X, Galaxy S8 and low end devices like iPhone 5s and Xiaomi Redmi Note 8, while the complete testing scaled over 650 users across India on a multitude of devices Figure 3.

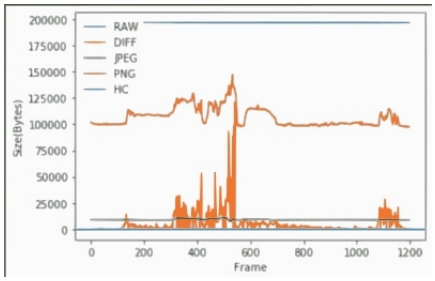
The input frames included the detection object (hand) in various positions with lateral and rotational movements. Further, to evaluate performance under noise we introduced random motion in the input frames. During the testing, compute latency for object detection λ_c was below the threshold of 30 FPS for all the mobile devices and therefore the protocol defaulted to forced LC mode. Although while testing on a PC - with 3.8Ghz Ryzen 5-3600x CPU, and 8GB RAM. We observe that λ_c maintained above 30FPS and the protocol dynamically switched between HC and LC Mode based on the network latency λ_n, with a threshold RTT set to 90ms.

5.1 Latency Evaluation

We evaluate minimum latency for sending frames by calculating the total time taken to compress and decompress frames in the client and server side respectively. We first tested with H264 [5], a very efficient codec that provides high-quality

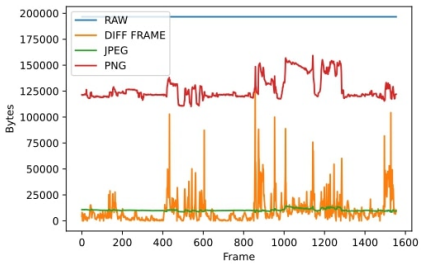


(a) Frame Delay of different Compression formats vs HC Mode of operation . HC mode is an order of magnitude slower than LC mode

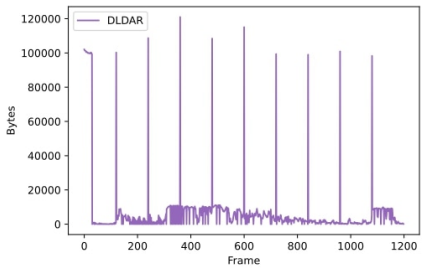


(b) Compression Size of Different Format vs HC mode of Operation. HC mode is 3 orders of magnitude smaller than even the smallest compression method JPEG

Fig. 5. When device uses HC mode in same scenario



(a) Transmission Size of Different Formats of Compression. At around 400 and 900 frames, frame motion was introduced which increases the size of the difference frame.



(b) Transmission Size of DLDAR protocol. The large spikes indicate a hard reset to a PNG frame, while the small spikes are soft resets to JPEG frames.

Fig. 6. Comparison between transmission size of different formats

images and uses a minimal amount of bandwidth. We observe that H264 takes more than a minute of compression latency (Figure 4a) while transferring at 60 frames per second, making it not suitable for real time applications.

The fastest way of sending frames to the server while having unlimited bandwidth is RAW images. In these conditions we compare the compression latency of JPEG, PNG and Difference Frame with RAW image (Figure 4b). We observed that maintaining a latency of 15 to 30 ms gives optimal performance, while all the methods maintained reasonable latency, JPEG and Difference frame performed the best. Difference frame starts loosing its performance once considerable amount of frame motion is introduced (Figure 4b). PNG produces better picture quality but has higher compression latency compared to JPEG and Difference Frame. From this we find that the most suitable format for transferring of data is either difference frame or JPEG and the protocol optimises latency by using these formats. From the data (Figure 5a) we can see that is is only possible to get 8-10 FPS using HC mode. Thus we default to LC mode unless the network condition gets degrade to a point where we are only getting 10-15 FPS in LC mode.

5.2 Size Evaluation

We observe that RAW takes the highest amount of bandwidth and hence removed from the list of possible compression techniques. From the Figure 6a, we observe that JPEG and Difference Frame take up least amount of bandwidth for

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

transferring frames, while JPEG performs consistently even when considerable frame motion is introduced. From the above testing we can see that there is no one compression method that yields the best result, thus we must dynamically switch between Difference frame and JPEG to get optimal data transfer, while intermediately switching to PNG to preserve picture quality. From the data (Figure 5b) we can see that data transmitted in HC mode is of negligible size when compared to any of the frame transmission methods.

Figure 6b shows the protocols performance by mapping data usage in a simulated test consisting of normal and induced frame motion, which achieved around 5MB data usage over a 40s period on mobile devices while 220KB of data was used for same period on PC. We observed that the protocol maintained 30-60 FPS consistently across the mobile devices with good quality of augmentation. The same test was conducted on different methods of transmission and the data size and quality was measures as detailed in Table 1. While transmitting frames at 720p RAW and PNG produced high picture quality which helps in better object detection while taking up large amounts of data. JPEG took up small amounts of data, but compromised on picture quality thereby hindering accurate object detection. DLDAR without much frame motion took up the least amount of transmission data while maintaining high picture quality and resets to JPEG when exhibiting high frame motion.

6 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, flexibility of adding custom functionality, real-time performance across devices and low bandwidth usage to maximise the AR experience. We have developed an optimization mechanism based on network latency and device compute to min-max latency and quality. We have implemented and deployed the DLDAR protocol and validated its performance through extensive testing and deployed a web application that was tested by a large user group.

REFERENCES

[1] T. Bray. 2014. The JavaScript Object Notation (JSON) Data Interchange Format. *RFC* 7159 (2014), 1–16.
[2] Marc Carrascosa and Boris Bellalta. 2020. Cloud-gaming:Analysis of Google Stadia traffic. arXiv:2009.09786 [cs.NI]
[3] D. Chatzopoulos, C. Bermejo, Z. Huang, and P. Hui. 2017. Mobile Augmented Reality Survey: From Where We Are to Where We Go. *IEEE Access* 5 (2017), 6917–6950. https://doi.org/10.1109/ACCESS.2017.2698164
[4] De-Yu Chen and Magda El-Zarki. 2017. Impact of Information Buffering on a Flexible Cloud Gaming System. In *Proceedings of the 15th Annual Workshop on Network and Systems Support for Games* (Taipei, Taiwan) *(NetGames '17)*. IEEE Press, 19–24.
[5] Jian-Wen Chen, Chao-Yang Kao, and Youn-Long Lin. 2006. Introduction to H.264 advanced video coding, Vol. 2006. 6 pp.–. https://doi.org/10.1109/ASPDAC.2006.1594774
[6] I. Fette and A. Melnikov. 2011. The WebSocket Protocol. *RFC* 6455 (2011), 1–71.
[7] Borko Furht (Ed.). 2008. *JPEG*. Springer US, Boston, MA, 377–379. https://doi.org/10.1007/978-0-387-78414-4_98
[8] Borko Furht (Ed.). 2008. *Portable Network Graphics (Png)*. Springer US, Boston, MA, 729–729. https://doi.org/10.1007/978-0-387-78414-4_181
[9] IEEE Digital Reality 2020 (accessed September 16, 2020). *Standards*. IEEE Digital Reality. https://digitalreality.ieee.org/standards

[10] K. Kiyokawa, M. Billinghurst, B. Campbell, and E. Woods. 2003. An occlusion capable optical see-through head mount display for supporting co-located collaboration. In *The Second IEEE and ACM International Symposium on Mixed and Augmented Reality, 2003. Proceedings.* 133–141. https://doi.org/10.1109/ISMAR.2003.1240696

[11] Rob Kooper and Blair MacIntyre. 2003. Browsing the Real-World Wide Web: Maintaining Awareness of Virtual Information in an AR Information Space. *International Journal of Human-Computer Interaction* 16, 3 (1 Jan. 2003), 425–446. https://doi.org/10.1207/S15327590IJHC1603_3

[12] Tobias Langlotz, Thanh Quoc Nguyen, Dieter Schmalstieg, and Raphael Grasset. 2014. Next Generation Augmented Reality Browsers: Rich, Seamless, and Adaptive. *Proc. IEEE* 102, 2 (2014), 155–169.

[13] Kyungmin Lee, David Chu, Eduardo Cuervo, Johannes Kopf, Yury Degtyarev, Sergey Grizan, Alec Wolman, and Jason Flinn. 2015. Outatime: Using Speculation to Enable Low-Latency Continuous Interaction for Mobile Cloud Gaming. In *Proceedings of the 13th Annual International Conference on Mobile Systems, Applications, and Services* (Florence, Italy) *(MobiSys '15)*. Association for Computing Machinery, New York, NY, USA, 151–165. https://doi.org/10.1145/2742647.2742656

[14] Li Liu, Wanli Ouyang, Xiaogang Wang, Paul Fieguth, Jie Chen, Xinwang Liu, and Matti Pietikäinen. 2019. Deep Learning for Generic Object Detection: A Survey. arXiv:1809.02165 [cs.CV]

[15] Wei Liu, Dragomir Anguelov, Dumitru Erhan, Christian Szegedy, Scott Reed, Cheng-Yang Fu, and Alexander C. Berg. 2016. SSD: Single Shot MultiBox Detector. In *Computer Vision – ECCV 2016*, Bastian Leibe, Jiri Matas, Nicu Sebe, and Max Welling (Eds.). Springer International Publishing, Cham, 21–37.

[16] Huynh Nguyen Loc, Youngki Lee, and Rajesh Krishna Balan. 2017. DeepMon: Mobile GPU-based Deep Learning Framework for Continuous Vision Applications.. In *MobiSys*, Tanzeem Choudhury, Steven Y. Ko, Andrew Campbell, and Deepak Ganesan (Eds.). ACM, 82–95. http://dblp.uni-trier.de/db/conf/mobisys/mobisys2017.html#LocLB17

[17] B. MacIntyre, A. Hill, H. Rouzati, M. Gandy, and B. Davidson. 2011. The Argon AR Web Browser and standards-based AR application environment. In *2011 10th IEEE International Symposium on Mixed and Augmented Reality*. 65–74. https://doi.org/10.1109/ISMAR.2011.6092371

[18] Thomas Olsson and Markus Salo. 2011. Online user survey on current mobile augmented reality applications. *2011 10th IEEE International Symposium on Mixed and Augmented Reality, ISMAR 2011*, 75 – 84. https://doi.org/10.1109/ISMAR.2011.6092372

[19] Z. Oufqir, A. El Abderrahmani, and K. Satori. 2020. ARKit and ARCore in serve to augmented reality. In *2020 International Conference on Intelligent Systems and Computer Vision (ISCV)*. 1–7. https://doi.org/10.1109/ISCV49265.2020.9204243

[20] PTCGroup. 2020. Vuforia Developer Portal. https://developer.vuforia.com/

[21] X. Qiao, P. Ren, S. Dustdar, L. Liu, H. Ma, and J. Chen. 2019. Web AR: A Promising Future for Mobile Augmented Reality—State of the Art, Challenges, and Insights. *Proc. IEEE* 107, 4 (2019), 651–666. https://doi.org/10.1109/JPROC.2019.2895105

[22] X. Qiao, P. Ren, S. Dustdar, L. Liu, H. Ma, and J. Chen. 2019. Web AR: A Promising Future for Mobile Augmented Reality—State of the Art, Challenges, and Insights. *Proc. IEEE* 107, 4 (2019), 651–666. https://doi.org/10.1109/JPROC.2019.2895105

[23] J. Redmon and A. Farhadi. 2017. YOLO9000: Better, Faster, Stronger. In *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 6517–6525. https://doi.org/10.1109/CVPR.2017.690

[24] Hafez Rouzati, Luis Cruiz, and Blair MacIntyre. 2013. Unified WebGL/CSS Scene-Graph and Application to AR. In *Proceedings of the 18th International Conference on 3D Web Technology* (San Sebastian, Spain) *(Web3D '13)*. Association for Computing Machinery, New York, NY, USA, 210. https://doi.org/10.1145/2466533.2466568

[25] Ryan Shea, Andy Sun, Silvery Fu, and Jiangchuan Liu. 2017. Towards Fully Offloaded Cloud-Based AR: Design, Implementation and Experience. In *Proceedings of the 8th ACM on Multimedia Systems Conference* (Taipei, Taiwan) *(MMSys'17)*. Association for Computing Machinery, New York, NY, USA, 321–330. https://doi.org/10.1145/3083187.3084012

[26] N. Sun, Y. Zhu, and X. Hu. 2019. Faster R-CNN Based Table Detection Combining Corner Locating. In *2019 International Conference on Document Analysis and Recognition (ICDAR)*. IEEE Computer Society, Los Alamitos, CA, USA, 1314–1319. https://doi.org/10.1109/ICDAR.2019.00212

[27] Ta Nguyen Binh Duong and Suiping Zhou. 2003. A dynamic load sharing algorithm for massively multiplayer online games. In *The 11th IEEE International Conference on Networks, 2003. ICON2003.* 131–136. https://doi.org/10.1109/ICON.2003.1266179

[28] Y. Tao, Y. Zhang, and Y. Ji. 2015. Efficient Computation Offloading Strategies for Mobile Cloud Computing. In *2015 IEEE 29th International Conference on Advanced Information Networking and Applications*. 626–633. https://doi.org/10.1109/AINA.2015.246

[29] X. Zhou, W. Gong, W. Fu, and F. Du. 2017. Application of deep learning in object detection. In *2017 IEEE/ACIS 16th International Conference on Computer and Information Science (ICIS)*. 631–634. https://doi.org/10.1109/ICIS.2017.7960069

[30] Christian Zimmermann and Thomas Brox. 2017. Learning to Estimate 3D Hand Pose from Single RGB Images. In *IEEE International Conference on Computer Vision (ICCV)*. https://lmb.informatik.uni-freiburg.de/projects/hand3d/ https://arxiv.org/abs/1705.01389.