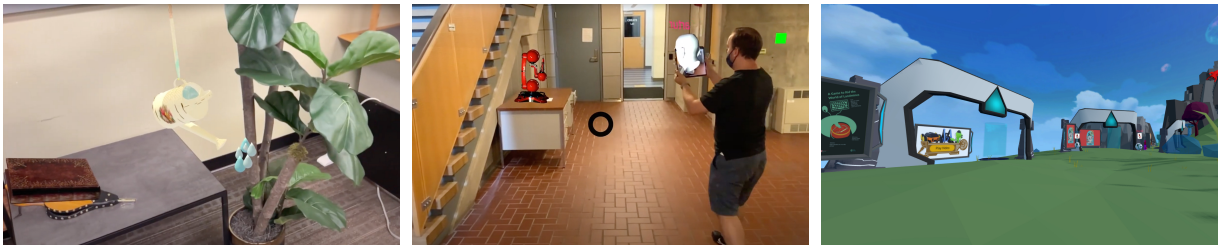
ARENA: The Augmented Reality Edge Networking Architecture

Nuno Pereira* Anthony Rowe* Michael W Farb* Ivan Liang* Edward Lu* Eric Riebling*

Carnegie Mellon University



(a) (b) (c)

Figure 1: ARENA demo applications. (a) Digital and physical art mashup: Rube Goldberg machine with digital objects (watering can, book) interacting with real objects (servo-controlled air blower, plant); (b) Industrial digital twins: visualizing robotic arm movements; (c) Virtual worlds: multi-user virtual reality.

Abstract

Many have predicted the future of the Web to be the integration of Web content with the real-world through technologies such as Augmented Reality (AR). This has led to the rise of Extended Reality (XR) Web Browsers used to shorten the long AR application development and deployment cycle of native applications especially across different platforms. As XR Browsers mature, we face new challenges related to collaborative and multi-user applications that span users, devices, and machines. These *collaborative* XR applications require: (1) networking support for scaling to many users, (2) mechanisms for content access control and application isolation, and (3) the ability to host application logic near clients or data sources to reduce application latency. In this paper, we present the design and evaluation of the AR Edge Networking Architecture (ARENA) which is a platform that simplifies building and hosting collaborative XR applications on WebXR capable browsers. ARENA provides a number of critical components including: a hierarchical geospatial directory service that connects users to nearby servers and content, a token-based authentication system for controlling user access to content, and an application/service runtime supervisor that can dispatch programs across any network connected device. All of the content within ARENA exists as endpoints in a PubSub scene graph model that is synchronized across all users. We evaluate ARENA in terms of client performance as well as benchmark end-to-end response-time as load on the system scales. We show the ability to horizontally scale the system to Internet-scale with scenes containing hundreds of users and latencies on the order of tens of milliseconds. Finally, we highlight projects built using ARENA and showcase how our approach dramatically simplifies collaborative multi-user XR development compared to monolithic approaches.

Index Terms: Computer systems organization—Architectures—Distributed architectures; Computing methodologies—Computer graphics—Graphics systems and interfaces Mixed / augmented reality

*e-mail:{npereira, agr, mwfarb, hiliang, elu2, er1k}@andrew.cmu.edu. Nuno Pereira is a visiting scholar from the School of Engineering of the Polytechnic of Porto (ISEP), Portugal.

1 Introduction

People have long imagined mixed reality systems where users could seamlessly interact with digital content that is tightly coupled with the physical world [44]. This has catalyzed a number of enabling AR technologies like localization, tracking, gesture recognition, and display hardware. Most recently, we have seen platforms like AR Kit/Core [24, 29] support mobile AR applications like Google Maps, Pokemon GO, IKEA Place, etc [48]. These are currently developed as monolithic apps designed to run in isolation with an arduous develop, debug, and deploy design cycle. To address this challenge, researchers have spent the last two decades working on extending Web browser functionality to include AR and VR [7, 11, 14, 20]. Web browsers naturally capture the spirit of a single front-end application environment (Browser) that retrieves a standard description of data (HTML) from many networked resources (URLs pointing to Web Servers) and interprets/renders the data based on the clients' local capabilities. Learning from the success of traditional Web Applications, this approach provides an easy mechanism to quickly build applications that span a wide range of platforms, from phones and tablets to headsets. As a result, we are now seeing standards such as WebXR [46] making their way into commercial browsers, as these "Extended Reality" (XR) capabilities mature into the mainstream.

However, hosting AR applications in a browser is only part of the challenge. Imagine a scenario where people are walking through a shared common area, such as a museum or an airport. A user might run a navigation app to direct them to their destination. In the background, an airline loyalty app could suggest the closest lounges or ticket counters. In parallel, a user could load personalized food recommendation apps or use a friend finder app to connect with colleagues on their way to the same event, etc. If you extend this concept to defense or industrial IoT scenarios, we can imagine examples of how it would be advantageous to paint digital information from multiple sources together in a single AR fabric for increased situational awareness. The hope of being able to combine users, applications, and sensing data into a uniform representation has fueled much of the excitement behind MetaVerse concepts, Digital Twins, and the AR Cloud [9, 16].

Unlike traditional Web design workflows, collaborative XR applications pose new systems challenges for delivering interactive content at scale. Web frameworks are often structured around representational state transfer architectures (REST) that are not ideally suited for XR environments where you have a set of large models

and many small, low-latency interactions. This becomes more apparent when application state needs to be maintained across multiple users and applications. In Web applications like online games, which require low-latency shared state, system designers often implement one-off solutions with building blocks like WebRTC and WebSocket-based frameworks. We argue that future XR systems will require a more structured, safe, secure, and scalable infrastructure backing (a distributed operating system of sorts) to manage and host network connected environments.

There are three main framework requirements for collaborative Web XR applications. First, the system needs to support low-latency, consistent interaction and graphical updates across a scene. If a user clicks on a button or an application triggers an animation in response, these messages should be distributed in a timely manner to provide a responsive and consistent user experience. Second, to support multitenancy with users running their own applications, we need mechanisms that arbitrate access control and the ability to sandbox and isolate programs. From a systems perspective, this should strike the right balance between expressiveness of access control rights while not bogging down the messaging system with constant permissions checks. Finally, we need mechanisms allowing applications to be launched on targets that optimize objective functions like reducing latency. In Web systems, this usually means running custom code on the client or at the server. Given how locality significantly impacts latency, we require a runtime management system that is able to capture the state of the system resources and their interconnects and dispatch applications accordingly.

In this paper, we present the AR Edge Networking Architecture (ARENA), which is a framework designed to both simplify and scale collaborative multi-user XR applications. It simplifies programming applications where there is a mix of virtual and physical systems, where developers usually have to manually place computation and data storage across interconnected components. This allows developers to easily host multiple applications that interact with users and other agents (like sensors, actuators, or digital interfaces) in an immersive 3D environment. In the same way that a desktop operating system can run multiple applications, ARENA apps can start and exit dynamically in a hot-pluggable manner anywhere on the network. It has built-in support for geographic content lookup and mechanisms for accurate device localization from a number of different types of localization systems (Ultra-wideband Ranging Radios, Outside-in Optical Trackers, Optical Tags, etc). The graphical content is delivered using the A-Frame virtual reality framework [25] for execution on WebXR enabled browsers [46]. The use of modern XR Web technologies makes content accessible from tablets, phones, headsets, and desktop browsers. Users can view a scene in VR using a VR headset or within a 2D projection of a 3D environment in a desktop browser window. The same 3D content is available in AR, and ARENA provides mechanisms to simplify camera pose registration (see Section 3.2). Some example projects built using ARENA are shown in Figure 1 (more examples later in Section 4).

Core to our goal of scalability, ARENA organizes content into a set of *Scenes* that are hosted by local servers that form a *Realm*. Realms host services including web servers for static content, a Publish-Subscribe messaging bus for real-time data distribution, and a resource manager that monitors and dispatches applications. They provide natural spatial and/or organizational boundaries for those who want to host their own servers. A Scene is an abstraction that contains a group of related virtual assets like 3D objects, configuration parameters, and applications with shared end-points that allow user interactions. Scenes group together applications and users in a physical and/or virtual environment, and multiple Scenes can be overlaid simultaneously. For example, in our earlier public space scenario, there might be Scenes for each application. All live interactions in ARENA, like graphical updates, sounds, or I/O events are sent as commands over the Realm's common PubSub bus. ARENA also provides a Persistent Data Store that tracks the latest state of any objects marked as persistent that can be retrieved by clients first joining a Scene. A token-based user management system provides users with access tokens that define their access within the Scene.

Based on our set of collaborative XR framework requirements, we first evaluate ARENA's viability as an XR programming environment in terms of client-side performance as well as the ability for the backend to scale. We benchmark frame rate as a function of increased graphical complexity on a variety of modern platforms. We also evaluate the impact of load on interaction latency and show an average end-to-end response time for network-connected applications below 20ms on local wireless networks and above 40ms in the cloud. ARENA's topic structure layout was designed to scale with PubSub broker clustering, and it uses several extensions for capturing network graphs and provides federated access control. We show the ability for Scenes and Realms to isolate and scale environments with Scenes supporting hundreds of users within a Realm, and a Realm supports tens of thousands of Scenes. We also discuss how different organizations can federate Realms to support Internet-scale interactions. ARENA is currently running at multiple universities, each with a Realm that can share Scenes as needed. On the backend, we present several micro-benchmarks addressing security runtime overhead, end-to-end application latency across a number of network interfaces, and typical CPU, memory, and bandwidth profiles as application complexity and number of users increase on the system. Finally, we discuss several example applications, supporting tools, and an example Python Application Programming Interface (API) that can be used for writing programs.
In summary, our main contributions are:

1. The design of a framework for developing scalable collaborative Web XR applications including: (1) a geospatial lookup service, (2) a persistent data store, (3) an authentication and access control system, (4) a Web client front-end, and (5) a scalable PubSub architecture.
2. A distributed runtime that hosts network-connected applications near users and/or data sources.
3. An evaluation of client performance, security overhead, and scalability of our framework.
4. An open-source implementation with a discussion of various example projects that we believe could support and amplify much of the related work in collaborative interfaces.

2 Related Work

In this section, we review four main categories of related work: (i) Browser support for XR, (ii) tools to facilitate development and authoring of XR applications, (iii) previous approaches for scaling large virtual environments, and the Publish/Subscribe (PubSub) model and finally (iv) tools developed for XR collaboration.

2.1 AR Browsers

Early work on AR Browsers started in the late 90s through the early 2000s [7, 11, 14]. Argon [20] was perhaps the most notable example of a mobile AR browser, proposing the standardization of a platform to support XR applications and freeing the developer from many underlying AR technical details. It introduced the idea of content channels that can be displayed concurrently, something our work also supports through ARENA Scenes. Today, thanks to WebXR [46], we have access to standard APIs, allowing us to adapt, with minimal changes, to different display capabilities and experiences. ARENA benefits from the Web application model, including scalability, no installation, and a large developer base (consequently, many projects to draw from). Above the browser layer, we leverage A-Frame [25], an Entity-Component-System (ECS) architecture used to facilitate the development of primarily VR applications, but that we also use in AR. A-Frame is built on three.js [40], a 3D library which uses WebGL [33] to render content.

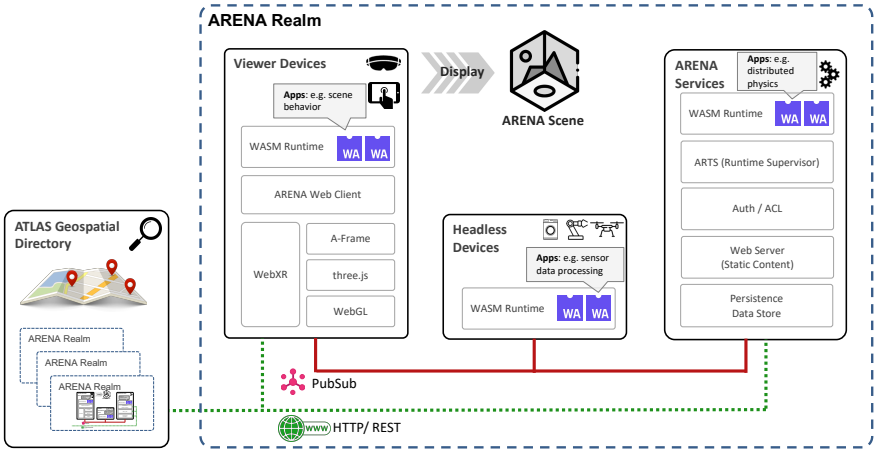


Figure 2: ARENA Overview. Realms represent a geographically distinct set of resources that coordinate over a common message bus.

Currently, WebXR-capable browsers are available in platforms such as Pixel 3, Pixel 4, and many other ARCore-enabled devices [28], Oculus Quest [37], or Microsoft Hololens [35]. We imagine the number of WebXR-supported platforms will continue to grow.

2.2 Native Development and Authoring

ARKit and ARCore [24, 29], from Apple and Google respectively, ease the task of creating experiences for iOS and Android devices (many headsets on the market are Android devices). Adobe also recently introduced Aero [23] to create AR experiences. Nevertheless, a significant number of XR applications are created in platforms originally meant for 3D game development, such as Unity or the Unreal Engine [26, 41]. Unfortunately, these common development platforms often neglect interaction and collaboration between physically co-located or remote users. Other important services for AR applications, such as geo-referencing content or access control, are often *ad hoc* solutions left to the developer. ARENA proposes a unified platform, advancing solutions that game development platforms can also adopt. Beyond just the networking interconnect, game architectures assume a single entity controlling/developing each game and don't have to deal with the same (hot loadable) programmability and security issues that crop up in metaverse-style systems.

2.3 Server and Communication Scaling

Modifiable virtual worlds (MVEs), such as *Second Life* and *Minecraft*, and networked games share some attributes with our architecture and are relevant context for this work. We note, however, that one of our main goals is to support interaction between co-located users, something important for AR applications that require pockets of locality. This contrasts with MVEs and networked games, which aim to enable a single large virtual world with many networked users across different geographical areas.

Traditional networked games often exploit the fact that the game provider is a single author of the content, which makes it easier to do world partitioning or pre-computations of visibility and 3D models [6]. More applicable to our scenario (content not controlled centrally) is another common technique of performing interest management to optimize the flow of data [19]. Other examples include systems that deliver different levels of detail based on computed metrics (combining proximity, recent interaction, and other aspects) [2]. This later work ([2]) also proposes to predict user state based on previous observations (dead reckoning). Many networked gaming systems, including popular MVEs such as *Minecraft*, use replicated, non-communicating game instances, which do not scale past a few hundred users [45].

One fundamental piece of our architecture is the Publish/Subscribe (PubSub) model, which aids in streamlining interest management by creating a topic hierarchy that partitions and isolates locally interacting users. PubSub allows disseminating information between data producers (publishers) and data consumers (subscribers), where publishers forward their data through brokers. PubSub is a central piece of many IoT and cloud infrastructures, and we can find many popular systems today, such as Google Cloud's Pub/-Sub [30], Apache Kafka [15], and MQTT [18, 31]. Peer-to-peer alternatives [36] have some scaling advantages in that Scenes can be completely independent, but have drawbacks associated with access control and per-Scene capacity (often a fraction of what is possible in a centralized model).

Load balancing users and topics among brokers is a critical aspect in the design of a distributed PubSub architecture [5]. We leverage users' geographic locality to connect clients to the nearest broker cluster and load balance between the brokers inside a cluster. We also partition the PubSub hierarchy to manage traffic across brokers and clusters and evaluate its performance in Section 5.4.

2.4 XR Collaboration Tools

We have seen an explosion of social VR platforms including Rec Room [39], AltspaceVR [39], and even Web-based platforms such as Facebook Spaces [27], VRChat [42], and Mozilla Hubs [17]. While ARENA can be used for multi-user VR worlds, it was primarily designed for AR. Significant prior work looked into collaborative AR/VR tools for writing and sketching on whiteboards [10], authoring mechanical models [1, 49], information analysis [4], architectural discussions [12], sharing VR experiences with multi-modal displays [8], promoting cooperation amongst different organizations [21], and many more. We believe ARENA promotes the rapid development of similar tools by providing the underlying infrastructure for collaborative XR. We show examples of these types of applications built with ARENA in Section 4.

3 System Architecture

Figure 2 presents an overview of ARENA[1]. A directory service, called *Atlas*, allows users to find nearby content based on coarse location and then supports managing the data needed to link Scene content with the physical world (see 3.2). As users find local content, they are handed off to a *Realm*, which is a server (or group of servers) that hosts ARENA 3D content and services. Channeling interactions through local/nearby Realms helps to improve latency-sensitive interactions. Realms connect hardware components like

[1]See documentation and source links at `https://arenaxr.org/`

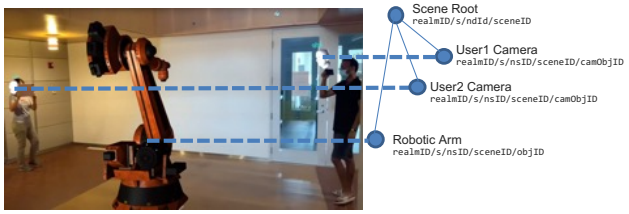


Figure 3: Objects in an ARENA Scene are implicitly networked over a PubSub bus, where each object is controlled by a topic end-point. The topic hierarchy is partitioned by Realm (`RealmID`), owner name space (`nsID`), Scenes (`sceneID`) and objects within the Scene (`camObjID` or `objID`).

viewing devices, such as headsets, mobile phones, or tablets, and other *headless devices* embedded in the environment (e.g., cameras and other sensors used for localization and environment awareness). Realms also include a set of *ARENA services* (message bus, content server, persistence, runtime manager) to support devices in that geographical area. Most services expose REST APIs to, for example, query current state or permissions, or create access tokens.

User devices connected to ARENA can not only show 3D content, but also host hot-pluggable applications. We created a common runtime to support sandboxed code launched from any connected target (see Section 3.3). We leverage modern WebXR-capable browsers to support diverse platforms and rendering capabilities, and the browser software in Figure 2 also depicts several existing frameworks we used: A-Frame [25], three.js [40], and WebGL [33]).

3.1 ARENA Scenes

ARENA Scenes include 3D content, configuration parameters, applications with shared end-points that allow user interactions, and information about markers that might serve as location anchors to the Scene. Scenes exist within a tree-like hierarchy with configurable access control and are often attached to a physical location. Using a Web analogy, the Realm is like a (local) webserver and the Scene is like a particular Web application at a URL end-point.

Scene Objects: ARENA Scenes are a collection of Entities to which Components can be attached, following A-Frame's Entity-Component-System (ECS) architecture [25]. We support the majority of A-Frame's primitives (e.g., geometries like boxes, circles, spheres) and components (attributes that can be attached to objects, such as position, rotation, material, sound). We also added ARENA-specific components for AR markers, programs, networked events, and options. All ARENA objects have well-defined schemas, which are the basis for the over-the-wire message format shown in Figure 4b and are transmitted over the PubSub. All messages have an `object_id`, a `type`, and an `action` (create, update, delete). Attributes in `data` are the object-specific attributes and components. The example shows a box geometry with `depth`, `height`, and `width` attributes, `position` and `rotation` components, and an ARENA-specific AR Marker component. Figure 4a shows a Web interface developed to make simple edits to a Scene, with a typical Scene object list, including Scene options, a program, lights, and several GLTF models. We imagine more advanced interfaces could be developed to create Scenes, and show later (Section 4) an example of an application for interactive 3D authoring.

Scene Loading: Scenes are loaded akin to Web applications within a Web browser. However, unlike most standard Web browsers, it is possible to simultaneously view a composition of multiple Scenes without switching between tabs. In XR, a user might have access to one or more Scenes in the same physical area that can be layered within an XR browser session. When the Scene is loaded, its current state is fetched from a data store service that tracks the persisted state of the Scene (see Section 3.4). Compositing Scenes together in a single view approximates how people naturally interact with the physical environment (as opposed to manually switching tabs).

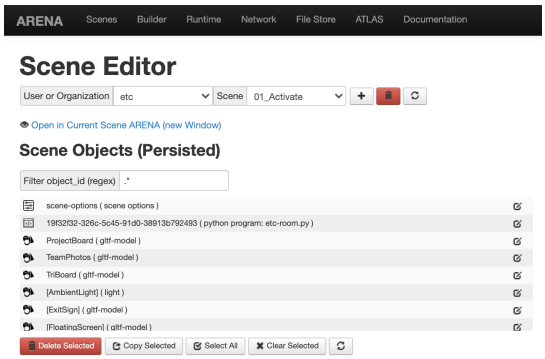
Real-time Updates: Once loaded, each of the 3D assets in a Scene are then updated in real-time over the Realm's local PubSub bus. For example, if an application changes the color of a cube, this would be captured in a message over the bus. Figure 3 exemplifies how a 3D Scene is represented. Each object in a Scene is managed by a topic end-point on the PubSub bus, making them "implicitly" networked. When a user moves their camera or clicks on an object, these updates and events are transmitted as messages. This network transparency allows any number of applications and users running from different devices to interact seamlessly within the 3D environment. As shown in Figure 3, users can see an avatar representation of other users in AR/VR as their camera pose is being published.

Access Control: Each user is given a token which defines read-/write access to topics within the Scene structure. This mechanism allows for very granular control over the objects in a Scene. For example, we can make certain objects invisible to a user by not granting read access to that particular object in the PubSub structure. For simplicity, we currently use Scenes as our basic unit of access control. We have defined two basic roles for users in a Scene: editor and viewer, and Section 3.4 describes the access control and authorization service.

3.2 Anchoring to Reality

ARENA provides several mechanisms to help streamline the management and sharing of anchor data as well as simplifying the process of combining multiple tracking technologies into a uniform coordinate system. As previously mentioned, Scenes can be registered and discovered by the Atlas service. Atlas operates in a hierarchical manner much like the Internet's Domain Name Service (DNS), but using a mixture of GPS coordinates, UUIDs, and Scenes instead of domain names. UUID markers can be embedded into QR codes, BLE beacons or other digital markers (WiFi, LTE tower, etc). Atlas can also provide absolute and/or local coordinates for markers that are associated with scenes. For example, a user could scan a QR code or read a BLE beacon which provides a UUID that maps to a GPS coordinate along with any Scenes containing that GPS coordinate. Atlas stores a GPS location for each Scene along with a 3D bounding polygon. The GPS location is typically assigned to the origin of the Scene's local coordinate system. A user can perform followup queries to Atlas for assets that fall within each Scene. For example, a Scene might contain a number of AprilTags (low bit-density tracking markers [47]) that have GPS coordinates as well as local coordinates referenced from the Scene's origin. It is worth noting that a Scene's address can be used to form a URL for virtual environments that have no physical location.

Since Atlas is a public facing entity that needs administrative management, ARENA also supports the ability to store location data within a Scene by attaching real-world properties to objects. Figure 4b shows an example of how a box object in a scene can have an AR marker property attached to it. This simplifies the common case where a developer builds a local Scene and wants a quick way to manage beacon data annotations for localization systems. AR markers can be set as 'static' or 'dynamic' to determine if clients should use them for relocalization or if clients should provide location information for the tag. While we wait for WebXR anchor support in released browsers [32], our current client can decode AprilTags [47] in browsers that allow camera access (e.g., Mozilla WebXR Viewer). If the client decodes a static tag, it uses the location data to compute the pose of the device's camera. If the tag is dynamic (and the client has a confident fix on its location), it publishes its estimate of the tag location to that object. In this way, multiple users in a space can update and share the location of dynamic tags. Since these tags are attached to objects, this naturally updates object positions and is



(a) Scene builder Web interface.

```
{ "object_id": "abox",
  "persist": true,
  "type": "object",
  "action": "create",
  "data": {
    "object_type": "box",
    "depth": 1,
    "height": 1,
    "width": 1,
    "position": { "x": 1, "y": 1, "z": 1},
    "rotation": { "x": 0, "y": 0, "z": 0},
    "armarker": {
      "lat": 40.4432,
      "lon": 79.9428,
      "markerid": "1",
      "markertype": "apriltag_36h11",
      "size": 150,
      "ele": 200
    }
  } }
```

(b) Message and Object definition.

Figure 4: Simple Scene builder and ARENA message/object definition example.

```
# Import arena python library
from arena import *
# Init library, connect to scene
scene = Scene(host="arenaxr.org", scene="example")
@scene.run_once # run once; @scene.run_forever(interval_ms) runs periodically
def main():
    # Create models; Moon model is a child of the Earth
    earth = GLTF(object_id="model-earth", position=(0, 0.1, 0),
     scale=(10, 10, 10), url="models/Earth.glb")
    moon = GLTF(object_id="model-moon", position=(0, 0.05, 0.6),
     scale=(0.05, 0.05, 0.05), url="models/Moon.glb", parent="model-earth")
    # Add models to scene
    scene.add_object(earth)
    scene.add_object(moon)
    # Define animation; Earth (and Moon, as it is a child) rotate together
    scene.update_object(
        earth,
        animation=Animation(
            property="rotation",
            end=(0,360,0),
            loop=True,
            dur=20000,
            easing="linear") )
# Start tasks
scene.run_tasks()
```

Figure 5: Sample Python script to create a Scene with Earth and Moon models rotating together.

reflected across all users and programs. User cameras and rigs can also be updated by external tracking systems. For example, we have a Python agent that converts OptiTrack [38] motion capture objects into ARENA position updates. If you target those outputs to specific user cameras or objects, they are automatically localized within the scene. This seamlessly works side-by-side with devices that use optical tags or even UWB localization. We have a number of helper applications that leverage different localization systems to help build tag maps (e.g., OptiTrack can be used to calibrate AprilTags within a shared space). Again, this is an example where the implicitly networked nature of all objects dramatically simplifies merging data from multiple sensing modalities.

3.3 Application Runtime

ARENA applications are compiled into WebAssembly (WASM), an open standard that defines a portable binary-code format for executable programs, currently supported by all major Web browsers. WASM programs are run in a secure sandbox and have been gaining traction outside of the browser as a lightweight and secure option for serverless-style computing [3, 43]. There are compilers for many languages that target WASM.

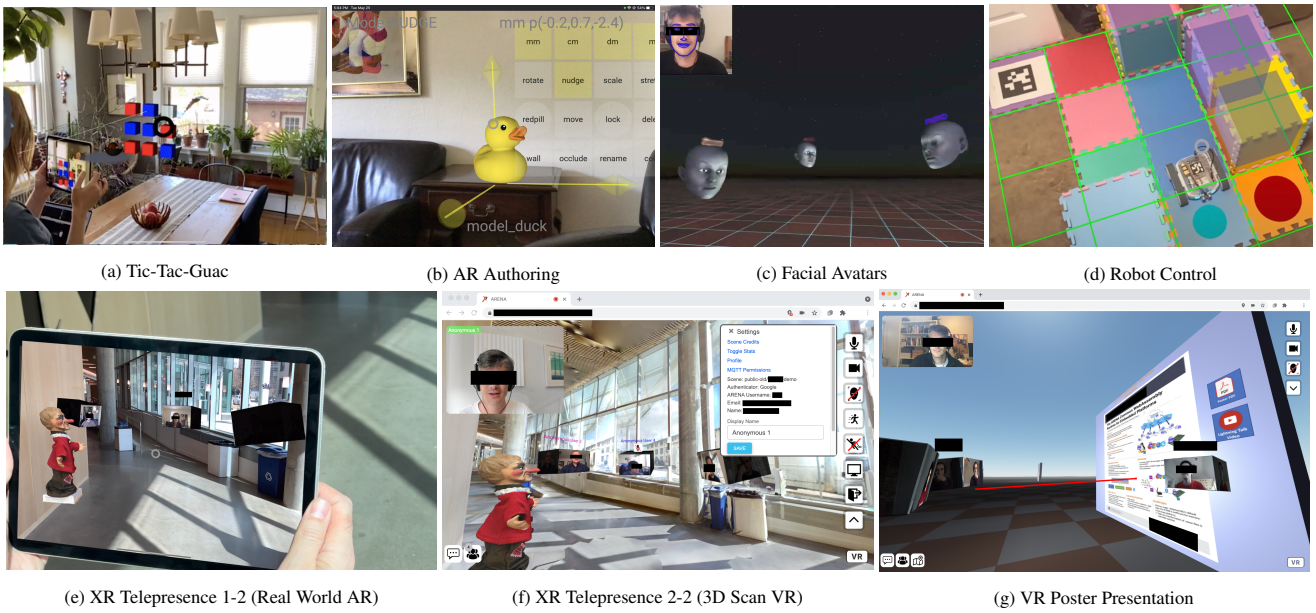
ARENA includes a WASM runtime environment for browser-capable devices that leverages the already available browser infrastructure, whereas other headless compute elements run a standalone WASM runtime. We are currently developing WASM runtimes in both Linux-capable devices and even dispatch Ahead-of-Time (AOT) compiled WASM to microcontrollers[2]. Our WASM runtime accepts requests to execute programs, provides sandboxed execution with access to (also sandboxed) networked resources, and manages the WASM programs' lifetime, including live migration capabilities (i.e., context swap across devices). The WASM runtime provides a basis for agile programs that operate in the dynamic, distributed computing contexts we imagine for future XR applications. It is an enabler for ARENA applications that can span cloud, edge, and device platforms in a network transparent manner. We are also developing a program manager for Scenes, which we call *init3D*, to provide facilities to manage programs interactively from within Scenes. Section 3.4 describes the ARENA Runtime Supervisor (ARTS), which manages programs in a Realm.

We developed a Python library to facilitate the development of ARENA applications[2]. These applications can be sandboxed in the WASM runtime, currently with limited library support due to the still immature support for Python in WASM toolchains. We believe the Python library provides a very accessible development option for ARENA applications. Our current API allows us to create and update objects in a Scene, define animations, and set up callbacks on events and timers. The library provides a scheduler and a design pattern familiar to game developers, which includes decorators to create one-shot, periodic, and delayed (start after a given time) tasks. Any entity represented in Python is automatically updated upon arrival of network messages, and we provide calls to load any preexisting Scene content upon startup. A simple illustrative Python script that loads GLTF models of the Earth and Moon and rotates them together is shown in Figure 5.

3.4 Services

PubSub Message Bus: The message bus is supported by a MQTT Mosquitto broker [18], modified to keep track of connected clients and data flows. This is organized into a graph that is available to users and, more importantly, to the runtime supervisor, ARTS (see the following paragraph). The broker is also configured with a JWT plugin that implements the PubSub ACL on the topic structure (more details later in this section), and we use Mosquitto's bridging to create clusters of brokers as detailed and evaluated in Section 5.4.
ARENA Runtime Supervisor (ARTS): By leveraging the WASM runtime and resource monitoring integrated into the PubSub bus, the ARENA Runtime Supervisor (ARTS) manages the heterogeneous compute resources of an ARENA realm[2]. ARENA Scenes include program objects that specify how a program is started. As a user loads an ARENA Scene, program objects originate requests to ARTS, which, in turn, will forward these requests to one of the available runtimes that previously indicated its availability and compute resources. ARTS currently implements simple scheduling policies, such as (runtime) round-robin or least loaded (runtime), but it will intelligently respond to available networking and compute resources, quality-of-service, and security policies as we develop its capabilities. While compute placement is not the focus of this work, we

[2] https://github.com/conix-center/



(a) Tic-Tac-Guac (b) AR Authoring (c) Facial Avatars (d) Robot Control

(e) XR Telepresence 1-2 (Real World AR) (f) XR Telepresence 2-2 (3D Scan VR) (g) VR Poster Presentation

Figure 6: **ARENA Applications.** Figure **6a** shows the AR POV of User A playing a game (center) with User B using an AR tablet (left). Figure **6b** shows the AR POV of User A, editing a 3D model (center) with a movable control panel (right). Figure **6c** shows the VR avatar/audio conference POV of User A (top-left, landmarks in blue) with Users B/C/D as translated facial avatars (center). Figure **6d** shows the AR POV of a sensing robot and mini-AprilTag (center) with a Scene anchor AprilTag (top-left). A telepresence conference is shown from User A's AR tablet POV in Figure **6e** and also from User A's VR POV in Figure **6f**. Figure **6g** shows the VR POV of User A (top-left) in User B's (right) poster session with red laser pointer (center) and Users C/D (left). Some images trimmed for space.

provide a preliminary study of its impact in Section 5.

Persistence Data Store: Once a user connects to a Realm and loads a particular Scene, a browser is given all of the 3D objects within the scene. This content is initially requested from a Persistence Data Store that tracks the latest state of any persistent objects (not all objects need to be persistent).

Authentication and Access Control: The root of trust for an ARENA Realm derives from the Access Control List (ACL) permissions managed by the Authentication and Access Control service. Users are authenticated using OAuth, and the service emits JSON Web Tokens (JWT) based on permissions for which Scene users control or grant control as an ACL. PubSub brokers and other services (e.g., the persistence datastore) use JWT to enforce access control. PubSub brokers accept and validate these tokens and use them to allow/disallow publish or subscribe access. We use the PubSub topic structure to sculpt and whitelist which 3D objects, chat, runtime, and other communications bind this ACL to the user's JWT. To prevent clients from elevating their privileges by publishing malicious messages on the topics they are allowed to publish, we perform client-side checks on message reception and discard invalid messages. Section 5.1 shows that security overhead is minimal.

Web Server: Static content such as 3D models, images, sound, and the basic Scene skeleton are fetched from a Web server. ARENA Scenes take advantage of this basic Web infrastructure to facilitate the delivery and access to content, which also allows us to leverage common Web scaling techniques like CDNs.

4 Applications

Over the course of designing ARENA, we had several developers from a variety of backgrounds build an assortment of applications. These programmers ranged from graphics and animation experts and students learning about XR design to low-level computer systems programmers with less experience related to graphical systems. During this time, we iterated on our programming API to help target common design patterns. Initially, our ARENA library was a thin

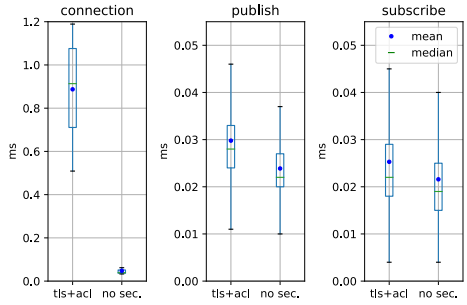


Figure 7: Connection, publish, and subscribe security (TLS+ACL) overhead, compared to a broker with no security enabled. Box plot shows minimum, Q1, median, Q3, and maximum.

layer above the PubSub message protocol that we implemented in C and in Python. Over time, we moved more toward an ECS architecture with a Python library that supported the most common loop and callback functions. In the following section, we describe a few applications developed using ARENA and highlight trade-offs compared to more traditional systems.

Large-scale VR Chat: As an extension of the collaborative user-presence feature of ARENA, we integrated the open source Jitsi [22] videoconferencing stack into our main JavaScript Web stack to enable user video chats in VR (Figure 6g). We have experimented with events having more than 80 users in a single Scene. User video feeds are rendered on 3D cubes, and their audio feed volumes are automatically adjusted based on camera movement and stereo position in 3D. We have hosted multiple VR conferences, poster sessions, and interactive 3D demonstrations using VR chat.

AR Authoring: The AR Builder (ARB) tool (Figure 6b) was written using the ARENA Python library to provide interactive 3D authoring. ARB reacts to MQTT-published user camera positions to render a rotating 3D control panel of buttons to create, delete, rotate, move,

	Video (30 FPS @ 1080p)		Audio (2 chan, 16bit, 44KHz)		Object Updates (10Hz)		User Motion (10Hz)	
	Raw	H.264	Raw	128bit mp3	Raw	80% RLE	Raw	80% RLE
Throughput (Bytes/Second)	18,662,400	889,856	176,400	16,384	1,250	250	1,250 * n^2	250 * n^2
# Streams Norm. to Video Stream	0.005	1	5	54	711	3,559	26	105
# Streams Norm. to Raw Video	1	209	1,057	11,390	149,299	746,496	386	1,531
# of Streams (Users) at 10Mbit/s	0.007	1.4	7	76	1,000	5,000	31	125
# of Streams (Users) at 100Mbit/s	0.067	14.05	70	762	10,000	50,000	100	396

Table 1: Typical messaging data rates compared to Audio and Video streaming for graphical updates and user camera updates.

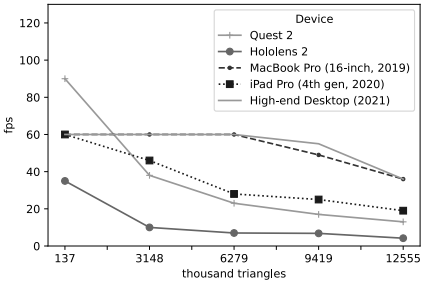


Figure 8: Frames per second (FPS) across five different devices as we increase the number of triangles in the 3D scene.

stylize, stretch, and manipulate 3D models and primitives in AR. It can be launched in any scene for AR or VR to place models in physical and virtual spaces.

MR Games: Tic-Tac-Guac (Figure 6a) presents a 9-block Tic-Tac-Toe game where users in MR can click a block on their turn to change a block either blue or red. The game ends with all blocks falling and bouncing off the ground plane based on our collision and physics engines (also network attached agents).

Facial Avatar: We added a facial landmark (Figure 6c) detection algorithm [13] and a landmark-to-head pose estimator. Both of these are compiled to WebAssembly and run in the browser in an ARENA Scene. The user's facial landmarks and head pose are published across our MQTT network, and can be received by an ARENA Python program. The Python program employs a custom-trained machine learning model which generates an emotion vector using facial landmarks received. The program publishes a rigged, 3D avatar head that mirrors the user's mouth movements and emotions. For this, Jitsi is disabled, allowing for live facial and mouth movements while speaking, without the need to share video.

XR Sensing Robot: Figure 6d shows a robot click-and-control application allowing users to guide a small robot. We used the infrared and color sensing robot coding platform Sphero RVR, mounted with a Raspberry Pi (RPi). We wrote an application for Sphero integrating our Python library to give the robot a planned direction to travel, which it then reconciles with real-world sensors and reports its observation of obstacles in real-time to the ARENA scene. There is a global AprilTag to localize the tablet and a small dynamic AprilTag on the robot to give it outside-in location updates. We ran the application on the RPi to demonstrate self-contained compute, infusing physical sense with virtual sense.

XR Telepresence: We leveraged the ARENA network and Jitsi to create a Mixed-Reality Telepresence conference (Figures 6e, 6f). For this we scanned a campus conference building with a Matterport [34] scanner and uploaded the resulting model to an ARENA Scene. Three VR users entered the 3D Scene as video-cube avatars. Then, a user physically present in the same conference building used an AR tablet to enter the Scene, anchored in place with an AprilTag. The AR user sees and hears the three remote VR users, and the remote VR users see and hear the other VR and AR users in a VR scanned version of the building. Again, this highlights the simplicity of creating cross-domain interactive network applications.

5 Evaluation

To put the network bandwidth involved in our collaborative XR framework into context, we compared typical audio and video streaming with graphical and user camera updates in an ARENA Scene as shown in Table 1. Camera movements must be delivered to all users in a Scene, and thus grow quadratically. For comparison, a standard Netflix video stream consumes the same bandwidth as over 700 objects updating at 10Hz each. Later in this section, we profile latency, bandwidth, and memory in more detail as the number of users increases. The remainder of this section will benchmark the overhead of the security mechanism introduced in ARENA's PubSub, browser rendering performance, end-to-end latency, and PubSub broker clustering.

5.1 Security Overhead

To evaluate the impact of supporting secure channels and PubSub access control (see access control and authorization in Section 3.4), we instrumented our MQTT brokers and profiled the time to handle a client connection, a message publication, and a topic subscription. The resulting delays compared to a broker with no security enabled are shown in Figure 7. Connection time is significantly higher, mostly due to the transport-layer security (TLS) connection setup. ACL-related checks increase publish and subscribe mean execution time by 24% and 17%. Relative to communication and application processing delays, on the order of milliseconds, these are very small. Later in this section, we see that brokers can be scaled horizontally, which also helps amortize these overheads.

We also profiled the browser client code to measure the relative overhead of client-side message parsing and validation, including security checks to ensure users do not abuse their permissions in a Scene. We profiled an application that created a Scene with more than 12.5 thousand triangles, updating objects through PubSub at 70Hz, and observed that parsing and validation accounted for about 2% of the total execution time of the code.

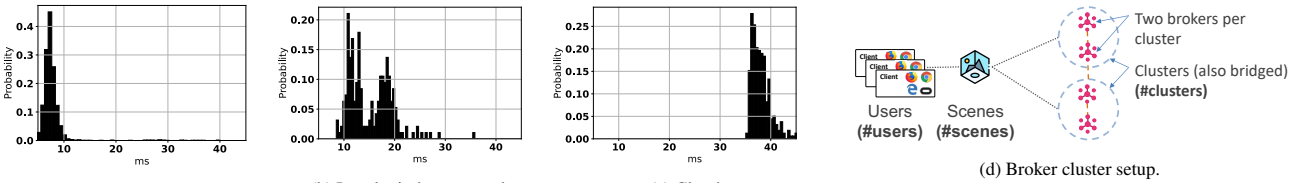
5.2 Rendering Performance

While improving rendering performance is not the aim of this work, it is an important metric to show the feasibility of using current WebXR-enabled browsers in practice. To benchmark the rendering capabilities of modern browsers, we collected the frames per second (FPS) across five different devices as we increase the complexity of the scene rendered, as shown in Figure 8. We see that it performs surprisingly well even on low-compute platforms like Hololens 2. It is worth noting that WebXR is already compatible with some edge rendering systems like AirLink on the Oculus Quest 2.

5.3 End-to-End Latency

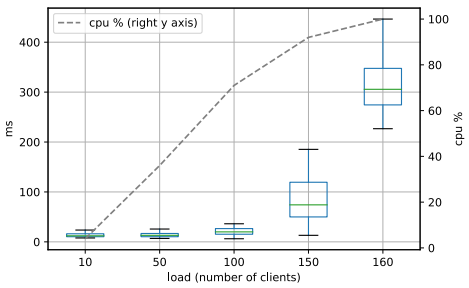
This section will start by showing the effect of compute placement under light load on a single broker given different network interconnects. We then show how the latency changes as we increase broker load. Finally, we show how using multiple brokers can isolate Scenes and scale to support many users.

Compute Placement: Let us look at the achievable latency under three basic placement scenarios of interactive agents controlling the behavior of a networked scene: (i) on a node connected across a local wired network, (ii) across a wireless network, and (iii) placed

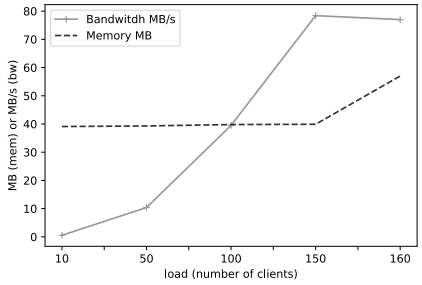


(a) Local wired network. (b) Local wireless network. (c) Cloud. (d) Broker cluster setup.

Figure 9: End-to-end latency across three different placements, and broker cluster experimental setup.



(a) End-to-end latency and CPU utilization vs broker load.



(b) Bandwidth and memory vs broker load.

Figure 10: Latency, bandwidth and memory as broker load increases.

in the cloud. We created an agent using the ARENA Python library that reacted to user inputs in a Scene. The Python agent subscribes to the object's events to receive click events, which trigger a location update of an object in the Scene (this update is published to the corresponding topic providing access to the object).

We measured the end-to-end latency from the click event to when the update is received back to the client, shown in Figure 9. While the placement of the Python agent impacts the end-to-end delay, observe that even when placed at the cloud (nearby AWS), the end-to-end delay is often under 40 ms, which can support many interactive applications. Applications requiring very low latency (e.g., a networked physics engine) can be supported by placing the apps/services near users, and we can observe this in Figure 9a, with latencies under 10 ms. Our architecture was designed to allow such compute/delay trade-offs, and the placement of agents that control our networked Scenes is the subject of further research.

Broker Load: We now consider the load on a single broker and its impact on the end-to-end latency of networked interactive agents. We placed a similar Python agent (from the previous paragraph) on the local network and measured the end-to-end delay as we increased the broker load by adding users to the Scene. Each user placed in the Scene was configured to send position updates (simulating user movement) at 10Hz and subscribed to updates from everyone else. The resulting end-to-end delay, memory, and bandwidth as a function of the load is presented in Figure 10. We can see the broker is at 92% CPU utilization with 150 users, and, when we get to around 160 users, the broker is CPU-bound, maintaining a throughput of 80

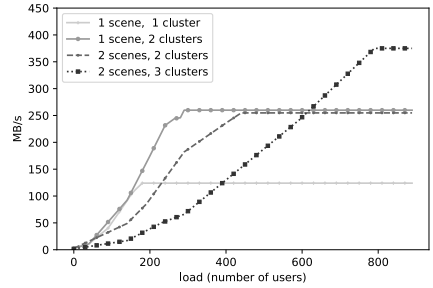


Figure 11: Broker cluster throughput. All clusters have two brokers.

MB/s, while memory starts to increase due to queuing.

5.4 Broker Clustering

We use MQTT brokers' clustering capabilities to scale PubSub by carefully managing both where clients connect and the traffic across brokers and between clusters. Our current setup uses Mosquitto [18], which is designed to use a single CPU but can scale to multiple CPUs by bridging brokers together (even on a single machine). In our clustering configuration, depicted in Figure 9d, we bridge brokers to form clusters inside a machine such that can fully utilize the available CPUs and network bandwidth. To isolate traffic between brokers and clusters, we make sure that Scenes and associated event traffic are disjoint in the PubSub topic tree. Further scaling, isolation, and security can be achieved with multiple Realms.

Figure 10 (previous section) shows that a single broker becomes CPU-bound with 160 users. Thus, increasing the cluster to two brokers will allow reaching the network capacity available in the same machine (this is the scenario: 1 Scene, 1 cluster, shown in Figure 11). Scaling further requires more network capacity, which we can achieve by, for example, adding more clusters or splitting users across scenes. We can see this effect in Figure 11, showing the system throughput for four different variations of the number of Scenes and clusters (all clusters have two brokers). Observe the difference from a single broker (Figure 10) to a cluster with two brokers. With a two-broker cluster, we are bound to 125 MB/s (the maximum bandwidth available) which is reached at 180 users. With two clusters, we are bound to 2x125 MB/s, and isolation between scenes allows around 40% more users. The scaling of multiple clusters is further shown with three clusters, which reaches 800 users at 3x125 MB/s.

6 Conclusion

This paper presents ARENA, a multi-user, collaborative framework for WebXR applications. ARENA simplifies development with the ability to hot-load content within an implicitly networked environment. Our design allows hosting and deploying applications based on the locality of users and resources within a scene, which helps reduce latency for interactive XR applications. We believe ARENA enables a new class of highly connected XR capabilities and opens future work related to compute placement, security, programming and authoring models, and perhaps even new XR application ecosystems.

ACKNOWLEDGMENTS

This work was supported in part by the CONIX Research Center, one of six centers in JUMP, a Semiconductor Research Corporation (SRC) program sponsored by DARPA. This work would not have been possible without many collaborations across the CONIX Center. The authors would also like to thank John Balash and CMU's Entertainment Technology Center (ETC) for their help with designing, debugging and building ARENA applications.

REFERENCES

[1] O. Bergig, N. Hagbi, J. El-Sana, and M. Billinghurst. In-place 3d sketching for authoring and augmenting mechanical systems. In *2009 8th IEEE International Symposium on Mixed and Augmented Reality*, pp. 87–94, 2009. doi: 10.1109/ISMAR.2009.5336490

[2] A. Bharambe, J. R. Douceur, J. R. Lorch, T. Moscibroda, J. Pang, S. Seshan, and X. Zhuang. Donnybrook: Enabling large-scale, high-speed, peer-to-peer games. In *Proceedings of the ACM SIGCOMM 2008 Conference on Data Communication*, SIGCOMM '08, p. 389–400. Association for Computing Machinery, New York, NY, USA, 2008. doi: 10.1145/1402958.1403002

[3] D. Bryant. Webassembly outside the browser: A new foundation for pervasive computing. `https://icwe2020.webengineering.org/wp-content/uploads/2020/06/ICWE2020_keynote-David_Bryant.pdf`, 2020. Online. Accessed: May 2021.

[4] M. Cavallo, M. Dholakia, M. Havlena, K. Ocheltree, and M. Podlaseck. Dataspace: A reconfigurable hybrid reality environment for collaborative information analysis. In *2019 IEEE Conference on Virtual Reality and 3D User Interfaces (VR)*, pp. 145–153, 2019. doi: 10.1109/VR.2019.8797733

[5] C. Chen, H.-A. Jacobsen, and R. Vitenberg. Algorithms based on divide and conquer for topic-based publish/subscribe overlay design. *IEEE/ACM Transactions on Networking*, 24(1):422–436, 2016. doi: 10.1109/TNET.2014.2369346

[6] D. Cohen-Or, Y. Chrysanthou, C. Silva, and F. Durand. A survey of visibility for walkthrough applications. *IEEE Transactions on Visualization and Computer Graphics*, 9(3):412–431, 2003. doi: 10.1109/TVCG.2003.1207447

[7] S. Feiner, B. MacIntyre, T. Hollerer, and A. Webster. A touring machine: prototyping 3d mobile augmented reality systems for exploring the urban environment. In *Digest of Papers. First International Symposium on Wearable Computers*, pp. 74–81, 1997. doi: 10.1109/ISWC.1997.629922

[8] J. Gugenheimer, E. Stemasov, J. Frommel, and E. Rukzio. Sharevr: Enabling co-located experiences for virtual reality between hmd and non-hmd users. In *Proceedings of the 2017 CHI Conference on Human Factors in Computing Systems*, CHI '17, p. 4021–4033. Association for Computing Machinery, New York, NY, USA, 2017. doi: 10.1145/3025453.3025683

[9] C. Hackl. The metaverse is coming and it's a very big deal. `https://www.forbes.com/sites/cathyhackl/2020/07/05/the-metaverse-is-coming--its-a-very-big-deal`. Online. Accessed: May 2021.

[10] Z. He, R. Du, and K. Perlin. Collabovr: A reconfigurable framework for creative collaboration in virtual reality. In *2020 IEEE International Symposium on Mixed and Augmented Reality (ISMAR)*, pp. 542–554. IEEE, 2020.

[11] F. Hohl, U. Kubach, A. Leonhardi, K. Rothermel, and M. Schwehm. Next century challenges: Nexus—an open global infrastructure for spatial-aware applications. In *Proceedings of the 5th Annual ACM/IEEE International Conference on Mobile Computing and Networking*, MobiCom '99, p. 249–255. Association for Computing Machinery, New York, NY, USA, 1999. doi: 10.1145/313451.313549

[12] T.-W. Hsu, M.-H. Tsai, S. V. Babu, P.-H. Hsu, H.-M. Chang, W.-C. Lin, and J.-H. Chuang. Design and initial evaluation of a vr based immersive and interactive architectural design discussion system. In *2020 IEEE Conference on Virtual Reality and 3D User Interfaces (VR)*, pp. 363–371. IEEE, 2020.

[13] V. Kazemi and J. Sullivan. One millisecond face alignment with an ensemble of regression trees. *2014 IEEE Conference on Computer Vision and Pattern Recognition*, pp. 1867–1874, 2014.

[14] R. Kooper and B. MacIntyre. Browsing the real-world wide web: Maintaining awareness of virtual information in an ar information space. *International Journal of Human-Computer Interaction*, 16(3):425–446, 2003.

[15] J. Kreps, N. Narkhede, J. Rao, et al. Kafka: A distributed messaging system for log processing. In *Proceedings of the NetDB*, vol. 11, pp. 1–7, 2011.

[16] C. E. Lathan and G. Ling. Spatial Computing Could Be the Next Big Thing. `https://www.scientificamerican.com/article/spatial-computing-could-be-the-next-big-thing/`. Online. Accessed: May 2021.

[17] D. A. Le, B. Maclntyre, and J. Outlaw. Enhancing the experience of virtual conferences in social virtual environments. In *2020 IEEE Conference on Virtual Reality and 3D User Interfaces Abstracts and Workshops (VRW)*, pp. 485–494, 2020. doi: 10.1109/VRW50115.2020.00101

[18] R. A. Light. Mosquitto: server and client implementation of the mqtt protocol. *Journal of Open Source Software*, 2(13):265, 2017.

[19] E. S. Liu and G. K. Theodoropoulos. Interest management for distributed virtual environments: A survey. *ACM Comput. Surv.*, 46(4), Mar. 2014. doi: 10.1145/2535417

[20] B. MacIntyre, A. Hill, H. Rouzati, M. Gandy, and B. Davidson. The argon ar web browser and standards-based ar application environment. In *2011 10th IEEE International Symposium on Mixed and Augmented Reality*, pp. 65–74, 2011. doi: 10.1109/ISMAR.2011.6092371

[21] S. Nilsson, B. Johansson, and A. Jonsson. Using ar to support cross-organisational collaboration in dynamic tasks. In *2009 8th IEEE International Symposium on Mixed and Augmented Reality*, pp. 3–12, 2009. doi: 10.1109/ISMAR.2009.5336522

[22] 8x8 Inc. Free video conferencing software for web & mobile — jitsi. `http://jitsi.org`. Online. Accessed: May 2021.

[23] Adobe Website. Aero. `https://www.adobe.com/products/aero.html`. Online. Accessed: May 2021.

[24] Apple Inc. ARKit Website. `https://developer.apple.com/augmented-reality/`, May 2021. Online. Accessed: May 2021.

[25] Diego Marcos, D. McCurdy, K. Ngo, and Et al. A-Frame Framework (v1.0). `https://github.com/aframevr/aframe/`. Online. Accessed: May 2021.

[26] Epic Games. Unreal Engine Website. `https://www.unrealengine.com/en-US/`, May 2021. Online. Accessed: May 2021.

[27] Facebook Inc. Facebook Spaces. `https://www.facebook.com/spaces`. Online. Accessed: May 2021.

[28] Google Inc. ARCore supported devices. `https://developers.google.com/ar/discover/supported-devices`. Online. Accessed: May 2021.

[29] Google Inc. ARCore Website. `https://developers.google.com/ar`. Online. Accessed: May 2021.

[30] Google Inc. Google Cloud Pub/Sub. `https://cloud.google.com/pubsub/`. Online. Accessed: May 2021.

[31] IBM. MQTT V3.1 Protocol Specification. `https://public.dhe.ibm.com/software/dw/webservices/ws-mqtt/mqtt-v3r1.html`. Online. Accessed: May 2021.

[32] Immersive Web WG. WebXR Anchors Module. `https://immersive-web.github.io/anchors/`. Online. Accessed: May 2021.

[33] Khronos Group. WebGL 2.0 Specification. `https://www.khronos.org/registry/webgl/specs/latest/2.0/`, Oct 2020. Online. Accessed: May 2021.

[34] Matterport Inc. Capture, share, and collaborate the built world in immersive 3d. `https://matterport.com/`. Online. Accessed: May 2021.

[35] Microsoft. Hololens Website. `https://www.microsoft.com/en-us/hololens/`. Online. Accessed: May 2021.

[36] Networked A-Frame Developers. Networked A-Frame. https://github.com/networked-aframe, June 2021. Online. Accessed: May 2021.

[37] Oculus. Quest Website. `https://www.oculus.com/quest/`. Online.

Accessed: May 2021.

[38] Optitrack Inc. Optitrack Motion Capture Systems. https://optitrack.com/. Online. Accessed: May 2021.

[39] Rec Room Inc. Rec Room Website. https://recroom.com/. Online. Accessed: May 2021.

[40] Three.js Developers. Three.js Library. https://threejs.org/. Online. Accessed: May 2021.

[41] Unity Labs. Unity Website. https://unity.com/. Online. Accessed: May 2021.

[42] VRChat Inc. VRChat. https://hello.vrchat.com/. Online. Accessed: May 2021.

[43] WASM WG. Webassembly overview. https://webassembly.org/. Online. Accessed: May 2021.

[44] I. E. Sutherland. The ultimate display. In *Proceedings of the IFIP Congress*, pp. 506–508, 1965.

[45] J. van der Sar, J. Donkervliet, and A. Iosup. Yardstick: A benchmark for minecraft-like services. In *Proceedings of the 2019 ACM/SPEC International Conference on Performance Engineering*, ICPE '19, p. 243–253. Association for Computing Machinery, New York, NY, USA, 2019. doi: 10.1145/3297663.3310307

[46] W3C. WebXR Device API. https://immersive-web.github.io/webxr/. Online. Accessed: May 2021.

[47] J. Wang and E. Olson. Apriltag 2: Efficient and robust fiducial detection. In *2016 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 4193–4198. IEEE, 2016.

[48] W. Wang. *Understanding Augmented Reality and ARKit*, pp. 1–17. Apress, Berkeley, CA, 2018. doi: 10.1007/978-1-4842-4102-8_1

[49] C. Weichel, M. Lau, D. Kim, N. Villar, and H. W. Gellersen. Mixfab: A mixed-reality environment for personal fabrication. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, CHI '14, p. 3855–3864. Association for Computing Machinery, New York, NY, USA, 2014. doi: 10.1145/2556288.2557090