Characterization of Multi-User Augmented Reality over Cellular Networks

Kittipat Apicharttrisorn*, Bharath Balasubramanian†, Jiasi Chen*, Rajarajan Sivaraj‡, Yi-Zhen Tsai*,
Rittwik Jana†, Srikanth Krishnamurthy*, Tuyen Tran†, and Yu Zhou†
*University of California, Riverside, California, USA
{kapic001 | ytsai036}@ucr.edu, {jiasi | krish}@cs.ucr.edu
†AT&T Labs Research, Bedminster, New Jersey, USA
{bb536c | rj2124}@att.com, {tuyen | yuzhou}@research.att.com
‡AT&T Labs Research, San Ramon, California, USA
rs405h@att.com

***Abstract*—Augmented reality (AR) apps where multiple users interact within the same physical space are gaining in popularity (*e.g.,* shared AR mode in Pokemon Go, virtual graffiti in Google's Just a Line). However, multi-user AR apps running over the cellular network can experience very high end-to-end latencies (measured at 12.5 s median on a public LTE network). To characterize and understand the root causes of this problem, we perform a first-of-its-kind measurement study on both public LTE and industry LTE testbed for two popular multi-user AR applications, yielding several insights: (1) The radio access network (RAN) accounts for a significant fraction of the end-to-end latency (31.2%, or 3.9 s median), resulting in AR users experiencing high, variable delays when interacting with a common set of virtual objects in off-the-shelf AR apps; (2) AR network traffic is characterized by large intermittent spikes on a single uplink TCP connection, resulting in frequent TCP slow starts that can increase user-perceived latency; (3) Applying a common traffic management mechanism of cellular operators, QoS Class Identifiers (QCI), can help by reducing AR latency by 33% but impacts non-AR users. Based on these insights, we propose network-aware and network-agnostic AR design optimization solutions to intelligently adapt IP packet sizes and periodically provide information on uplink data availability, respectively. Our solutions help ramp up network performance, improving the end-to-end AR latency and goodput by ∼40-70%.**

***Index Terms*—Augmented reality, Mobile communication, Cross layer design, Radio access networks**

I. Introduction

Augmented reality (AR), with its premise of virtual objects integrated with our physical environment, promises new immersive experiences, and the market is forecast to reach 100 billion dollars by 2021 [1], [2]. AR applications such as navigation, entertainment (*e.g.,* Pokemon Go), and field service involve multiple users, co-located in the same shared outdoor environments, relying on low latency communication over the cellular network to obtain a consistent view of virtual objects. In this paper, we consider scenarios where multiple users are co-located in the same real physical space, and wish to view a common set of virtual objects. Our measurements of such off-the-shelf AR apps over cellular networks show that the user-perceived end-to-end AR latencies are extremely high. For example, Fig. 1 shows the CDF of the AR latencies of the Google CloudAnchor AR app [3], running on a 4G LTE production network of a Tier-I US cellular carrier at different locations and times of day (details in §IV). The results show a median 3.9 s and 12.5 s of aggregate radio access network (RAN) latency and end-to-end AR latency (as explained below), respectively. As latency is a key contributor to AR quality-of-experience (QoE) [4], a deeper understanding of the root causes of these high latencies is needed in order to improve AR's end-to-end performance over cellular networks, which is the key focus of our paper.

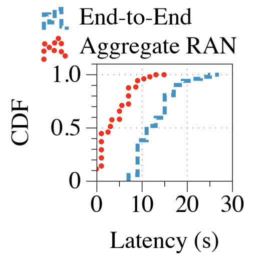


Fig. 1. Multi-User AR latency

In this context, end-to-end AR latency is the total time from when one User Equipment (UE) places a virtual object in the real world until when another UE can view the object on her screen. Aggregate RAN latency is defined as the subset of this time for over-the-air transmissions.

Why is AR different?: While the cellular network is relatively well equipped to handle traditional applications such as web and video, AR presents new challenges because of its unique application and communication characteristics, as discovered in this paper. In brief, AR differs from other multimedia applications such as video or 360° Virtual Reality (VR) streaming, short-form video uploads (*e.g.,* Snapchat or Instagram Stories) and video conferencing in the following key ways:

- *Lack of playback buffers and latency-sensitivity*: In video and 360° VR streaming, the length of the playback buffer, which caches the yet-to-be-played video chunks for the player, impacts streaming patterns and traffic burst periods. We observe that off-the-shelf AR apps do not continuously upload data, and they consume AR data holistically, unlike in video/VR, where the player consumes data frame-by-frame. Hence, AR data objects need to be delivered *quickly* to the AR device (user equipment, or UE), in order to avoid latency-based QoE issues. While video conferencing similarly lacks playback buffers and has tight latency deadlines, delayed frames can be skipped or played back quickly later, whereas delayed AR transmissions can lead to inconsistent user manipulations of the virtual objects (*e.g.,* user A touches a non-existent virtual object that has already been moved by user B).

- *Lack of application adaptation mechanisms*: Video and 360° VR streaming make use of adaptive bit rate (ABR) mechanisms, such as MPEG-DASH, which adapt the streaming resolutions of the video chunks to avoid video QoE issues. However, off-the-shelf AR currently does not use ABR, and cannot be modified to do so in closed source commercial AR systems. Additionally, even if such systems were transparent, it is unclear how such application-layer adaptations should be done in AR. (discussed in §V-C)
- *Uplink-heavy TCP traffic*: In multi-user gaming, the traffic is comprised of small uplink UDP [5] data (mainly from user movements/actions) with stringent latency requirements. For short-form video uploads (such as Instagram Stories or Snapchat), live video upload or conferencing, even though the traffic is uplink-heavy, latency tolerance is higher than AR [4]. Live or buffered video streaming apps such as YouTube use downlink QUIC [6] instead of TCP, and are more latency-tolerant than AR. In contrast, we observe that AR network traffic is different from all these, since it is uplink-heavy, TCP-based and latency-sensitive. Hence, TCP performance for the AR session is critical to its end-to-end latency and throughput, and we investigate its interactions with the RAN and AR in this work.

Contributions: Motivated by these unique characteristics of multi-user AR, we perform the first detailed experimental study across the application, IP, and RAN layers to characterize how the cellular network impacts AR applications. We provide crucial insights on cross-layer inter-dependencies involved in multi-user AR streaming. Existing work on AR either focuses on real object detection with cloud/edge support [7], [8], [1] over WiFi, or efficient device localization [9] while neglecting the communication aspects. Works involving multiple AR devices [10], [11], [12] focus on application layer performance, without quantifying the interactions between AR application, the cellular network, and cloud processing.

Our main contribution in this paper is a measurement-driven characterization of multi-user AR on both (i) public 4G LTE cellular carriers of a Tier-I US mobile network operator to capture realistic network, RF, and traffic conditions, and (ii) an experimental LTE industry testbed with fully-implemented RAN protocol stack and virtual EPC that allows us to vary network settings (such as cell load, radio bearer QoS class, etc.) for controlled testing, in order to quantify their impact on AR performance. We study widely-used AR apps in the market utilizing an off-the-shelf AR platform, Google ARCore [13]. This platform is broadly representative since it provides multi-user AR capabilities in Android devices and analogous APIs are provided by Apple ARKit [14], Microsoft Hololens [15], and Magic Leap [16] (see §III for methodology details).

Our measurement study leads to several insights:

- To quantify differences between multi-user AR and other multimedia applications, we compare the user-perceived latencies across these apps in §IV-A. Then, we provide a component-wise breakdown of the end-to-end (E2E) AR latency and show that the RAN accounts for ∼31.2% of the overall latency on average over public LTE (∼3.9s). This causes the AR devices to experience high delays when interacting with the virtual objects (§IV-B). We also characterize the performance of RAN optimization techniques such as QoS Class Index (QCI) adaptation to serve AR traffic. While QCI adaptation improves the E2E latency for AR users by ∼33%, it reduces the throughput of non-AR users by ∼31.6% (§IV-D).
- We show that AR traffic is bursty with large time gaps between successive bursts of uplink data (*e.g.,* 20s on average for CloudAnchor with burst sizes of ∼2.5 MB) whenever a virtual object is placed. This causes the TCP congestion window (`cwnd`) to enter slow-start before the beginning of each burst. We show that RAN segmentation latency at the RLC layer significantly impacts TCP performance, especially during the slow-start phase (§IV-E).
- We propose a network-aware AR app design optimization technique that intelligently adapts IP packet sizes for the AR app, based on the underlying RAN conditions. Our methodology in selecting packet sizes addresses the trade-off between minimizing segmentation of packets at the RAN (caused by larger IP packets), which adversely impacts network latency, and minimizing network overhead (caused by smaller IP packets), which adversely impacts application goodput. Our technique improves the aggregate RAN latency, end-to-end AR latency, network throughput and application goodput by ∼40-70% (§V-A).
- We propose a network-agnostic AR app design optimization technique that periodically updates the LTE base station (eNB) of the uplink RAN buffer status of the hosting AR device, even during gaps between AR bursts. We achieve this by generating negligible, periodic amounts of dummy data, which enables constant UE buffer status updates to the eNB. This results in improved RAN resource allocation for the AR device and minimizes uplink signaling latencies. Our technique improves the aggregate RAN latency by ∼50%, at a marginal cost of additional bandwidth (§V-B).

Since existing AR apps are closed-source and their data transmissions being opaque, our work focuses on network-layer characterizations and solutions. However, we provide a brief discussion on how one can adapt the AR application content to reduce latency, potentially with other user-perceived performance costs that may be acceptable (§V-C).

We release our RAN latency analysis tool, which runs on client devices (UEs), as open source [17]. It can be used by researchers with data captured on public LTE networks, without any modifications needed to the eNB.

II. AR Background and Related Work

Background on AR: When current AR devices (*e.g.,* those running the Apple ARKit, Google ARCore, or Microsoft Hololens AR platforms) wish to place virtual objects in the

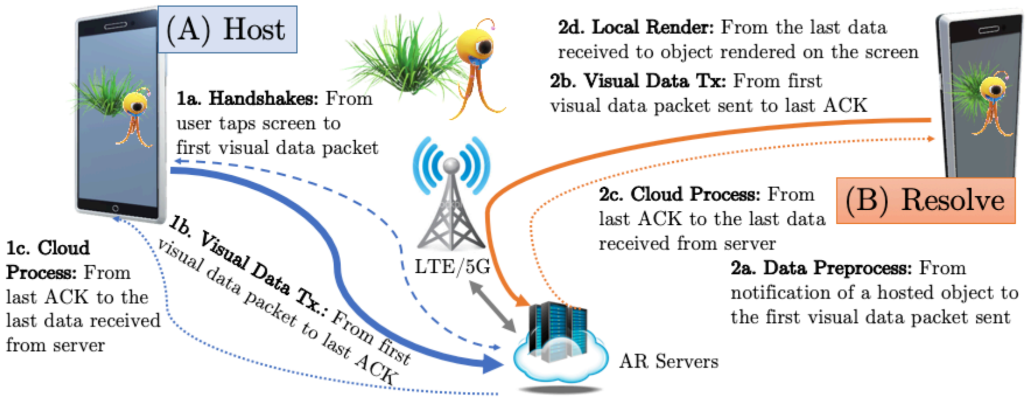

(A) Host
1a. Handshakes: From user taps screen to first visual data packet
1b. Visual Data Tx.: From first visual data packet to last ACK
1c. Cloud Process: From last ACK to the last data received from server
LTE/5G
AR Servers
2d. Local Render: From the last data received to object rendered on the screen
2b. Visual Data Tx: From first visual data packet sent to last ACK
2c. Cloud Process: From last ACK to the last data received from server
(B) Resolve
2a. Data Preprocess: From notification of a hosted object to the first visual data packet sent

Fig. 2. Cloud-based multi-user AR. Use of the cloud is mandated for multi-user AR apps in Android.

real world, they first perform device localization in order to create a consistent 3D coordinate system of the real world. The real-world coordinate system (called the *world frame*) provides a common reference for the devices to place the virtual objects, and is constructed using Simultaneous Localization and Mapping (SLAM) techniques [18]. Once the devices have a common world frame and know the poses (location and orientation) of the virtual objects, the virtual objects can be drawn on each device's display when it is within the user's field-of-view. Below we describe the steps involved in synchronizing the world frame between device A (which places a virtual object), and device B (which receives and renders that virtual object), as illustrated in Fig. 2.

1) **Hosting (device A):** (a) ***Handshakes***: Device A initiates connections with the cloud: a Firebase database, and two Google Cloud instances for visual positioning. (b) ***Visual Data Tx***: Device A sends real-world visual data and information about the virtual objects (position, orientation, 3D sprite/texture maps) to the cloud. (c) ***Cloud Process***: The cloud processes A's visual data using SLAM to compute the world frame.
2) **Resolving (device B):** (a) ***Data Preprocess***: Device B scans and retrieves camera frames and pre-processes the data. (b) ***Visual Data Tx***: Device B sends its visual data to the cloud. (c) ***Cloud Process***: The cloud matches B's visual data against the world frame computed in step 1c, and computes B's location and orientation in the world frame. (d) ***Local Render***: B uses the information from the cloud to render the virtual object at the correct position and orientation on the display.

Related Work: To the best of our knowledge, we are the first to perform an in-depth measurement study of AR applications operating on cellular networks.

Mobile AR: Many works study object detection for single-user AR [1], [8], [19], [20], [21], [22], with cloud/edge processing to reduce latency and/or energy. A few papers [11], [12], [10] discuss multi-user AR with focus on the application-layer sharing. In this work, we focus on SLAM-based AR, prevalent in off-the-shelf AR systems, and the communication aspects of multi-user AR when operating on cellular networks.

Multi-user SLAM: Some work has been done on multi-user SLAM in the robotics context [23], [24]. These works mainly focus on the SLAM algorithms themselves, and not their communication aspects.

QoS for cellular networks: Work on service-level QoS for the cellular network allocates physical resource blocks (PRBs) for users through smart eNB schedulers, QCI selection, or combinations thereof [25], [26], [27]. However, naively applying these techniques may not work well for AR's bursty traffic patterns (§IV-D).

III. Methodology, Testbeds, and Tools

Multi-User AR apps: We investigate multi-user AR with the *Cloud Anchor* [3] and *Just a Line* [28] demo apps provided by Google. *Cloud Anchor* allows one user to place a virtual object in the scene and a second user to view it. *Just a Line* allows two users to draw virtual graffiti in a shared physical space. These apps all rely on Google's CloudAnchor API [29], which is a key API to provide multi-user capabilities for Android AR apps. Thus, our observations across different apps are corroborated as they all rely on this fundamental API. The experiments in this paper are done using Cloud Anchor, except where it is specified otherwise.

Experimental setup: (a) *Industry LTE testbed:* We use an in-house outdoor 10MHz LTE testbed (operating on Band 30, 2.3 GHz) with a virtual EPC core and an LTE eNB, having 2 LTE cells with 2$\times$2 MIMO capability. Each cell yields peak uplink and downlink rates of 25 Mbps and 50 Mbps, respectively. The AR UE pair (hosting UE, rendering UE) and the load phones are connected to the eNB. We use a pair of OnePlus 5T phones as AR test devices and Samsung Galaxy S7 phones as load UEs to provide background traffic. The phones can support 2x2 MIMO. We varied the RF conditions of the AR UEs resulting in uplink SINR values ranging from $5-17$ dB and RSRP values ranging from -85 dBm to -105 dBm. The eNBs are running on HP380 servers with WindRiver Linux and transmission power of 26 dBm using 2T2R antennas. The eNBs are connected to the vEPC core, and further to the public IP network. The testbed also allows controlling the user and traffic load on the cells, and modifying parameters like QoS Class Identifier (QCI) for service differentiation of traffic classes (see Sec. IV-D).

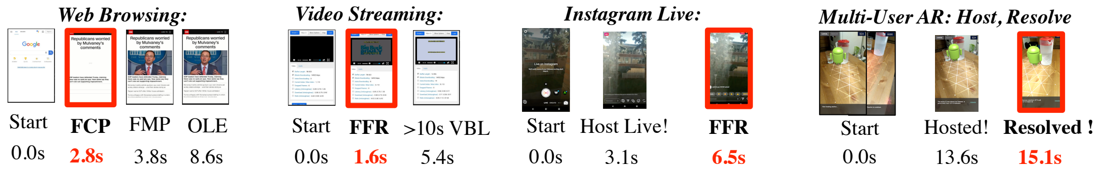


Fig. 3. Application screenshots, red-boxed screens are user-perceived events (FCP: First Contentful Paint, FMP: First Meaningful Paint, OLE: onLoad Event, FFR: First Frame Rendered, VBL: Video Buffer Length)

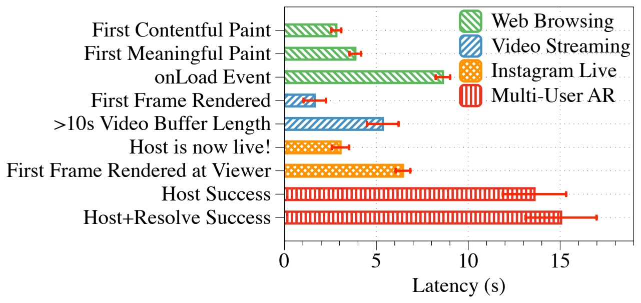


Fig. 4. User-perceived latency

(b) *Public LTE network:* We perform experiments over a public 20 MHz LTE network with 2× 2 MIMO and Carrier Aggregation capability on a Tier-I US carrier. We use a pair of Google Pixel 2 phones as AR test devices[1]. We perform experiments on public LTE measurements with RSRP and SINR values of the AR UEs ranging from -85 to -110 dBm and $5-17$ dB, respectively. We perform measurements in different locations: a campus cafeteria, a public mall, downtown and residential areas, and commercial business, with 5-12 trials per location during daytime or nighttime.

Measurement Tools: 1) *Application layer*: We instrument the AR apps to synchronize and log Unix timestamps of application events. For web and video applications, we profile the latencies on Android devices using Chrome's remote debugging developer tools [30]. 2) *TCP/IP layer*: We use `tcpdump` to capture IP packets with timestamps. 3) *RAN layer*: We measure the RAN latency by running MobileInsight [31] (MI) to capture LTE PDUs on the test UEs. We develop a custom analyzer [17] to parse the MI logs, extract PDU-level information and compute RAN latency.

IV. Measurements of AR Over Cellular Networks

A. Application-layer performance

AR streaming vs other applications: Latency is a key metric for multi-user AR. If the users experience disparate latencies, consistency issues may result, such as one user placing a virtual object and another user not being able to view it quickly, or one user attempting to manipulate a virtual object that has already been moved by another user.

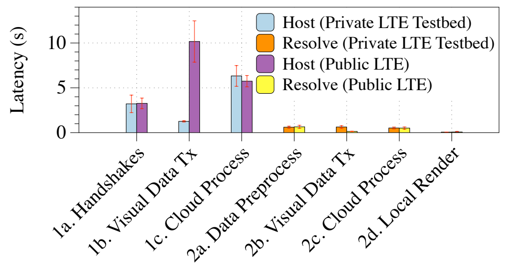
From Fig. 1, we have shown that the end-to-end latency has a very high median value of 12.5 s, resulting in poor QoE for the AR user [4]. Here, we discuss why other applications, such as web, on-demand video and live streaming uploads over cellular networks, do not suffer from similar QoE problems. For this, we conduct a set of experiments where a user surfs `www.cnn.com`, streams an MPEG-DASH video, hosts a live video stream on Instagram, and plays multi-user AR on public LTE (all experiments were conducted in sequence within a one-hour duration). Fig. 3 shows screenshots from the perspective of the user, and Fig. 4 shows the latency of each application-level event. For web browsing, although the complete content is only loaded (`onLoad` event) 8.6 s after the user starts browsing , the first contentful paint and first meaningful paint happen at 2.8 s and 3.8 s respectively. Similarly, for on-demand video, the user sees the first frame rendered on the screen after 1.6 s, due to video rate adaptation by MPEG-DASH. On-demand video also has playback buffers for pre-fetching video chunks and so is not latency-critical, as AR is. Similar video rate adaptation mechanisms apply to Instagram Live. The stream goes live within 3.1s (as notified by the server), and a viewer can view the first frame 3.4s later (6.5s after the host started the stream).

In summary, even though it takes 5-8s to download the full web or video content, user experience is not impacted because web and video have application-layer adaptation mechanisms (*e.g.,* paint the visible part of the webpage as soon as possible, adapt the video quality to have a low time-to-first render, or cache frames in the video buffer). In contrast, in multi-user AR, which lacks application-layer adaptation mechanisms, the resolving user can only see the content after the hosting user finishes its entire data transmission, which takes ∼15s.

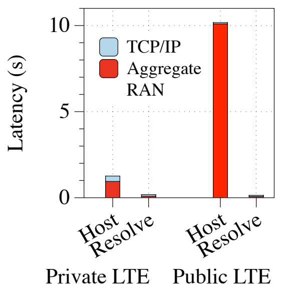
B. Breakdown of end-to-end latency

Latency breakdown: Having shown the detrimental impact of latency on AR QoE, we seek to understand the key contributors to the high end-to-end latencies observed in Sec. IV-A. We plot the constituent components of the latency in Fig. 5a, descriptions of which are provided in Sec. III. On our industry LTE testbed, we see that the key latency contributors are on the hosting side (device A): the handshakes, the visual data transmission, and the cloud processing all together consume 10s on average (86.6% of the total end-to-end latency), while the resolving side (device B) includes data preprocessing, visual data transmission, cloud processing, and local rendering, and are relatively quick (12.9% of end-to-end latency). On public LTE, the hosting steps takes 93.3% of the end-to-end latency because of its long visual data transmission latency. The visual data transmission is the largest contributor to latency on public LTE (10.1s on average, or 48.7% of the end-to-end latency), but consumes less time on the LTE testbed (17.2% of the end-to-end latency) due to lack of contention

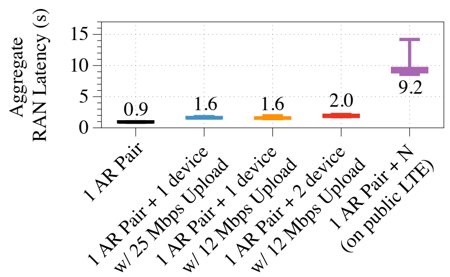
[1]Different UE device models do not make a significant difference in the experiments because AR cloud servers perform the heavy computations and AR UEs only send data to the cloud and render virtual objects.



(a) Latency breakdown

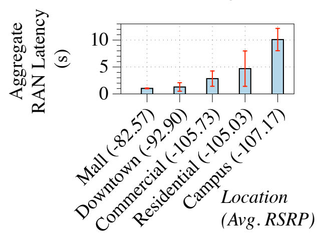


(b) RAN latency

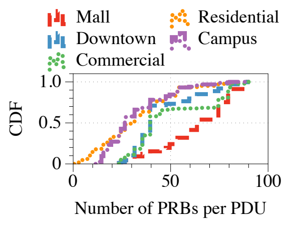


(c) Impact of other users

Fig. 5. Breakdown of end-to-end latency. Visual data transmission and RAN latencies are significant.



(a) Aggregate RAN latencies at five locations



(b) Uplink physical resources

Fig. 6. CloudAnchor measurements at five different locations.

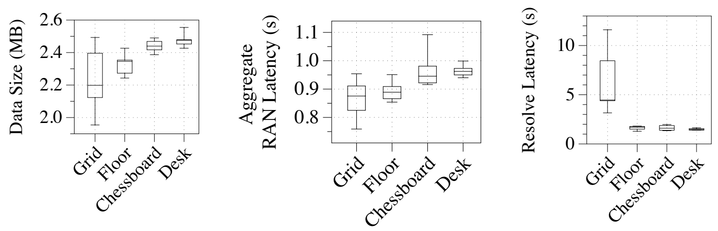
with other users (we observe similar latency on low congestion public LTE at a mall in daytime/weekday shown in Fig. 6a). The high transmission latencies exceed previously observed communication delays on AR research prototypes [7]; we hypothesize that this is due to information from multiple frames being uploaded, as well as additional image features such as point cloud data (full details are unknown because the off-the-shelf AR systems that we test on are closed source).

Wireless latency matters: Since we observe above that the uplink visual data transmission latency by the hosting device is significant, we further decompose the visual data transmission time into TCP/IP and radio access network (RAN) components. This provides an understanding of how much time is spent on the wireless link, and how much time is spent in the wired backbone. The aggregate TCP/IP latency across IP packets is measured as the elapsed time from the first visual data packet transmission at the TCP layer to the reception of the last packet's ACK (step 1b in Sec. III). The aggregate RAN latency across IP packets is the total time from when the first visual data packet is received by the RAN layer (i.e. LTE's PDCP), until when the last packet is sent to the MAC layer for transmission. It is subsumed within the aggregate TCP/IP latency. The results are shown in Fig. 5b. We observe that the RAN contributes the majority of the visual data transmission time (71.7% of the visual data transmission time on LTE testbed, and 98.5% on public LTE), suggesting that the wireless link is a key contributor to end-to-end latency, especially on public LTE where the data transmission on the RAN takes an average of 10.1 s. Hence further optimizations of the wireless link are needed, as discussed in §IV-D and §V.

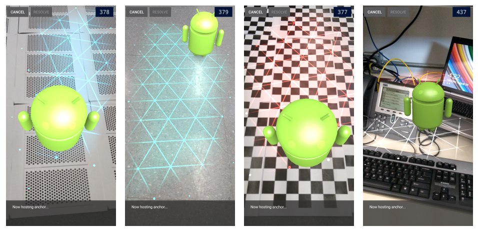
In addition to the above results on campus using public LTE, we repeat the measurements at four other locations with different wireless signal strengths (RSRP), as shown in Fig. 6a. RAN latencies at these locations range from 1-10s and seem to be correlated with the RSRP. The devices with poor signal strength tend to have fewer uplink resources allocated, as shown in Fig. 6b. Hence our wireless link optimizations of Sec. IV-D, particularly the QCI-based adaptations that impact uplink resources, can potentially provide the most gains for devices with poor signal quality.

Finally, we examine the impact of background users on AR performance, and the impact of AR applications on the background users. In our industry LTE testbed, we set up background devices uploading `iPerf3` UDP traffic with finite send buffer (12 or 25 Mbps, representing 50% or 100% of the maximum uplink RAN capacity, respectively), and plot the results in Fig. 5c. We see one background user cause a 65.5% increase in RAN latency for the AR user and two background users cause a 111.3% increase. On the public LTE where the number of background users and their traffic are uncontrolled and unknown, the RAN latency for the AR user increases $\sim 10\times$ possibly due to high cellular network congestion.

Resolve latency: While in the majority of cases, the visual data transmission by the hosting device (step 1b, §II) contributes greatly to the end-to-end latency, in a few cases, we actually observed that the virtual object resolving process can cause high delay. This is despite the small amounts of data being uploaded by the resolving device (step 2b, §II). We observe that this happens when the user tries to place virtual objects in real-world environments lacking visual features (*e.g.,* high-contrast edges, colors, etc.). We experimented with several real-world environments ranging from simple to complex, as shown in Fig. 7b. The RSSI remains relatively constant at -66 dBm. We measured the data size, uplink RAN latency, and resolve latency and plot the results in Fig. 7a. In the simple grid and floor environments which lack visual features, we observed relatively less data uploaded by host device A (2.2-2.34 MB on average) and thus lower uplink RAN latency (0.88-0.89 s on average). However, these simpler environments also caused high resolve latency, as shown in Fig. 7a. This is due to multiple rounds of communication between the device B and the cloud, unlike the typical scenario of 1-2 rounds of communication we had observed in the non-grid environments. We hypothesize that the lack of visual features in the grid environment causes difficulties in the world frame construction (step 1c, §II) , resulting in these multiple rounds

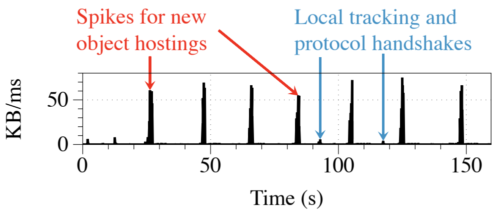


(a) Data size, host RAN latency, resolve latency

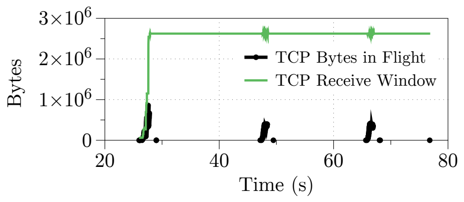


(b) Real-world environments: grid, floor, chessboard, desk.

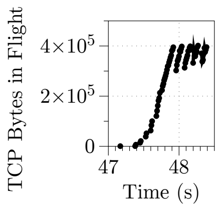
Fig. 7. Hosting augmented objects in a grid environment results in the smallest data sizes and RAN latency, but requires multiple attempts to resolve the virtual object.



(a) Throughput over time.



(b) Bytes in Flight of first three bursts in 8a.



(c) Second burst zoom in 8b.

Fig. 8. AR apps exhibit large, unpredictable data spikes on the same TCP connection, which results in TCP slow start re-triggering each spike.

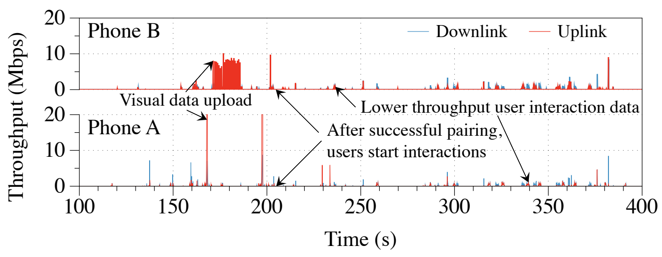


Fig. 9. Another AR app, Just A Line, also exhibit smaller data spikes after the initial large ones, corresponding to users drawing virtual graffiti.

of communication (step 2b) as device B uploads additional visual data to aid in cloud processing (step 2c).

C. AR traffic characteristics

Bursty uplink AR traffic: To understand why and how often the visual data transmissions occur, we examine their relationship with AR user interactions (*e.g.*, placing virtual objects, drawing virtual graffiti). Fig. 8a shows a sample throughput trace of device A running the CloudAnchor app (other traces are similar; we show one example for brevity). The large spikes correspond to large data transmission bursts when the user touches device A's screen to place a virtual object. We observe that most of the data transmissions happen on the uplink from host device A (2.5 MB on average), while the amount of data generated on the downlink or by device B is negligible ($<$ 100 KB). These larger data sizes contribute to the high end-to-end latencies discussed above.

We also observe a second, smaller type of AR user interaction data, as exemplified by the throughput trace in Fig. 9 from the Just a Line app. Fig. 9 shows both the large visual data spikes near $t = 170, 200$ s, similar to those observed in the CloudAnchor app in Fig. 8a, and smaller bursts near $t = 230, 295, 340$ s, etc. These smaller bursts correspond to the user touching the screen drawing virtual graffiti (447 bytes of IP packet length on average), and are smaller than the initial visual data spikes (1430 bytes on average).

In summary, AR traffic has bursts of both large and small data, corresponding to different types of user interactions/scenarios. Based on our understanding of ARCore [3], we posit that the larger spikes correspond to visual data about the scene, which is necessary whenever the app is initialized or the user moves to a new location and wishes to place virtual objects, while the smaller spikes correspond to user interactions with the virtual objects after the initial visual data has been uploaded. Time delay between data spikes depends on the frequency of user interactions, which can be unpredictable, depending on the application content.

Interaction with TCP: One implication we observe from the bursty nature of AR traffic is its interaction with TCP congestion control. All the data spikes happen in the same TCP stream, and so are affected by the same receive window. In Fig. 8c, we plot the number of TCP bytes in flight corresponding to the second spike in Fig. 8b, which corresponds to the first three spikes in Fig. 8a. We observe that each time a data spike happens (when the AR application has visual data to send), the number of TCP bytes in flight has to grow in a slow start phase. This is because the TCP congestion window decreases when the connection is idle, in between the AR user's interactions. On the other hand, applications such as video live streaming continuously have application-layer video data ready to upload (we observed this in our experiments with *Instagram Live*), and can continuously grow the congestion window without repeatedly dropping to slow start.

D. Can dedicated QoS classes help AR?

QoS Class Identifiers (QCI) are widely used by network providers [25] to offer differentiated QoS for services, where

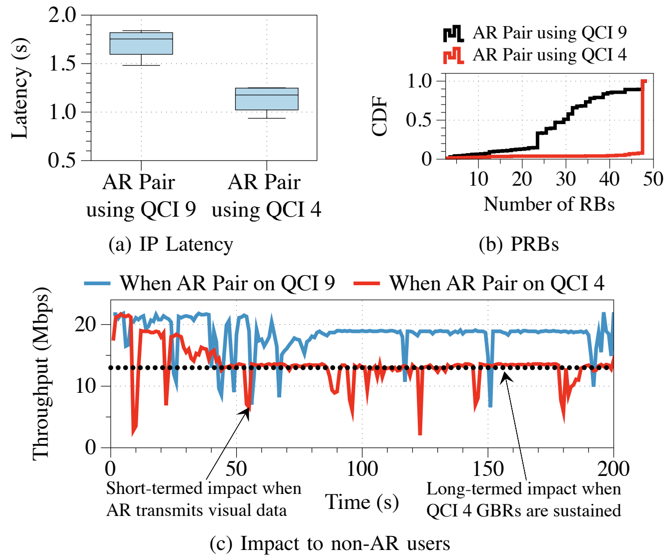


Fig. 10. Impact of QoS dedicated bearer: Assigning QCI-4 with a guaranteed bit rate to the AR user reduces its latency, but decreases the throughput of other users, even if the AR user is not transmitting in between user interactions.

a service is assigned to a bearer with a specific QCI value for data transmission. In our industry LTE testbed, allowing control over the QCI classes, we set up one AR pair and one background user uploading 25 Mbps `iperf3` traffic. The AR users are configured to use QCI-4 with a guaranteed bit rate (GBR), while the other user remains on the default, best-effort QCI-9. We configure QCI-4 to have a very high GBR of 25 Mbps (the total available bandwidth), in order to ensure that the AR users receive prioritization without any limitations.

Fig. 10a shows the latency of the AR user with and without QCI-4. QCI-4 helps reduce the TCP/IP latency by 33%, which represents an upper bound improvement even if more users were present. The improvement is due to resource prioritization for the AR user, because the higher priority of QCI-4 allows the AR flow to be scheduled on the majority of available PRBs in the cell, as shown in Fig. 10b, and across consecutive TTIs contiguously.

While a dedicated bearer such as QCI-4 can help reduce AR latency, Fig. 10c shows the performance achieved by the non-AR iPerf user. When the AR flow is assigned to QCI-9, the iPerf user can obtain most of the available bandwidth (20 Mbps), with occasional dips due to the AR user's data bursts. In contrast, when the AR flow is assigned to QCI-4, the iPerf user has its throughput reduced to an average of 13 Mbps, even when the AR user is not transmitting, because the eNB permanently reserves wireless resources for the GBR user [25]. While a live-streaming video service assigned to a GBR bearer stream would continuously and predictably utilize the assigned network resources, the AR flow wastes network resources due to its bursty and unpredictable nature.

These results suggest several challenges in designing an "AR-specific" QCI class. The RAN should to be able to predict when an AR data spike is about to begin, quickly assign this flow to a dedicated bearer, estimate when the spike is about to end, and finally remove the dedicated bearer. This can prevent negative impacts to other users in the network. For example, in light of the large and small data spikes observed in Sec. IV-C, we may not need to keep a dedicated bearer after the large data spikes have occurred.

E. Below the IP Layer: RAN Analysis

In this section, we take a detailed look at AR's behavior below the IP layer, in order to understand LTE's impact on AR performance. The IP layer passes its packets to LTE's PDCP layer, and from there to the RLC, MAC, and finally PHY layer as PDUs for transmission. The channel conditions and the traffic load generated by all the users determine the size of the RLC PDU in the current scheduling period for a given device. Based on the PDU size, the RLC layer then performs an important operation: it concatenates or segments the IP packets to fit into the RLC PDU(s), which is a key contributor to RAN latency.

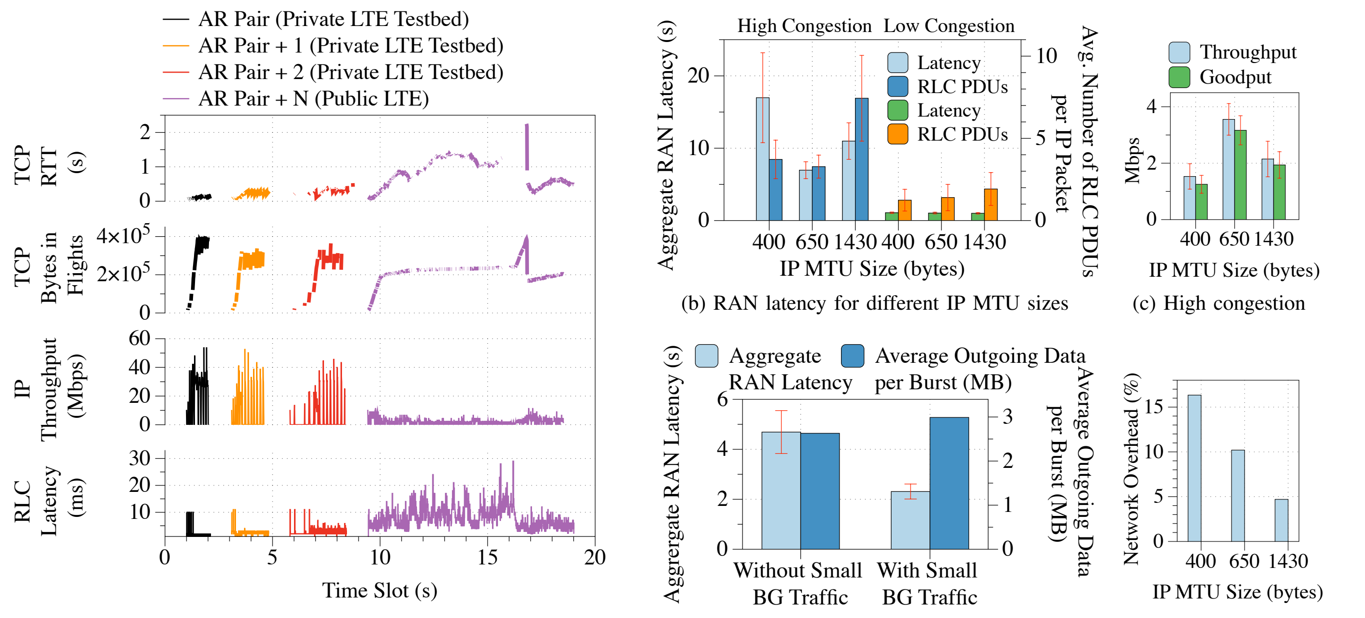
Fig. 11a illustrates the relationship between per-packet RLC latency and IP throughput for the similar test cases as §IV-B. *AR pair (+ #load phones)* where load phones generate 12 Mbps finite-buffer traffic are performed on our private LTE testbed while *AR pair + N* trails are done in public LTE with unknown traffic and number of load phones. Across test cases, we observe that the TCP RTT (first row) increases with RLC latency (bottom row), especially in the public LTE test case. The longer RTT can cause the TCP congestion window to ramp up slowly. This is shown by the relatively smaller rate of growth in the number of TCP bytes in flight over time (second row), especially during the slow-start phase when the RLC latency is higher, subsequently resulting in reduced IP throughputs (shorter and sparser lines in the third row). The impact of RLC latency on TCP slow start is crucial in AR because AR can be prone to frequent slow start phases due to the time gap between user interactions, as discussed in Sec. IV-C. The relationship between RLC latency, TCP RTT, and IP throughput suggests that RLC latency is an important factor to increase the throughput and improve the latency of AR applications. We discuss potential solutions below.

V. AR Design Optimizations

In this section, based on the insights gleaned from the traffic characterization in §IV, we provide AR design optimizations.

A. Network-Aware Optimization: Packet Size Adaptation

When the AR app uses larger IP packet sizes for transmission, it could experience heavier segmentation at the RLC layer, especially when the RLC PDU sizes are significantly smaller than the packet sizes. This happens in scenarios when the RAN is congested and/or when the UE's RF conditions are poor, as shown in Fig. 11. As a result, the per-packet RLC latency and subsequently, the TCP RTT increase, adversely impacting the growth of the TCP `cwnd` during an AR burst, deteriorating the end-to-end performance of the AR session. While using smaller IP packets can help address this issue, they increase network overhead due to generation of a higher number of packets for the same burst and under-utilization



(a) Per-packet RLC latency

(b) RAN latency for different IP MTU sizes

(c) High congestion

(d) Small background traffic

(e) Network Overhead

Fig. 11. Per-packet RLC latency is high at the beginning of the data spike, increasing RTT especially during TCP slow start. A smaller IP MTU reducing RLC segmentation and small amount of background traffic are possible solutions to help reduce AR latency.

of the available RAN capacity. With sub-optimal, smaller IP packets, this overhead becomes significantly high, affecting application goodput (see Fig. 11e). We propose a technique to optimize the packet sizes of the AR app by heuristically addressing this non-linear trade-off, based on underlying RAN conditions. In particular, when there is significant RLC segmentation of the packets, we reduce the IP packet size closer to a moving average of the instantaneous RLC PDU sizes for the UE. We carefully adapt IP packet sizes so that the gain from a quicker increase in the TCP congestion window for an AR burst offsets the loss from additional overhead of using more IP packets for the same burst.

Packet sizes can be varied by configuring the Maximum Segment Size (MSS) of the AR flow or Maximum Transmission Unit (MTU) of the AR UE. We conduct CloudAnchor experiments by adapting the IP packet sizes of the AR streaming session over public LTE networks under different network conditions (both congested and less-congested scenarios) using our technique and present the results in Fig. 11b. We evaluate the packet sizes selected by our technique (650 bytes) against the default large packet size of 1430 bytes and a smaller sub-optimal packet size of 400 bytes.

In Fig. 11b, for a more congested public LTE network (*i.e.,* campus) scenario, the default large packet size of 1430 bytes undergoes significant segmentation (around 7 RLC PDUs per packet) and hence, the aggregate RAN latency is high ($\sim$11 s). The optimal packet size for this scenario, yielded by our technique, is a smaller value, around 650 bytes. Upon setting this, the aggregate RAN latency is reduced by 37% and 58%, when compared to 1430 bytes and 400 bytes, respectively. At the same time, the network throughput and the AR application goodput from a 650-byte packet size increases by over 62% and 150% than the 1430-byte and 400-byte packet sizes, respectively (Fig. 11b, 11c). The 400-byte packet sizes achieve lowered throughput despite similar RLC segmentation to 650-byte is because the former under-utilizes the network capacity, observed by the maximum TCP `cwnd` for 400-byte reaching only 493KB, while 650-byte and 1430-byte can reach 657KB and 690KB, respectively. However, in low network congestion environments (*i.e.,* the mall), reducing the packet size has little impact on aggregate RAN latency because the eNB already allocates a larger RLC PDU to the device, resulting in little RLC segmentation. Hence, our technique selects the default large packet size close to 1430 byes. In conclusion, network-aware AR app design by choosing smart packet sizes for the app can improve aggregate RAN latency, end-to-end AR latency, network throughput and application goodput.

B. Network-Agnostic Optimization: Small Background Traffic

Another AR design optimization we propose is "priming" the eNB with information about the amount of data that the AR application will transfer in its next burst. This technique is network-agnostic, without the need to adapt to variations in the RAN. Typically, when the hosting device starts sending either a new uplink AR burst or new data in the middle of an AR burst after a longer idle period (lasting for tens of milliseconds), the UE has to request for resources from the eNB, incurring protocol signaling latency. The eNB is initially unaware of the uplink sending buffer, and may only allocate a small uplink grant (max. 125 bytes) for the UE. Then, upon data PDU transmission, the UE also piggybacks the uplink buffer size, which the eNB subsequently uses to allocate larger resource grants. This causes RLC segmentation, increasing the per-packet RAN latency, especially in congested scenarios. In order to make the eNB aware of the device's uplink buffer during an AR session, we generate small amounts of background uplink traffic, using an `icmp` packet of 100 bytes every $2-5$ms. Since there is an active small data transfer even during inter- or intra-AR burst idle periods, the UE is always scheduled minimal resources and it constantly piggy-backs information about its uplink sending buffer to the eNB. This maximizes buffer-aware scheduling for the UE, which minimizes protocol signaling latency and RLC segmentation. In Fig. 11d, we plot the aggregate RAN latency and amount of outgoing data, with and without the additional background traffic, for an AR session over public LTE network. The results show that this small background traffic helps reduce aggregate RAN latency by $\sim$50% on average, at the cost of a negligible increase in outgoing data size (including the extra background traffic). Our UE logs show that the average uplink resource grant for each MAC PDU during the AR session increases from 593 bytes to 1191 bytes with small background traffic.

C. Discussion: Application-layer Optimizations

Finally, we briefly discuss other potential application-layer solutions to reduce AR latency. In our existing experimental setup, the network data transmissions were opaque due to the internals of the Google ARCore platform being closed source. However, we hypothesize that the data transmissions consist of device data that is used for localization, as localization is known to be an integral part of AR [18] Reducing the fidelity of the device localization data, for example by quantizing the data or sub-sampling the data in time, could reduce the amount of data requiring transmission and thus the network latency. On the other hand, this may reduce device localization accuracy and impact the placement of virtual objects in the user's display; thus, we intend to explore such effects in future work, using open-source AR systems [9] that allow modification of the application layer.

VI. Conclusions

The goal of high quality AR has engendered tremendous amount of research, but there has thus far been little focus on the impact of the cellular network. In this paper, we show through extensive measurements on both an industry LTE testbed and public LTE that RAN latency is a significant part of the end-to-end AR experience, accounting for nearly 31.2% of the total latency. Unless this is reduced significantly, there is little hope for achieving AR with high QoE. However, our results also provide hope: AR traffic is very bursty in nature, making it a suitable candidate for practical traffic management schemes like QCI (which improves latency by up to 33%). Further, we also design network-aware and network-agnostic optimizations that improve latency by $\sim$40-70%. Future work includes a longitudinal study of AR users to learn specific AR app behaviors, which can then drive the development of a smart QCI-based scheduler specifically tailored for AR traffic characteristics. We will also quantify how 5G technologies can help close the gap of achieving seamless multi-user AR QoE by reducing the overall RAN latency.

Acknowledgment

We thank AT&T Labs Research for its internship program where the research was conducted. We also thank the anonymous reviewers for their valuable comments. This work has been supported in part by NSF grant CNS-1942700.

References

[1] L. Liu *et al.*, "Edge assisted real-time object detection for mobile augmented reality," *ACM MobiCom*, 2019.

[2] T. Merel, "The reality of vr/ar growth." https://techcrunch.com/2017/01/11/the-reality-of-vrar-growth/.

[3] Google, "Share ar experiences with cloud anchors," https://developers.google.com/ar/develop/java/cloud-anchors/cloud-anchors-overview-android.

[4] Z. Chen *et al.*, "An empirical study of latency in an emerging class of edge computing applications for wearable cognitive assistance," in *ACM/IEEE Symposium on Edge Computing*. ACM, 2017.

[5] M. Suznjevic *et al.*, "Analyzing the Effect of TCP and Server Population on Massively Multiplayer Games," *International Journal of Computer Games Technology*, vol. 2014, no. 602403, pp. 1 – 17, 2014.

[6] A. Langley *et al.*, "The quic transport protocol: Design and internet-scale deployment," in *ACM SIGCOMM*, 2017.

[7] P. Jain *et al.*, "Overlay: Practical mobile augmented reality," *ACM MobiSys*, 2015.

[8] X. Ran *et al.*, "DeepDecision: A Mobile Deep Learning Framework for Edge Video Analytics," *IEEE INFOCOM*, 2018.

[9] P. Li *et al.*, "Monocular visual-inertial state estimation for mobile augmented reality," in *IEEE ISMAR*, Oct 2017, pp. 11–21.

[10] X. Ran *et al.*, "ShareAR: Commnication-Efficient Mobile Augmented Reality," *ACM HotNets*, 2019.

[11] W. Zhang *et al.*, "Cars: Collaborative augmented reality for socialization," *ACM HotMobile*, 2018.

[12] H. Qiu *et al.*, "AVR: Augmented vehicular reality," *ACM MobiSys*, 2018.

[13] Google, "Arcore overview," https://developers.google.com/ar/discover/.

[14] Apple, "Arkit - apple developer," https://developer.apple.com/arkit/.

[15] Microsoft, "Shared experiences in unity," https://docs.microsoft.com/en-us/windows/mixed-reality/shared-experiences-in-unity, March 2018.

[16] "Magic leap," https://www.magicleap.com/.

[17] K. Apicharttrisorn *et al.*, "Custom mobileinsight analyzer and python scripts for ran-level deep analysis," https://github.com/patrick-ucr/ran_latency_analyer_mi.

[18] D. Schmalstieg and T. Hollerer, *Augmented reality: principles and practice*. Addison-Wesley Professional, 2016.

[19] Q. Liu and T. Han, "Dare: Dynamic adaptive mobile augmented reality with edge computing," *IEEE ICNP*, 2018.

[20] W. Zhang *et al.*, "Jaguar: Low Latency Mobile Augmented Reality with Flexible Tracking," *ACM Multimedia*, 2018.

[21] K. Chen *et al.*, "MARVEL: Enabling mobile augmented reality with low energy and low latency," *ACM Sensys*, 2018.

[22] K. Apicharttrisorn *et al.*, "Frugal following: Power thrifty object detection and tracking for mobile augmented reality," *ACM SenSys*, 2019.

[23] D. Zou and P. Tan, "Coslam: Collaborative visual slam in dynamic environments," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 35, no. 2, pp. 354–366, 2012.

[24] P. Schmuck and M. Chli, "Multi-uav collaborative monocular slam," *IEEE International Conference on Robotics and Automation*, 2017.

[25] M. Alasti *et al.*, "Quality of service in wimax and lte networks," *IEEE Communications Magazine*, vol. 48, no. 5, pp. 104–111, 2010.

[26] H.-C. Jang and C.-P. Hu, "Fairness-based adaptive qos scheduling for lte," in *IEEE International Conference on ICT Convergence*, 2013.

[27] M. Anas *et al.*, "Combined admission control and scheduling for qos differentiation in lte uplink," in *IEEE Vehicular Technology Conference*, 2008.

[28] Google, "Just a Line - Draw Anywhere, with AR," https://justaline.withgoogle.com/.

[29] Google, "Arcore cloud anchor api," https://console.cloud.google.com/marketplace/details/google/arcorecloudanchor.googleapis.com.

[30] K. Basques, "Get started with remote debugging android devices," https://developers.google.com/web/tools/chrome-devtools/remote-debugging.

[31] Y. Li *et al.*, "Mobileinsight: Extracting and analyzing cellular network information on smartphones," in *ACM MobiCom*, 2016.