The Power of Asynchronous SLAM in Multi-User AR over Cellular Networks: A Measurement Study

Yuting Guo[1], Sizhe Wang[1], Moinak Ghoshal[1], Y. Charlie Hu[2], Dimitrios Koutsonikolas[1]
[1]Institute for the Wireless Internet of Things, Northeastern University, USA,
[2]Purdue University, USA

ABSTRACT

With the rapid deployment of 5G, an important question is whether 5G can support latency-critical apps such as multi-user AR, which allows multiple users to interact in the same physical space in real time. Recent studies showed that a popular multi-user cloud-based AR app (Cloud Anchor) suffers long end-to-end (E2E) latency in multi-user interactions under both LTE or 5G and hence cannot be supported by today's cellular networks. In this paper, we revisit the feasibility of multi-user AR over wireless networks by experimenting with another popular multi-user AR app, Just a Line, and find that its E2E latency is in the order of a few 100s of ms over both LTE and 5G, making real-time multi-user interactions feasible. We conduct a detailed measurement study of the Just a Line app over cellular networks and uncover the drastic user-perceived performance difference between the two multi-user apps stems from two fundamentally different architecture designs of performing SLAM: asynchronous vs. synchronous updates.

CCS CONCEPTS

• **Human-centered computing** → **Ubiquitous and mobile computing**; • **Networks** → **Mobile networks**; • **Information systems** → **Multimedia information systems**.

ACM Reference Format:
Yuting Guo, Sizhe Wang, Moinak Ghoshal, Y. Charlie Hu, Dimitrios Koutsonikolas. 2023. The Power of Asynchronous SLAM in Multi-User AR over Cellular Networks: A Measurement Study. In *ACM SIGCOMM 2023 Workshop on Emerging Multimedia Systems (EMS '23), Septmeber 10, 2023, New York, USA.* ACM, New York, NY, USA, 7 pages. https://doi.org/10.1145/3538394.3546042

1 INTRODUCTION

Augmented Reality (AR) promises unprecedented interactive and immersive experiences to users by augmenting physical objects in the real world with computer-generated perceptual information. As such, a complete AR app often needs to perform several challenging tasks to understand and interact with the physical environment, such as pose estimation or object detection [1].

While single-user AR can potentially perform AR tasks locally on the mobile device [10], *multi-user AR apps*, which allow multiple users to interact within the same physical space, critically rely on the cellular network and often a cloud server to support user interactions. Further, to provide high-quality, interactive experience, such networked AR apps need to perform the needed AR tasks (e.g., pose estimation and synchronization to the same physical environment) at very low latency, which places high uplink bandwidth demand on the wireless network. It is because of this stringent network requirement that networked AR has been widely viewed as one of the "killer" apps for 5G, e.g., in the AT&T and Microsoft alliance as well as the Verizon and AWS alliance to showcase 5G edge computing solutions [2, 7].

Unfortunately, previous studies have shown that cellular networks *cannot* support multi-user cloud-based AR apps. First, Apicharttrisorn *et al.* conducted an in-depth measurement study [8] of a popular multi-user app (Cloud Anchor [3]) that performs the most basic multi-user interaction, i.e., displaying a virtual object, to study whether LTE can support the needed QoE of multi-user AR. That study showed that the latency from the moment a user places a virtual object in the physical environment to the moment the object is displayed on another device is 12.5 s in the median case over LTE, which renders the most basic user interaction in multi-user AR apps practically infeasible. Motivated by the much higher bandwidth and lower latency provided by 5G mmWave, in a follow-up study [13], Ghoshal *et al.* conducted a comparison of the performance of the same AR app side-by-side over both LTE and 5G mmWave and found that 5G mmWave did not reduce the E2E latency of the AR app compared to LTE in spite of its much higher bandwidth and lower RTT. Based on the results of these studies, a number of follow-up works proposed edge-assisted [9] (as opposed to cloud-assisted) multi-user AR to reduce the E2E latency or a new P2P paradigm for multi-user AR apps to replace the client-server paradigm [9, 18].

Nonetheless, the findings in [8, 13] were based on experiments with a single multi-user AR app, which was treated as a representative example of all multi-user AR apps. While the main building blocks of Cloud Anchor are indeed shared by most multi-user, cloud-assisted AR apps, the way these building blocks are put together is different in different apps, and these architectural differences might result in different performance over the same cellular network. Indeed, in this paper, we experiment with another popular multi-user AR app, Just a Line [6], and find that the E2E latency is much shorter, in the order of a few 100s of ms, over both LTE and 5G, which allows for real-time interactions between multiple users. Motivated by the striking difference in the E2E latency of these two apps, we perform a detailed measurement study of the performance of the Just a Line app over cellular networks and uncover the root cause of why this app performs much better than Cloud Anchor.

Our study uncovers a fundamental architectural difference between the two apps that has a major impact on the user-perceived

EMS '2023, September 10, 2023, New York, USA

ACM ISBN 978-1-4503-9393-5/22/08…$15.00
https://doi.org/10.1145/3538394.3546042

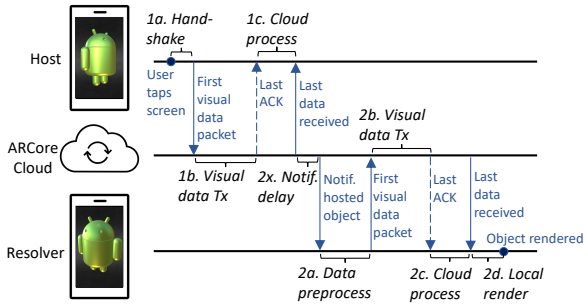
E2E latency. This difference is related to the way the two apps perform SLAM (Simultaneous Localization and Mapping), which serves as the foundation for localization in every mobile multi-user AR app. Cloud Anchor performs SLAM *synchronously*, i.e., before every virtual object placement. This coupling of the two operations – SLAM and virtual object placement – results in a high E2E latency, as a successful SLAM update always has to be completed before the resolver can render a virtual object placed by the host. In contrast, Just a Line decouples SLAM from the virtual object placement. SLAM is performed *asynchronously* in a periodic manner, and the app always uses the results from the most recent SLAM update to render a virtual object. The decoupling of the two operations allows Just a Line to achieve a much lower E2E latency compared to Cloud Anchor.

In summary, in this work, we revisit the question of whether today's cellular networks can support multi-user AR. In contrast to previous studies that gave a negative answer based on experiments with a single multi-user AR app, our study reveals that the answer to this question is more subtle and depends heavily on the app design, in particular, the way the app performs SLAM, i.e., synchronously vs. asynchronously. In this paper, we focused on the E2E latency of multi-user AR, similar to [8, 13], and ignored another important metric, spatial inconsistency [18], i.e., whether the virtual objects appear at the correct locations, with respect to the real world, on each user's display. While in our experiments we did not find any spatial inconsistency visually, in our future work, we plan to study how the two different SLAM approaches affect the spatial inconsistency, using appropriate tools and methodologies [18].

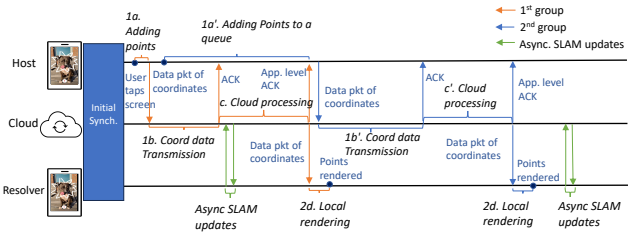
2 ASYNCHRONOUS VS. SYNCHRONOUS SLAM IN MULTI-USER AR

SLAM serves as the foundation for localization in most mobile multi-user AR apps. To create a common and consistent real-world coordinate system across multiple mobile devices, the users share their device coordinates and then SLAM is ran to estimate the device's current pose and the real-world coordinate features, before the virtual objects in the user's field of view are rendered on the screen. SLAM is computationally intensive and most popular AR apps enabled by Google ARCore, Apple ARKit, or Microsoft Hololens offload most of the computations to cloud servers to reduce the workload on the phones. Based on how SLAM is implemented in cloud-based AR systems, it can be broadly categorised into two types: 1) Synchronous SLAM and 2) Asynchronous SLAM. In the following, we describe in detail the workflow of both types of SLAM in Fig. 1 via two representative apps, Cloud Anchor and Just a Line, respectively.

Synchronous SLAM in multi-user AR. A certain class of multi-user AR applications perform SLAM whenever a change in user activity is observed. This could either mean the user changes their current position and moves to a new one, or the user adds/manipulates an overlaid virtual object. We call this reactive approach *synchronous SLAM*. Cloud Anchor [3] is a popular demo application released by Google that leverages synchronous SLAM. Fig. 1a shows the workflow of Cloud Anchor, which allows a user (host) to place a virtual object in the scene and a second user (resolver) to view it. The host initiates a connection with a cloud-based Firebase [4]



(a) Cloud Anchor: Synchronous SLAM.



(b) Just-a-Line: Asynchronous SLAM.

Figure 1: Workflow of different types of cloud-based multi-user AR.

database by creating a room ID (R). The resolver uses the same room ID and waits for incoming connections from the host via the cloud.

After an object is placed on the host's screen, the following events take place. The host device creates a connection to the ARCore cloud (**1a**), and starts uploading its coordinates and real-world visual information to the cloud (**1b**). The cloud on receiving the host's visual data, performs SLAM, returns the SLAM-computed world to the host and notifies the resolver to start the resolution process (**1c**). The resolver, upon receiving a notification from the Firebase after a notification delay (**2x**), initiates a connection to the ARCore cloud (**2a**), and uploads its visual data to the cloud (**2b**). The cloud then runs SLAM, compares the visual data with the host's SLAM-computed data to estimate the pose of the resolver in the real-world frame, and sends it back to the resolver (**2c**). The resolver uses this data from the cloud to estimate the virtual object's pose and display it on its screen (**2d**). The E2E latency is defined as the time from the initial handshake between the host and the cloud to the moment the virtual object is displayed on the resolver's screen, i.e., it is the sum of 1a, 1b, 1c, 2x, 2a, 2b, 2c, and 2d.

Asynchronous SLAM in multi-user AR. In contrast, another class of AR applications performs SLAM in a periodic manner, independent of the main application task, i.e., virtual object placement. We call this approach *asynchronous SLAM*. Just-a-line [6], another popular demo application released by Google, is an example of a multi-user AR application performing *asynchronous SLAM*. In this app, multiple users draw virtual graffiti in a shared physical space. The host draws the graffiti on the phone and the resolver sees the host's drawing on their phone's screen.[1]

[1]In Just a Line, the two devices can change roles or simultaneously play both roles during a session. In our experiments, for simplicity, we always keep one device as the host and the other one as the resolver.

Fig. 1b shows the workflow of Just a Line. After an initial synchronisation phase, where the devices communicate with each other via a cloud server to establish a session, the host taps their screen to start drawing and the app collects the coordinates of the first batch of drawn points (**1a**) and uploads them to the cloud (**1b**) for processing (**c**). After the cloud server finishes the processing, it sends an application-layer acknowledgement to the host and in parallel sends the coordinates to the resolver, which render the drawing on their display (**2d**).

Unlike Cloud Anchor, which allows placement of one object per session, Just-a-line allows users to continuously draw on their screens and performs the processing in batches. After the initial set of coordinates are sent by the host to the cloud, the host keeps adding the subsequent coordinates to a queue (e.g., **1a'**) and uploads the next batch of coordinates (**1b'**) to the cloud once it receives the acknowledgement for the completion of the previous cloud processing task. The rest of the process – cloud processing (**c'**) and rendering on the resolver side (**2d'**) – follows in a similar manner to how the first batch of points were processed and rendered on the resolver end. This sequence of steps is repeated until the session is ended. Here, we define the E2E latency for a batch of drawn points as the time from the moment the coordinates of the first point of a batch are added to the queue on the host side till the moment this batch of points is rendered on the resolver's screen. For example, the E2E latency is the sum of 1a, 1b, c, and 2d for the first batch of points, and the sum of 1a', 1b', c', and 2d' for the second batch.

Note that, in contrast to Cloud Anchor, the E2E latency here is not affected by SLAM. During 1b, 1b', ..., the host only uploads coordinates but no visual data, during c, c', ..., the cloud only processes coordinates, and there are no 2a, 2b, 2c phases, unlike in Cloud Anchor. After the initial synchronization phase, where both phones upload their visual data to the server to perform SLAM, SLAM is performed asynchronously; the resolver periodically sends their visual data (SLAM updates in Fig. 1b) to the cloud, which performs SLAM, updates the pose of the resolver in the real-world frame, and sends it back to the resolver.

3 METHODOLOGY

5G devices and carrier. Recent studies [13, 14, 16] have shown Verizon's 5G mmWave performance to be the best in terms of latency and throughput among the major US cellular providers today. Hence, we selected Verizon's 5G mmWave service for our experiments in this work. We used two rooted Google Pixel 5 phones for all our experiments over 5G, LTE, and WiFi networks. For the measurements over LTE, we disabled the 5G radio through the phone's settings. For the WiFi measurements, the phone was connected to an 802.11ac AP.

Experimental Methodology. We conducted our experiments near a busy street in Boston, MA, 80 ft away from the 5G mmWave base station (BS); we confirmed via SpeedTest measurements that this distance yielded the maximum possible downlink and uplink throughput. The experiments spanned a 1-week period and all the measurements were done at day time, from 9 am to 5 pm. For 5G mmWave, we consider two scenarios: (1) *static scenarios*, when the users stand and face towards the BS, and (2) *mobile scenarios*, when they move in front of the BS in a pseudo-random pattern incurring self-blockage. Due to the omnidirectionality of LTE and WiFi signals, the network performance is not affected by blockage or mobility. We performed the LTE experiments at the same location as the 5G mmWave experiments, and the WiFi experiments in an apartment, in the same room where we placed the WiFi AP.

Measurement Tools. To extract the end to end latency of the AR app, we modified the two apps to log the Unix timestamps and captured packets with timestamps using tcpdump.

4 RESULTS

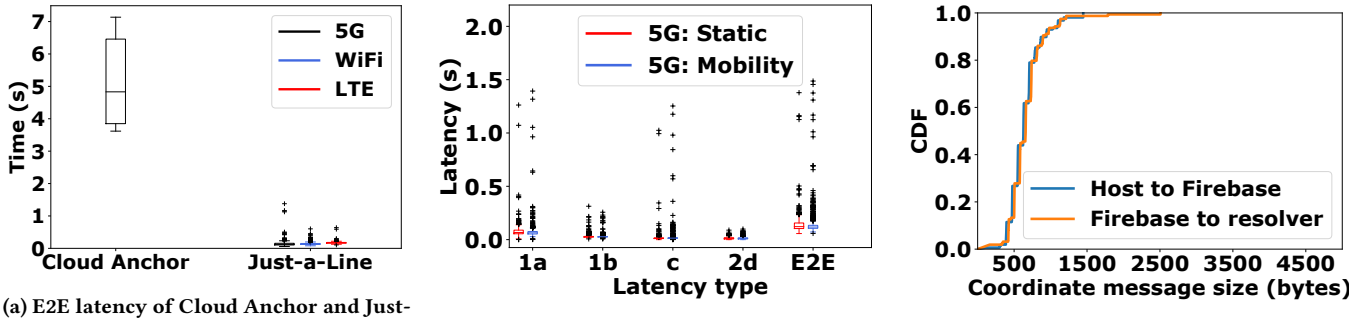
4.1 Synchronous vs. Asynchronous SLAM

E2E latency. Fig. 2a compares the E2E latency of Just a Line and Cloud Anchor in static scenarios. For Just a Line, we repeated the measurements over 3 different networks – WiFi, LTE, and 5G mmWave. For Cloud Anchor, we only conducted measurements over 5G mmWave.

Fig. 2a shows that the E2E latency with Cloud Anchor varies between 3.8 s and 7.1 s, with a median value of 4.9 s. Although these values are lower than those reported in [13] (3.9-11.9 s with a median value of 6.3 s), they still make real-time user interactions practically infeasible. In contrast, the E2E latency with Just a Line is *an order of magnitude lower*, regardless of the underlying network. The 75-th percentile is 168 ms, and only a few outliers approach 1.5 s. Note that this worst-case latency with Just a Line is still 2.3 s lower than the best-case latency with Cloud Anchor. Since the E2E latency with Just a Line is independent of the underlying network, in the remainder of the paper, we focus on 5G mmWave.

E2E Latency breakdown. To gain insight into the much lower E2E latency of Just a Line, we plot in Fig. 2b its E2E latency, as well as the latency of its individual phases over 5G mmWave, in static vs. mobile scenarios. We observe that the E2E latency remains unaffected by user mobility. Additionally, all individual app phases are completed very fast; the 75-th percentile of all four phases – 1a, 1b, c, 2d – is below 80 ms.

The studies in [8, 13] reported that tasks 1b and 1c are the main contributors to the E2E latency of Cloud Anchor. During 1b, the host uploads the virtual object's coordinates along with real-world visual data to the cloud, and, during 1c, the cloud performs SLAM and returns the SLAM-computed world to the host. Each of these two phases can take several seconds to complete (e.g., 0.8-2.6 s and 1.9-7.5 s, respectively, in [13]). In contrast, in Just a Line, the host only uploads coordinates of the drawn graffiti during 1b, and processes those coordinates during c, while the visual data required for SLAM are uploaded (by the resolver) and processed (by the cloud) asynchronously, as shown in Fig. 1. As a result, both these phases complete very fast (at most 400 ms and 1.3 s, respectively), as shown in Fig. 2b. Fig. 2c shows the CDF of the size of the coordinate messages sent by the host to the cloud and from the cloud to the resolver, at the beginning of 1b, 1b', ..., and 2d, 2d', ..., respectively. We observe that the size of both messages is very small (99-th percentile 1500 bytes, max 2400 bytes). In contrast, the size of the messages sent during 1b in Cloud Anchor (including both coordinates and SLAM updates) was found to be in the order of a few MB in [8, 13], which explains the difference (an order of magnitude) in the duration of 1b for the two apps.



(a) E2E latency of Cloud Anchor and Just-a-line.

(b) Latency breakdown of Just-a-Line.

(c) CDF of Just-a-Line coordinate message size.

Figure 2: Just-a-line: Latency breakdown and packet size statistics.

Fig. 2b also shows that, in addition to phase c, phase 1a also exhibits some outliers up to 1.4 s, while 1b and 2d are always completed very fast. Recall from §2 that, during 1a, the host uploads the coordinates of a batch of drawing points to the cloud, which processes them during c. Also recall that the uploading of the next batch of coordinates can only start after the cloud finishes processing of the current batch and sends a notification to the host. As a result, if the cloud delays processing of a batch of coordinates (due to overloading), this delay will also delay the uploading of the next batch of coordinates, i.e., a long c for a batch of coordinates always causes a long 1a for the next batch of coordinates. This explains the outliers observed for 1a and c in Fig. 2b.

4.2 Updates in Asynchronous SLAM

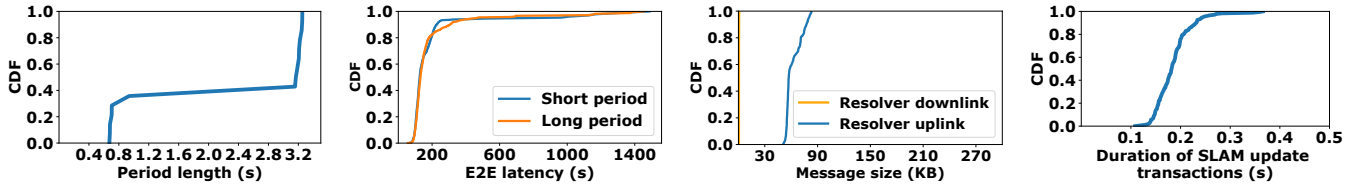
In the previous section, we focused on the E2E latency of Just a Line and ignored the asynchronous SLAM updates. We next turn our attention to the periodic SLAM updates and study their period, size, and impact on E2E latency.

Update interval. Fig. 3a plots the CDF of the SLAM update period over all static and mobile experiments. We observe that the CDF is largely bimodal; the SLAM update period is either 0.7 s or 3.1-3.2 s for most experiments. Interestingly, our experiments showed that the SLAM update period is not correlated with the user mobility. Figs. 4a-4d show four example timelines, two in static (Figs. 4a, 4b) and two in mobile scenarios (Figs. 4c, 4d). We observe that the app can use a short update period in static scenarios (e.g., Fig. 4a) and a long update period in scenarios involving user mobility (e.g., Fig. 4d). Upon closer inspection of our traces, we found that the SLAM update period is determined based on the dynamism of the real-world scene captured by the phone's camera rather than by user mobility. SLAM updates are sent more frequently when the phone faces a highly dynamic scene (e.g., a busy street) and less frequently when it faces a static scene (e.g., a building) regardless of user mobility.

By inspecting Figs. 4a-4d (and the rest of our traces), we observe no correlation between the SLAM update period and the E2E latency of Just a Line. These figures show that the E2E latency for most batches of coordinates is below 200 ms (as mentioned in §4.1), regardless of the SLAM update period, and outliers are observed in all four figures. This is confirmed in Fig. 3b, which plots the E2E latency separately for scenarios with short and long SLAM update period. We further observe that most E2E latency outliers occur between two consecutive SLAM update messages (which is expected, due to staleness of the latest SLAM update), but there are some exceptions; e.g., in Fig. 4b, the E2E latency spikes to 400 ms and 500 ms right after the 4th SLAM update message.

Update messages. Figs. 4a-4d also plot the size of the periodic SLAM update messages sent by the resolver to the cloud and the size of the responses sent by the cloud to the resolver after processing the SLAM updates. We observe that the size of the updates does not depend on the user mobility or the SLAM update period. We also found that in most runs, the size of all SLAM update messages is roughly constant (e.g., Figs. 4b,4d), but there are a few exceptions (e.g., Fig. 4c) where the size varies over time. Fig. 3c plots the CDF of the size of the SLAM update messages to the cloud and the size of the responses from the cloud in all scenarios. The SLAM update messages are about 50-60 KB 60% of the time and can be as large as 85 KB. In contrast, the size of the responses from the cloud to the resolver is much smaller, about 2-3 KB.

The size of the SLAM updates is similar to the size of the visual data uploaded by the resolver in the case of the Cloud Anchor app (50-100 KB in [13]) and much smaller than the visual data uploaded by the host in that app (a few 100s of KB up to a few MB, as mentioned in §4.1). This shows again the power of asynchronous SLAM. After the initial synchronization period (studied in the next section), the resolver in Just a Line only sends small periodic SLAM updates instead of full visual data, allowing the upload and the processing of those updates to complete much faster compared to the upload and processing of SLAM messages from the host in the case of Cloud Anchor. This is shown in Fig. 3d, which plots the CDF of the duration of all SLAM update transactions, i.e., the time from the moment the resolver starts sending a SLAM update message to the moment it receives the response from the cloud. We observe that the duration of the SLAM update transactions is very short, 100-370 ms. This latency is similar to the latency of the SLAM updates on the resolver side in Cloud Anchor (the sum of 2b and 2c in Fig. 1a), which was found to be in the order of a few 100s of ms in [8, 13]. In contrast, the latency of the SLAM updates on the host side in Cloud Anchor before each virtual object placement (the sum of 1b and 1c in Fig. 1a) is very long (in the order of 3



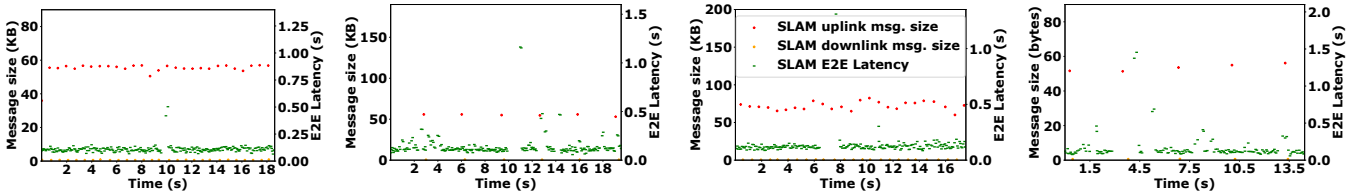
(a) CDF of SLAM update message intervals in Just a Line.

(b) E2E latency with long vs. short SLAM update period.

(c) CDF of SLAM update message size.

(d) CDF of the duration of SLAM update transactions.

Figure 3: Just-a-line SLAM statistics.



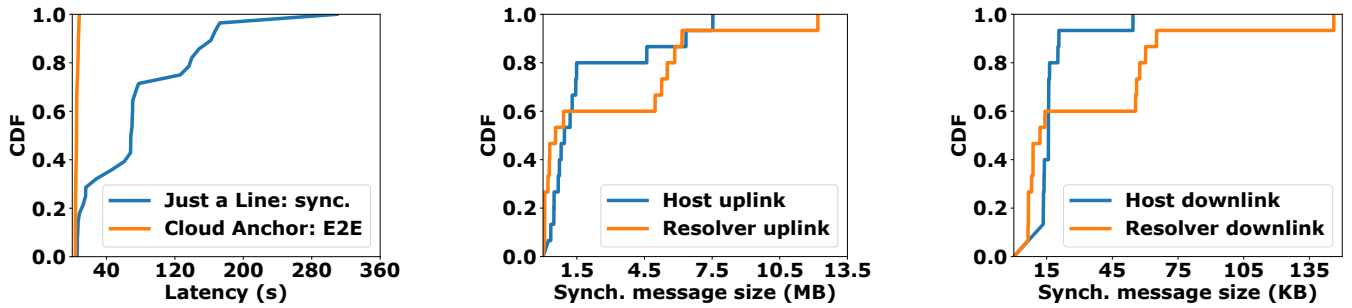
(a) Short SLAM period in a static scenario.

(b) Long SLAM period in a static scenario.

(c) Short SLAM period in a mobile scenario.

(d) Long SLAM period in a mobile scenario.

Figure 4: Just-a-line SLAM update timelines.



(a) Synchronization time comparison.

(b) CDF of total synch. uploaded bytes.

(c) CDF of total synch. downloaded bytes.

Figure 5: Just-a-line synchronization statistics.

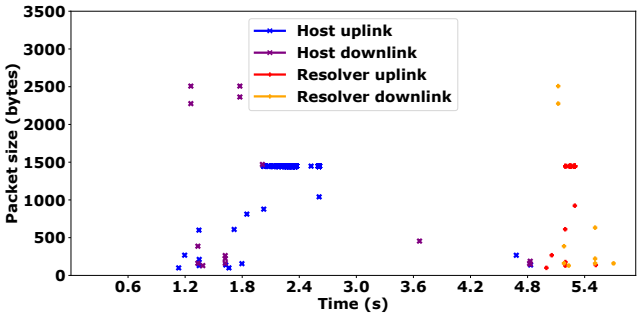
s in [8, 13]), and contributes significantly to Cloud Anchor's E2E latency. The asynchronous SLAM updates in Just a Line eliminate this long latency, keeping the E2E latency short.

4.3 Initial Synchronization

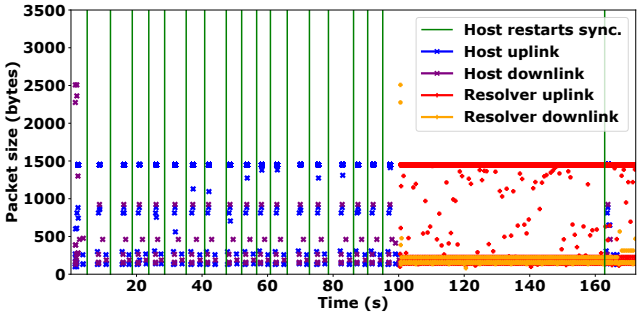
Finally, we study the initial synchronization in Just a Line. Fig. 5a shows the CDF of the duration of the initial synchronization phase over all runs. As a reference, we also plot the CDF of the E2E latency of Cloud Anchor, which involves the synchronization components (upload and processing of visual SLAM data) in every virtual object placement. Surprisingly, Fig. 5a shows that the initial synchronization phase in Just a Line can be very long, from 5.6 s up to 309 s, much longer than the E2E latency of Cloud Anchor (3.8-7.1 s).

Understanding high initial synchronization delay. We initially hypothesized that Just a Line might send larger messages during the initial synchronization phase compared to Cloud Anchor, resulting in a longer time to upload the messages and complete the synchronization. To examine this hypothesis, in Figs. 5b, 5c, we plot the CDFs of all the total bytes uploaded and downloaded, respectively, during Just a Line's synchronization phase. The downlink data size, from the cloud to the host/resolver, is very small, up to 55 KB to the host and up to 140 KB to the resolver (Fig. 5c). In contrast, the uplink data size, from the host/resolver to the cloud, is much larger, up to 7.5 MB from the host, and up to 12 MB from the resolver (Fig. 5b). The size of the uplink messages in Just a Line is larger than the size of the same messages in Cloud Anchor in the case of the host (up to 5 MB in [13]) and much larger in the case of the resolver (only 0.05-0.12 MB in [13]). Nonetheless, the much longer synchronization phase in Just a Line compared to the E2E latency in Cloud Anchor cannot be merely explained by the relative difference in the message size between the two apps.

We conjecture that the root cause of the much longer synchronization duration in Just a Line lies in the different ways the two



(a) Short synchronization example.



(b) Long synchronization example with multiple retries.

Figure 6: Examples of Just-a-line synch. timelines.

apps create synchronization anchors. In phase 1a of Cloud Anchor, the user of the host device needs to first tap the screen, and Cloud Anchor uses this tap to perform a hit-test [5] to find the intersections between the real-world 3D geometry and a virtual ray consisting of an origin and direction. The intersection is then used by the host to create an anchor. In our experiments, this way of creating anchors always succeeded, e.g., even when the phones faced a busy street, as Cloud Anchor always highlighted the empty sidewalk near the 5G BS, thereby the anchor was easily resolved. In contrast, Just a Line tries to automatically create and place the anchor. We found that this approach succeeded only when the phones faced a quiet scene (Fig. 6a). But when the phones faced the same busy street, the synchronization failure rate was very high, resulting in many retries which made the synchronization time as long as 300 s (Fig. 6b).

5 RELATED WORK

Unlike single-user AR (e.g., [10, 11, 15]), there have been very few works on multi-user AR. A few works [17, 20] focus on application layer sharing while our work focuses on the impact of the network on multi-user AR performance. In contrast to [8, 13], which study multi-user AR performance over LTE and 5G mmWave, respectively, using Cloud Anchor as a reference app, in this work, we study multi-user AR performance using another app, Just a Line. Our findings are very different from the findings in [8, 13] and reveal the power of asynchronous SLAM in providing high QoE for multi-user cloud-based AR apps over both LTE and 5G. A few recent works study edge-assisted [12, 19] or P2P-based [18] multi-user AR. In contrast, our work focuses on cloud-assisted multi-user AR, which is the default approach in most popular AR apps on the market.

6 CONCLUSION

In this paper, we revisited a key question in 5G edge computing: can 5G support multi-user AR? We studied Just a Line, a popular multi-user AR app different from the one used in prior studies, and found it achieves E2E multi-user interaction latency in the order of 100s of ms over both LTE and 5G, making real-time multi-user interactions feasible. Our detailed measurement study of Just a Line over cellular networks reveals that asynchronous SLAM, i.e., periodic SLAM updates in the background, used in Just a Line is what enables its real-time multi-user interaction latency, significantly lower than in Cloud Anchor, which uses synchronous SLAM, i.e., blocking SLAM update before each user interaction. Our study shows asynchronous SLAM is key to enabling continuous real-time user interaction in multi-user AR.

ACKNOWLEDGMENTS

This project is supported in part by NSF grant 2112778-CNS.

REFERENCES

[1] 2021. Fundamental concepts of ARCore. https://developers.google.com/ar/discover/concepts. (2021).
[2] Online. AT&T integrates 5G with Microsoft Azure to enable next-generation solutions on the edge. (Online). https://tinyurl.com/mu24dxut
[3] Online. *Google Cloud Anchor.* https://developers.google.com/ar/develop/java/cloud-anchors/overview-android
[4] Online. *Google Firebase.* https://firebase.google.com/
[5] Online. *Hit-tests place virtual objects in the real world.* https://developers.google.com/ar/develop/hit-test
[6] Online. *Just a line.* https://experiments.withgoogle.com/justaline
[7] Online. Verizon teams with NFL, AWS to showcase 5G edge. (Online). https://tinyurl.com/2s4bh4rb
[8] Kittipat Apicharttrisorn, Bharath Balasubramanian, Jiasi Chen, Rajarajan Sivaraj, Yi-Zhen Tsai, Rittwik Jana, Srikanth Krishnamurthy, Tuyen Tran, and Yu Zhou. 2020. Characterization of Multi-User Augmented Reality over Cellular Networks. In *Proc. of IEEE SECON.*
[9] Kittipat Apicharttrisorn, Jiasi Chen, Vyas Sekar, Anthony Rowe, and Srikanth Krishnamurthy. 2022. Breaking edge shackles: Infrastructure-free collaborative mobile augmented reality. In *Proc. of ACM SenSys.*
[10] Kittipat Apicharttrisorn, Xukan Ran, Jiasi Chen, Srikanth V. Krishnamurthy, and Amit K. Roy-Chowdhury. 2019. Frugal Following: Power Thrifty Object Detection and Tracking for Mobile Augmented Reality. In *Proc. of ACM SenSys.*
[11] Kaifei Chen, Tong Li, Hyung-Sin Kim, David E. Culler, and Randy H. Katz. 2018. MARVEL: Enabling Mobile Augmented Reality with Low Energy and Low Latency. In *Proc. of ACM SenSys.*
[12] Aditya Dhakal, Xukan Ran, Yunshu Wang, Jiasi Chen, and K.K. Ramakrishnan. 2022. SLAM-Share: Visual Simultaneous Localization and Mapping for Real-time Multi-user Augmented Reality. In *Proc. of ACM CoNEXT.*
[13] Moinak Ghoshal, Pranab Dash, Zhaoning Kong, Qiang Xu, Y. Charlie Hu, Dimitrios Koutsonikolas, and Yuanjie Li. 2022. Can 5G mmWave support Multi-User AR?. In *Passive and Active Measurement (PAM).*
[14] Moinak Ghoshal, Z. Jonny Kong, Qiang Xu, Zixiao Lu, Shivang Aggarwal, Imran Khan, Yuanjie Li, Y. Charlie Hu, and Dimitrios Koutsonikolas. 2022. An In-Depth Study of Uplink Performance of 5G mmWave Networks. In *Proc. of ACM SIGCOMM 5G-MeMU.*
[15] Luyang Liu, Hongyu Li, and Marco Gruteser. 2019. Edge Assisted Real-time Object Detection for Mobile Augmented Reality. In *Proc. of ACM MobiCom.*
[16] Arvind Narayanan*, Xumiao Zhang*, Ruiyang Zhu, Ahmad Hassan, Shuowei Jin, Xiao Zhu, Denis Rybkin, Dustin Zhang, Michael Yang, Z. Morley Mao, Feng Qian, and Zhi-Li Zhang. 2021. A Variegated Look at 5G in the Wild: Performance, Power, and QoE Implications. In *Proc. of ACM SIGCOMM.*
[17] Xukan Ran, Carter Slocum, Maria Gorlatova, and Jiasi Chen. 2019. ShareAR: Communication-Efficient Multi-User Mobile Augmented Reality. In *Proceedings of ACM HotNets.*
[18] Xukan Ran, Carter Slocum, Yi-Zhen Tsai, Kittipat Apicharttrisorn, Maria Gorlatova, and Jiasi Chen. 2020. Multi-User Augmented Reality with Communication Efficient and Spatially Consistent Virtual Objects. In *Proc. of ACM CoNEXT.*

[19] Pei Ren, Xiuquan Qiao, Yakun Huang, Ling Liu, Calton Pu, Schahram Dustdar, and Jun-Liang Chen. 2020. Edge AR X5: An Edge-Assisted Multi-User Collaborative Framework for Mobile Web Augmented Reality in 5G and Beyond. *IEEE Transactions on Cloud Computing* (2020).

[20] Wenxiao Zhang, Bo Han, Pan Hui, Vijay Gopalakrishnan, Eric Zavesky, and Feng Qian. 2018. CARS: Collaborative Augmented Reality for Socialization. In *Proc. of ACM HotMobile*.