Analysis of Cellular Network Latency for Edge-Based Remote Rendering Streaming Applications

Zhehui Zhang
University of California, Los Angeles
zhehui@cs.ucla.edu

Shu Shi
AT&T Labs-Research
shushi@research.att.com

Varun Gupta
AT&T Labs-Research
vgupta@research.att.com

Rittwik Jana
AT&T Labs-Research
rjana@research.att.com

ABSTRACT

Recent advances in low-latency streaming cloud technology have enabled remote rendering applications such as Google Stadia and Occulus VR. Edge-clouds located close to the end-user are expected to play an important part in improving user experience for remote rendering application. In this paper, we investigate network latency in edge-based remote rendering over LTE networks. We quantify network latency and identify the roadblocks in deploying remote rendering on existing cellular networks with extensive measurements. We show that crossing traffic could severely deteriorate latency performance and network side resource management can solve the issue. We also find the impact of various radio configurations and scheduling algorithm in LTE networks. Our results shed light on how the network side and client side can cooperatively reduce network latency for emerging applications.

CCS CONCEPTS

• **Networks** → **Network protocol design**; **Network performance evaluation**.

KEYWORDS

mobile VR, latency, mobile edge cloud

ACM Reference Format:
Zhehui Zhang, Shu Shi, Varun Gupta, and Rittwik Jana. 2019. Analysis of Cellular Network Latency for Edge-Based Remote Rendering Streaming Applications. In *ACM SIGCOMM 2019 Workshop on Networking for Emerging Applications and Technologies (NEAT'19), August 19, 2019, Beijing, China.* ACM, New York, NY, USA, 7 pages. https://doi.org/10.1145/3341558.3342199

1 INTRODUCTION

The emergence of Mobile Edge Clouds (MEC, also known as Multi-access Edge Cloud) has inspired researchers to utilize the computing resources on MEC to improve the performance of various mobile applications, such as cloud gaming [14], Virtual Reality (VR) [15, 18, 28], and 360° video streaming [19, 25, 26]. The common idea behind these approaches is to offload the computation from the mobile device to the cloud server via a remote rendering system. The server in the cloud processes the input content, performs expensive graphics rendering, and encodes the result scenes into a video stream. The mobile client simply displays video frames and forwards the user control signals back to the server. Such a remote rendering approach can effectively deliver the visual quality that high-end workstations can achieve to low-end mobile devices with a simple server-client based architecture. However, it also increases the overall system response time and makes the user experience largely depend on the network latency between server and client. Since MEC was designed to push the cloud resources closer to the mobile user and significantly reduce the latency, deploying the rendering server on MEC provides a unique opportunity to meet the requirements of the most latency-sensitive applications such as VR and gaming.

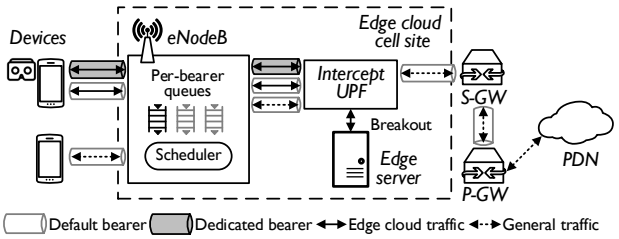


Figure 1: Edge based remote rending over LTE

Previous studies have shown promising results of remote rendering from MEC [28]. Some existing work [15, 18] only uses Wi-Fi to evaluate system performance, while the others characterize the underlying LTE performance with ping latency [14, 19]. State of the art LTE-VR [28] uses LTE traces to breakdown the network latency and analyze the potential latency bottleneck. However, this analysis is based on an emulated edge testbed rather than a real one. A comprehensive performance characterization of LTE and MEC along with various factors that affect is still missing. It is urgent for us to study end-to-end network latency rather than wireless part. Moreover, LTE-VR did not study how network side bearer setting and scheduling algorithms would affect network latency.

NEAT'19, August 19, 2019, Beijing, China

ACM ISBN 978-1-4503-6876-6/19/08…$15.00
https://doi.org/10.1145/3341558.3342199

In this paper, we use a real MEC testbed to deploy a remote rendering system and stream real-time video traffic over 4G/LTE networks. Our edge cloud server directly connects to LTE network as shown in Figure 1. The mobile device communicates with the edge server through the LTE network. In edge based scheme, the server is directly connected to the gateways in the LTE network. We breakdown the end-to-end latency and study how different environment settings, radio configurations, and scheduling algorithms affect system performance.

Our findings can be summarized as following:

- The majority of the end-to-end latency comes from the network. The network latency for a full video frame varies significantly over time. We compare with the ping latency and identify the roadblocks.
- While crossing traffic from the same phone or other phones makes the latency even worse, this can be eliminated by enforcing service differentiation for prioritized traffic. We validate it by configuring bearer with different quality class identifiers (QCI) on the commercial LTE testbed.
- Applying different scheduling algorithms helps further reduce latency and enhance overall system throughput when multiple users are streaming simultaneously. We quantify latency under three mainstream scheduling algorithm and analyze the tradeoffs.

2 BACKGROUND AND RELATED WORK

2.1 Remote Rendering Systems

The idea of offloading rendering computation to a remotely connected server was initially proposed to share graphics processing resources over networks when computers were not powerful to run graphics rendering. A survey [27] summarized how remote rendering has been evolving over years. Recently, the adoption of remote rendering mainly focuses on cloud gaming and mobile VR.

Cloud Gaming: By moving the game rendering to the cloud, cloud gaming enables users to play the latest games on any device any time without expensive game console hardware. However, a satisfying gaming experience requires a less than 100 ms interaction latency [9], which is defined as the time it takes from the user generating a control signal to the corresponding visual feedback showing on the screen. OnLive [9], the pioneer of this industry once claimed to achieve 80 ms latency but the measurement experiments [11] showed a much higher observed latency even with a wired broadband network connection. A study [14] measured the latency performance of running GamingAnywhere [11] (a popular open source cloud gaming platform) over mobile networks. However, the test system did not actually deploy the game server on a MEC and used the ping latency to an edge server to approximate the network latency. Even with this simplification, the results show that the end-to-end interaction latency over mobile networks can barely meet the 100 ms requirement. Outatime [16] is the state-of-the-art in this area and proposes speculation execution to enable systems tolerate longer latency up to 256 ms.

Mobile VR: The emerging development of Virtual Reality (VR) is expected to expand the market size of VR headsets to over 34 billion within five years[24]. There are two types of headsets currently available in the market: tethered VR systems (e.g., Oculus Rift) that requires a wired connection between the headset and a powerful workstation, and mobile VR (e.g., Samsung GearVR) that relies on the mobile hardware within the headset to perform rendering and display. Mobile VR provides the ultimate mobility but cannot match the rendering quality offered by tethered VR due to the performance gap between mobile and desktop GPU. Several works have proposed to improve mobile VR quality by adopting remote rendering. However, compared to cloud gaming, mobile VR has a even more stringent latency requirement. Research [5] shows that VR applications require the latency between any head movement and displaying the corresponding visual feedback on the screen to be less than 25 ms.

Luyang et al. [18] designed a simple remote rendering framework to move all rendering computation to a remote server and optimized the system to achieve up to 90 Hz refresh rate via WiGig. Mangiante et al. [19] targeted a specific VR application: 360 degree video streaming, and proposed to use MEC for FoV rendering. However, the work did not present a full system or report the actual end-to-ed latency that we care about. Furion [15] separated the background scene from the foreground objects and used the remote server to render background scene only. Enabling clients to render locally makes the responsiveness independent from network latency, and therefore allows the rendering server to be pushed further away from the mobile client. However, the optimization of Furion only applies to the foreground object movements and the head movements that change only the viewpoint orientation. Furion relies on pre-fetching to accommodate other motion that changes viewpoint location (e.g., moving the avatar around in the virtual world), and the network bandwidth and latency again play a significant role.

2.2 LTE Resource Management

In this paper, we discuss how LTE resource management impact user quality of experience for remote rendering streaming application. LTE resource management coordinates network resources between different users to improve resource utilization and user experience. Figure 1 shows simplified LTE network architecture, consisting of the access network (the base stations) and the core network (only gateways shown in the figure). In LTE networks, resource management is enforced by both the base station and the core network. The base station allocates radio resources to a user for data transmission in the format of specific frequency slots and time slots. In the core network, the gateways handle resource allocation among base stations.

In LTE networks, resources are allocated at the per-bearer level. The Bearer is the logical channel carrying data packets requiring the same service quality. There are two types of bearers in LTE: default bearers and dedicated bearers. When a user registers in the EPC, a default bearer is built promptly to offer best-effort data service. To improve user experience for prior packet flows like Voice over LTE, a dedicated bearer is set up on demand. The operator classifies packet flows based on traffic flow templates [2]. After the EPC identifies prior packet flows, dedicated bearer setup procedure is triggered and the dedicated bearer is configured a specific QoS Class Identifier (QCI) and bearer settings. The Policy

Table 1: Standardized QCI characteristics[1]

QCI	Type	Priority	Delay budget (ms)	Loss rate
3	GBR	3	50ms	10^{-3}
6	Non-GBR	6	300ms	10^{-6}
9	Non-GBR	9	300ms	10^{-3}

and Charging Rule Function (PCRF) in the EPC is responsible for determining appropriate QoS parameters for each bearer. The bearer configurations are attached with each bearer as QCI, with a detailed specification in [1]. The QCI specifies resource type traffic priority, delay budget, and loss rate. We select three representative ones as shown in Table 1. In total there are 17 QCI, with priority ranging from 0.5 to 9, where 9 denotes the lowest priority. Guaranteed Bit Rate (GBR) bearers are served with higher priority than non-GBR bearers. Delay budget is the allowed maximum one-way network delay in the LTE network, defined from PDN to UE. Loss ratio is allowed maximum packet loss ratio.

After bearer setup, LTE entities schedule resources based on bearer configurations. Resources are first granted to GBR bearer first, then non-GBR bearers share the remaining while considering their priority. The scheduling algorithm is not specified by 3GPP so operators have implemented proprietary scheduling algorithms in practice. According to [8], though most of the operators use proportional fair for resource allocation, their implementation takes various factor into consideration. There is plenty of research on how to schedule resources to maximize efficiency and guarantee user fairness [7]. However, researchers emphasize the theoretical merits while arguably discuss real-world impact. Since operator side scheduling is proprietary, testing on real testbed can represent real-world scenarios.

Although most bearer settings are specified in [1], the operators implement radio resource management with different configurations. There are various configurations specific to each function in LTE. Among these configurations, packet corruption recovery related configurations are verified to affect latency significantly based on the previous study in [28]. Specifically, we study two configurations at Radio Link Control (RLC) layer, RLC mode and retransmission timer. RLC layer offers optional reliable packet delivery to upper layers by automatic repeat request. RLC mode determines whether the reliable transmission is required. The retransmission timer is the timer to trigger retransmission, which might prolong latency. In this paper, we show how radio configuration affect latency and scheduling algorithm affect latency. We compare configurations for different QCI bearers and find following radio layer packet recovery related configurations.

3 MEASUREMENT SETUP

Testbed setup We evaluate end-to-end latency with an edge-based remote rendering video streaming system on a private LTE testbed as shown in Figure 1. The LTE testbed consists of an eNodeB with LTE release 12, a data interceptor and a co-located edge server. The eNodeB manages two cells running on band 30 with 10MHz channel bandwidth. The packet interceptor user plane function (UPF) implements local breakout and routes edge cloud traffic to the edge server. Other general traffic are routed via the regular path onwards to the SGW and PGW. The PGW is connected to the Packet Data network (PDN). The testbed establishing dedicated bearers for different QCIs and routing traffic through dedicated bearers based on IP/Port/Protocol filters. The edge server is deployed as a VM with eight CPU cores and 16GB memory. The MEC platform complies with ETSI specifications [10]. We tested with Samsung Galaxy S7 and Samsung Galaxy Note 9.

Remote Rendering Application Since our main focus is on measuring network side latency, we develop a simplified application to simulate the behavior of the thin-client system in [18]. On the server side, we read a video file (H.264, 1080p, using I and P frame only) and store every video frame in memory. Each frame size ranges from 41 KB to 368 KB with an average of 58.3 KB. The client sequentially transmits a fetch request for one frame and the server replies with the frame data upon receiving the request. After receiving the full frame, the client uses MediaCodec that calls hardware codec APIs to decode the video and display on the screen. We use TCP in our experiments to guarantee reliable delivery. In an accompanying paper, we have also observed that TCP outperforms UDP for VR delivery using MEC [25]. The client only runs the streaming application unless specified. This simplified application allows us to accurately measure the end-to-end system latency for every frame and easily separate the latency caused by network and mobile device. We consider rendering and transcoding latencies on the server to be negligible. Note that cloud platforms typically take 5ms to process one frame [21].

Data collection and analysis We collect three types of traces to quantify network latency: (i) the application logs on server and client; (ii) the LTE network signaling traces with MobileInsight [17]; and (iii) the TCP/IP traces on clients and servers with tcpdump. We measure end-to-end latency at the client side. Since the server processing latency is negligible, the overall latency is broken down into network latency and client latency. To compare client latency and network latency, we further extract network latency by timestamping when the frame request was sent and when the frame was received. In total, we analyzed 35.6 million frames traces in both testbed and simulation.

4 LATENCY MEASUREMENTS

4.1 System latency overview

We begin by looking at overall latency performance and dissect network latency from the whole. The end-to-end latency composes of network latency, server-side latency and client-side latency. Here we define network latency as the time elapsed from the packet sent to the time response received at the client side application layer. Since server-side only transmit pre-render frames in our experiments, the server side latency is negligible. As shown in Figure 2, device latency is stable compared with network latency. Also, network latency constitutes 77.5% of total latency (96.1 ms of 124.0 ms). The medians are 81 ms and 112 ms for network latency and total latency. Though device side latency depends on device capability, we believe Samsung S7 and Note 9 are mid to high performance phone models.

We then investigate what is the root cause of abnormal network latency. As compared with the ping latency, the median of ping

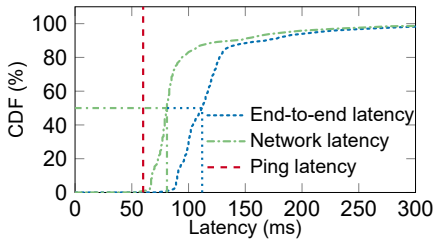


Figure 2: End-to-end latency breakdown

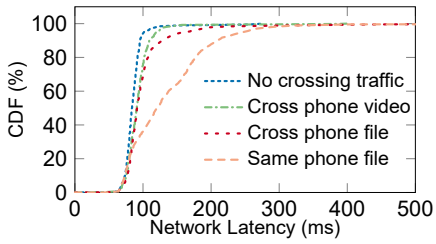


Figure 3: Latency with conflicting traffic

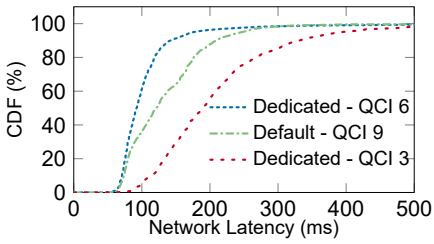


Figure 4: Latency with different QCI

latency is only 61 ms and the minimum is 31 ms. Since ping latency is collected using ICMP probing packets, the latency shall be longer than data packet. The reason why the ping latency is less than the network latency for the streaming application can be one of the followings: processing latency, transmission latency and queuing latency caused by congestion. Since there is only one device connected with the cell during testing, there should be minimum congestion. As shown in Figure 2, the minimum network latency is only 61 ms, which is far beyond standardized 4ms processing delay limits [3]. So we can safely assume the prolonged latency is transmission latency.

We further investigate the root cause of prolonged transmission latency. We prove that transmission latency is not prolonged by limited bandwidth but the retransmission. With 256QAM enabled, peak throughput is around 50.7Mbps during frame transmission. So the normal transmission latency for a single 58KB frame only takes 9 ms. In our experiments, 0.4% packets are retransmitted by TCP, which is higher than previously measured 0.06% in [12]. The retransmission delay is 154 ms on average. Packet loss can happen at wireless channel, interceptor or the end hosts. We validate that the wireless channel packet corruption are all recovered by RLC retransmission. Then we can conclude that packet loss happen at either the interceptor or the end hosts. We further check the interceptor logs and validate that no packets are discarded. So we conjecture that TCP retransmits packet because packets are lost at the end hosts.

Takeaway: Network latency is the key driver of overall latency, rather than device processing latency. Excessive network latency mostly comes from TCP retransmission rather than frame transmission.

4.2 Congestion scenarios

As discussed in the previous section, TCP retransmission prolong network latency. Another portion of network latency is queuing latency especially when the base station need schedule limited radio resources between multiple users. According to a recent study [6], most base station tends to serve only one user at a time, which add queuing latency to other waiting users. In this section, we study how congestion prolongs queuing latency.

We conducted experiments with two congestion scenarios: single phone and cross phone scenario. In the single phone scenario, the VR application and the traffic generator run on the same phone. The single phone scenario mimics the case of simultaneously downloading large files (at data rate of around 10Mbps) or streaming background video frames (at data rate of around 6.8Mbps) [15]. In the cross phone scenario, two applications run on different phones. The cross phone scenario emulates a congested cell with crossing traffic.

Experiment with testbed shows that under crossing traffic the overall latency will increase significantly. As shown in Figure 3, latency increases in both single phone scenario and cross phone scenario. In single phone scenario, per-frame network latency with crossing traffic increase by 56.9% (86.3 ms to 135.4 ms). Latency 75 percentile increase from 91 ms to 168 ms. In cross phone scenario, per-frame network latency with crossing traffic increase by 8.2% (86.3 ms to 93.4 ms) for video traffic and 19.6% (86.3 ms to 103.2 ms) for file traffic. Latency 75 percentile increase from 91 ms to 100 ms and 103 ms respectively.

The intuitive solution to avoid competition between latency-sensitive traffic and latency-insensitive traffic is to guarantee latency requirements by differentiating these two traffic and priority latency sensitive ones. In current streaming frameworks, pre-fetched based scheme is commonly adopted since it masks long latency to users [15, 23]. However, when pre-fetching fails, the user need to wait entire round trip for emergent frame request. The user tolerance difference motivates us to differentiate emergent frame request from pre-fetching frame requests.

Takeaway: Crossing traffic significantly prolongs network latency, especially for crossing traffic from the same device. Network side should differentiate emergent streaming traffic and serve with higher priority.

4.3 Radio configurations

As introduced in 2.2, each operators can customize radio configurations to guarantee service quality at different network entities. For example, different operators might assign different profiles for the same VoLTE traffic. It is often neglected by the researchers how radio configuration can make impacts beyond radio connection part. In this section, we would show how inappropriate radio configuration deteriorates latency performance. Since in 5G the layering design does not change significantly and studied radio configurations are still effective [4], the following findings still hold in 5G.

We first examine the generality of testbed radio configurations by comparing with existing configurations in operational LTE networks. We analyze traces from LTE-VR database [28], which includes 3.2 million cellular messages collected over 8 months on four major operators in the U.S. Our analysis of operational LTE networks aligns with our testbed setting. We checked configurations

from all four major operators by examining RRC signaling messages [4]. All of them use RLC AM mode for default bearer, which is the same in our testbed. In addition, the RLC re-transmission timer range from 35 ms to 60 ms in all four operators with 38 ms on average, which corresponds with 40 ms setting in the testbed. Thus we believe testbed configurations align with real-world scenarios and measurement should be consistent with experiments in the wild.

To investigate the impact of radio configurations, we consider three QoS with significantly different QoS parameters listed in Table 1. Note QCI 3 and QCI 6 are for the dedicated bearer and QCI 9 is for default bearer. We first show a comparison of two non-GBR bearers, QCI 6 and QCI 9, the former with higher priority than another. We then show a comparison of QCI 3 (GBR bearer) and QCI 9 (non-GBR bearer). Since the single phone experiments show more latency, we adopt the experiment setting of single phone generating the crossing traffic.

We first compare two non-GBR bearers setting of QCI 6 and QCI 9. We observe latency reduces by sending traffic through dedicated bearer with QCI 6 as shown in Figure 4. Per frame network latency with the dedicated bearer (QCI 9) is smaller than the default bearer (QCI 6). Average latency reduces by 26.7% (135.4 ms to 106.9 ms). The 75 percentile reduces from 177 ms to 115 ms. From the above analysis, we can draw the conclusion that network side service differentiation can reduce network latency significantly.

We then compare non-GBR bearer of QCI 9 with GBR bearer of QCI 3. The network should allocate resources to QCI 3 bearers prior to QCI 9 bearers. However, We observe that sending traffic with QCI 3 does not reduce latency. Average latency increases by 55.6% (135.4 ms to 210.7 ms). The root cause is improper radio configurations under GBR bearer that prolong packet loss recovery latency. RLC layer configuration for QCI 3 bearer is unacknowledged (UM) mode in testbed. RLC mode defines whether RLC layer recovery is enabled. Since RLC is configured as UM mode, packet loss recovery solely rely on TCP re-transmission, which could potentially prolong latency. We also get testbed side limited logs to verify our findings. Based on eNodeB side records, the queuing latency for dedicated bearer is 8.2 ms while for default bearer is 16.0 ms. Default dearer IP packet discard rate is 0 while dedicated bearer IP packet discard rate is 1.2 Kbps.

Takeaway: Prioritizing streaming traffic can help to reduce latency only when radio configurations are properly configured. Configuration sacrificing reliability help to reduce queuing latency but increase transport layer re-transmission, which prolong overall latency even under good channel quality.

4.4 Scheduling algorithm

The scheduling algorithm is essential to resource allocation. The base station dynamically schedules current resources based on user demands, channel conditions and other factors.

To understand latency under various scheduling algorithm in a large scale setting, we further evaluate our design in a simulation platform [22]. We measure latency with and without our design under various scheduling algorithms to show the generality of our design.

Our simulation setting is similar to [22]. Users are randomly distributed and follow a random walk mobility pattern. We simulate with 2 to 20 users per cell and 4 cells. We repeat each simulation 10 times. We use the same user traffic distribution, where each user runs video traffic and a constant bit rate traffic generator. To avoid TCP retransmission delay complicates latency comparison between different scheduling algorithms, we use UDP traffic. As we observed in experiment results, the priority factor can influence streaming performance with the same bearer setting. Specifically, we use RLC UM mode to assure the streaming performance is solely controlled by the scheduling priority.

We compare network latency for three scheduling algorithms, proportional fair, Modified LDWF, and EXP rule. In proportional fair scheduling, resources are allocated based on user metric α, $\alpha = r_i/\bar{R}_i$, where r_i is the estimated bitrate and $\bar{R}_i$ is the historical average bit rate for user i . The proportional fair is the most commonly used scheduling algorithm because of its convenience and efficiency [8]. In Modified LDWF, a delay related priority factor $w_i = -\frac{\log(Pkt\_loss)}{Target\_delay} HOL\_delay$ is added so the metric α is $\alpha = w_i \frac{r_i}{\bar{R}_i}$. In EXP rule, an exponential function of the end-to-end delay is adopted, where the metric became $exp(\frac{w_i - \bar{w}}{1+\sqrt{\bar{w}}})$ and $\bar{w}$ is the average of w_i for all users. The EXP rule is shown to have the best performance even though the complexity is high [22].

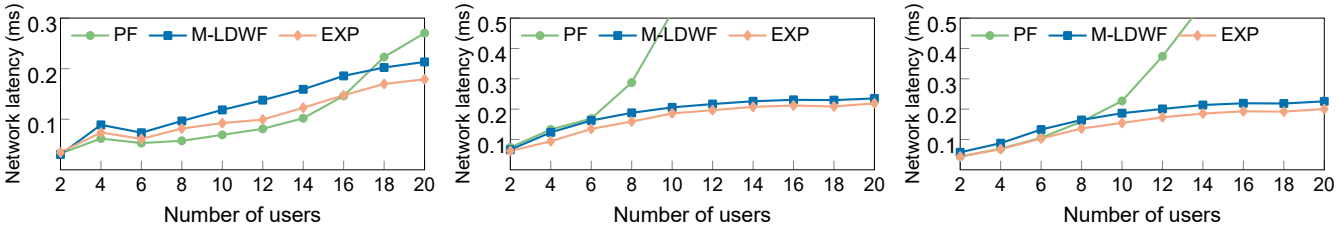
As shown in Figure 5a, average latency increases as the number of users increases. The EXP performs best compared with PF and M-LDWF. If crossing traffic is added, latency further increases, especially with PF scheduling algorithm. As shown in Figure 5b, with crossing traffic, average latency increases by 752.3% with PF (69.1 ms to 519.9 ms), 73.7% with M-LWDF (118.5 ms to 205.8 ms) and 102.1% with EXP algorithm (92.4 ms to 186.7 ms) in 10 users case. Latency increases most in PF algorithm compared with other two scheduling algorithm.

We also check how dedicated bearer affect latency. With PF algorithm, prioritizing streaming traffic barely affect latency. PF algorithm allocates resources based on user historical link rate. If the user is with bad signal strength, it is hard for the user to get a fair share of radio resources. With M-LWDF or EXP algorithm, prioritizing streaming traffic can both reduce latency. As shown in Figure 5c, prioritizing streaming traffic with dedicated bearer can reduce latency by 43.7% with PF (519.9 ms to 226.9 ms), 8.8% with M-LWDF (205.8 ms to 186.7 ms) and 16.7% with EXP algorithm (186.7 ms to 154.6 ms) in 10 users case. Since streaming traffic is identified with higher priority, more resources are allocated to streaming traffic. Especially when packets are approaching delay limits, the priority metric will become very high.

Takeaway: M-LWDF and EXP serve streaming traffic with less latency than PF since delay limit is considered in scheduling. By incorporating priority factor for different QCI, we can further reduce network latency for streaming traffic.

5 DISCUSSION

Our design applies to various streaming applications with flows that require different quality of service. For example, pre-fetching based VR puts different weights on different video frames based



(a) wo/ crossing traffic (b) w/ crossing traffic - default bearer (c) w/ crossing traffic - dedicated bearer

Figure 5: Simulation results

on whether that frame is pre-fetched or emergent. With the pre-fetching algorithm, the state of the art can guarantee on-time delivery of VR frames with probability >90% [23] when end-to-end latency does not exceed 200ms. (Linear regression-based viewpoint-prediction, good and fair signal strength, tolerable network jitters) Therefore, our design would help to reduce latency for emergent frames. Our method is compliant to LTE standard and compatible with existing flow classification. The network side relies on traffic flow template to identify service [2]. According to the specification, there are 13 packet filter can be added to filter a specific traffic type, including IP address, port number, protocol type, etc. Therefore, it depends on the operator to specify certain packet filters to specify emergent streaming traffic.

There are several practical issues to consider. First, considering delay limits at the scheduler would incur computation delay. But the prevalence of GPU is verified to increase the scheduler computation power [13]. Second, prioritized VR traffic competes with other dedicated bearer traffic. In the current operational LTE network, only VoLTE is using dedicated bearer for transmission. A naive solution is to prioritize VoLTE over VR traffic. The priority factor shall be cautiously designed so that voice call quality will not be affected. Third, setting up dedicated bearer on demand is time-consuming, which would take an extra round trip from the base station to the PCRF. To avoid initialization delay, we suggest the network to initiate dedicated bearer as any streaming traffic is detected. This could be a greedy solution though the risk of abusing dedicated bearer increases.

The final concern is how to avoid user abusing dedicated bearer. In this direction, the intuitive idea is to borrow some ideas from current VoLTE implementation. In current VoLTE, the user can only use VoLTE service after authenticated. Since call signals are directly processed by the chipset, the OS has no access to inject data in voice packets. As for signaling packets, unprivileged apps can easily obtain the VoLTE interface information and inject non-VoLTE data packets. But the exposed vulnerabilities may only exist for certain protocols and ports. Learned from previous VoLTE design, we should require the OS to employ permission control. The practical approach is to distinguish network interface dedicated to emergent streaming traffic and other privileged traffic for Internet data access. On the hardware side, the chipset should also check the traffic coming from the software's network interface and avoid unprivileged app use dedicated bearer. This is possible in dedicated chipsets for emerging applications [20].

6 CONCLUSION

In this paper, we analyzed network latency in edge based remote rendering streaming applications over real LTE testbed. We identified the roadblocks in further reducing latency with extensive measurements. We quantitatively showed that crossing traffic could severely deteriorate latency performance and network side resource management can solve the issue. We also investigated the impact of various radio configurations and scheduling algorithm.

REFERENCES

[1] 3GPP. 2018. TS23.203: Policy and charging control architecture (Release 15). http://www.3gpp.org/ftp/Specs/html-info/23203.htm
[2] 3GPP. 2018. TS24.008: Core network protocols; Stage 3 (Release 15). http://www.3gpp.org/ftp/Specs/html-info/24.008.htm
[3] 3GPP. 2018. TS36.321: Medium Access Control (MAC). http://www.3gpp.org/ftp/Specs/html-info/36321.htm
[4] 3GPP. 2019. TS36.331: Radio Resource Control (RRC). http://www.3gpp.org/ftp/Specs/html-info/36331.htm
[5] Michael Abrash. 2014. What VR Could, Should, and Almost Certainly Will Be Within Two Years. http://media.steampowered.com/apps/abrashblog/AbrashDevDays2014.pdf
[6] Arjun Balasingam, Manu Bansal, Rakesh Misra, Kanthi Nagaraj, Rahul Tandra, Sachin Katti, and Aaron Schulman. 2019. Detecting if LTE is the Bottleneck with BurstTracker . In *Proc. ACM MobiSys'19.*
[7] Francesco Capozzi, Giuseppe Piro, Luigi Alfredo Grieco, Gennaro Boggia, and Pietro Camarda. 2013. Downlink packet scheduling in LTE cellular networks: Key design issues and a survey. *IEEE Communications Surveys & Tutorials* 15, 2 (2013), 678–700.
[8] Jiasi Chen, Rajesh Mahindra, Mohammad Amir Khojastepour, Sampath Rangarajan, and Mung Chiang. 2013. A Scheduling Framework for Adaptive Video Delivery over Cellular Networks. In *Proc. ACM MobiCom'13.* ACM, 389–400.
[9] Chen, S. and Chang, Y. and Tseng, P. and Huang, C. and Lei, C. 2018. Cloud Gaming Latency Analysis. http://www.iis.sinica.edu.tw/~swc/onlive/onlive.html
[10] ETSI. 2019. Multi-access Edge Computing (MEC); Framework and Reference Architecture . https://www.etsi.org/deliver/etsi_gs/MEC/001_099/003
[11] Chun-Ying Huang, Kuan-Ta Chen, De-Yu Chen, Hwai-Jung Hsu, and Cheng-Hsin Hsu. 2014. GamingAnywhere: The first open source cloud gaming system. *ACM Transactions on Multimedia Computing, Communications, and Applications (TOMM)* 10, 1s (2014), 10.
[12] Junxian Huang, Feng Qian, Yihua Guo, Yuanyuan Zhou, Qiang Xu, Zhuoqing Mao, Subhabrata Sen, and Oliver Spatscheck. 2013. An in-depth study of LTE: effect of network protocol and application behavior on performance. 363–374.
[13] Yan Huang, Shaoran Li, Y. Thomas Hou, and Wenjing Lou. 2018. GPF: A GPU-based Design to Achieve 100 μs Scheduling for 5G NR. In *Proc. ACM MobiCom'18.*
[14] Teemu Kämäräinen, Matti Siekkinen, Antti Ylä-Jääski, Wenxiao Zhang, and Pan Hui. 2017. A measurement study on achieving imperceptible latency in mobile cloud gaming. In *Proceedings of the 8th ACM on Multimedia Systems Conference.* ACM, 88–99.
[15] Zeqi Lai, Y. Charlie Hu, Yong Cui, Linhui Sun, and Ningwei Dai. 2017. Furion: Engineering High-Quality Immersive Virtual Reality on Today's Mobile Devices. In *Proc. ACM MobiCom'17.*
[16] Kyungmin Lee, David Chu, Eduardo Cuervo, Johannes Kopf, Yury Degtyarev, Sergey Grizan, Alec Wolman, and Jason Flinn. 2015. Outatime: Using speculation to enable low-latency continuous interaction for mobile cloud gaming. In *Proceedings of the 13th Annual International Conference on Mobile Systems, Applications,*

and Services. ACM, 151–165.

[17] Yuanjie Li, Chunyi Peng, Zengwen Yuan, Jiayao Li, Haotian Deng, and Tao Wang. 2016. MobileInsight: Extracting and Analyzing Cellular Network Information on Smartphones. In *Proc. ACM Mobicom'16*.

[18] Luyang Liu, Ruiguang Zhong, Wuyang Zhang, Yunxin Liu, Jiansong Zhang, Lintao Zhang, and Marco Gruteser. 2018. Cutting the Cord: Designing a High-quality Untethered VR System with Low Latency Remote Rendering. In *Proc. ACM MobiSys'18*.

[19] Simone Mangiante, Guenter Klas, Amit Navon, Zhuang GuanHua, Ju Ran, and Marco Dias Silva. 2017. Vr is on the edge: How to deliver 360 videos in mobile networks. In *Proceedings of the Workshop on Virtual Reality and Augmented Reality Network*. ACM, 30–35.

[20] Lucas Matney. 2018. Qualcomm introduces a dedicated chip for mass market AR/VR headsets. https://techcrunch.com/2018/05/29/qualcomm-introduces-a-dedicated-chipset-for-mass-market-ar-and-vr-headsets/

[21] NVIDIA. 2017. Nvidia GRID cloud gaming platform. http://www.nvidia.com/object/cloud-gaming.html.

[22] G. Piro, L. A. Grieco, G. Boggia, F. Capozzi, and P. Camarda. 2011. Simulating LTE Cellular Systems: An Open-Source Framework. *IEEE Trans. on Vehicular Technology* 60, 2 (2011), 498–513.

[23] Feng Qian, Bo Han, Qingyang Xiao, and Vijay Gopalakrishnan. 2018. Flare: Practical Viewport-Adaptive 360-Degree Video Streaming for Mobile Devices. In *Proc. ACM MobiCom'18*.

[24] Research and Markets. 2018. Augmented Reality and Virtual Reality Market by Offering. https://www.researchandmarkets.com/research/qq837j/global_augmented

[25] Shu Shi, Varun Gupta, Micheal Hwang, and Rittwik Jana. 2019. Mobile VR on Edge Cloud: A Latency-Driven Design. In *Proc. ACM MMSys'19*.

[26] Shu Shi, Varun Gupta, and Rittwik Jana. 2019. Freedom: Fast Recovery Enhanced VR Delivery Over Mobile Net-works. In *Proc. ACM MobiSys'19*.

[27] Shu Shi and Cheng-Hsin Hsu. 2015. A Survey of Interactive Remote Rendering Systems. *ACM Comput. Surv.* 47, 4 (May 2015), 57:1–57:29.

[28] Zhaowei Tan, Yuanjie Li, Qianru Li, Zhehui Zhang, Zhehan Li, and Songwu Lu. 2018. Enabling Mobile VR in LTE Networks: How Close Are We?. In *ACM SIGMETRICS'18*.