Device-Based LTE Latency Reduction at the Application Layer

Zhaowei Tan and Jinghao Zhao, *UCLA;* Yuanjie Li, *Tsinghua University;*
Yifei Xu, *Peking University;* Songwu Lu, *UCLA*

https://www.usenix.org/conference/nsdi21/presentation/tan

This paper is included in the
Proceedings of the 18th USENIX Symposium on
Networked Systems Design and Implementation.
April 12–14, 2021

978-1-939133-21-2

Device-Based LTE Latency Reduction at the Application Layer

Zhaowei Tan[1], Jinghao Zhao[1], Yuanjie Li[2], Yifei Xu[3], Songwu Lu[1]
[1]*UCLA*, [2]*Tsinghua University*, [3]*Peking University*

Abstract

We design and implement LRP, a device-based, standard-compliant solution to latency reduction in mobile networks. LRP takes a data-driven approach. It works with a variety of latency-sensitive mobile applications without requiring root privilege, and ensures the latency is no worse than the legacy LTE design. Using traces from operational networks, we identify all elements in LTE uplink latency and quantify them. LRP designates small dummy messages, which precede uplink data transmissions, thus eliminating latency elements due to power-saving, scheduling, etc. It imposes proper timing control among dummy messages and data packets to handle various conflicts. The evaluation shows that, LRP reduces the median LTE uplink latency by a factor up to 7.4× (from 42ms to 5ms) for four tested apps over five mobile carriers.

1 Introduction

Low latency is critical to the proper functioning of various delay-sensitive mobile applications, such as mobile VR/AR, mobile gaming, mobile sensing, mobile machine learning, and emerging robot/drone-based image/speech recognition [22, 29, 36, 40]. These applications typically run on 4G LTE and 5G mobile networks, which offer ubiquitous access and seamless service. In this work, we study how to reduce network latency over LTE networks for such applications in the connected state. This complements the work that reduces the connection setup latency [33].

Many emergent latency-sensitive mobile apps differentiate themselves for their heavier *uplink* data transfer (e.g., user motion control, sensory data, and live camera streaming) from the device to the infrastructure. Our experiments further reveal that, uplink latency contributes to a large portion of overall latency in tested apps over operational LTE (§3.1). Reducing the uplink latency is thus as important as reducing the downlink latency. While the downlink transfer has been extensively optimized, the uplink data transfer is less studied.

Reducing the uplink LTE latency turns out to be more challenging than the downlink latency. The uplink data transfer in LTE is more complex, since it involves the interaction between the device and the network. It adopts the feedback-based device power-saving, base station-controlled scheduling for data transfer, on-demand radio resource allocation, retransmissions, etc. This results in more network latency sources with complex interactions. Traditional infrastructure-based solutions fall short to optimize them due to the lack of knowledge on device-side application usage patterns.

We design and implement LRP, a device-based, software-only LTE latency reduction solution that is readily usable for *every* commodity smartphone device. A salient feature of LRP is that it does not require root/system privilege, firmware modification or hardware change. It is thus applicable to every off-the-shelf commodity device, including Android and iOS. LRP explores the application-driven network latency reduction at the device, which complements those existing infrastructure-centric solutions that are good at downlink latency reduction. LRP focuses on reducing LTE uplink network latency, which is the bottleneck based on our empirical analysis.

The overall design of LRP takes the data-driven approach. Through analysis of operational LTE traces, we identify all elements in LTE uplink latency, and quantify them via two popular applications (§3.2). Based on the gained insights, LRP designates small dummy messages, which precede those uplink data packet transfers. It thus eliminates the latency elements due to power-saving and scheduling (§5). To make this conceptually simple idea work, LRP infers critical LTE parameters at the application layer, and performs proper timing control among dummy messages and data packet streams. To reduce the overall latency, LRP further resolves the conflict that arises among dummy messages, and avoids the conflict between data packets and dummy messages. All these solution components work at the application layer without root privilege, thus available for every off-the-shelf commodity mobile device. While LRP is mainly designed in 4G LTE, it can be readily generalized to benefit the emergent 5G (§5.5).

We implement LRP on commodity Android phones (§6). Our experiments confirm LRP's effectiveness (§7). LRP reduces the median LTE uplink latency by up to 7.4× (from

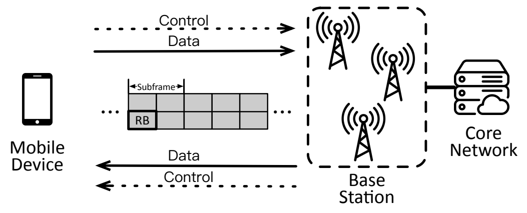


Figure 1: Network data transmission over LTE.

42ms to 5ms) for four tested applications over five mobile carriers. In any case, LRP ensures that the network latency for data transfer is no worse than the legacy LTE design. The incurred energy and data volume overhead is negligible.

2 Latency-Sensitive Mobile Apps over LTE

The mobile networks, such as 4G LTE and 5G, offer the only large-scale infrastructure that ensures universal coverage and "anytime, anywhere" access. Its infrastructure consists of radio base stations (BSes) and the core network (see Figure 1). A mobile device transfers its data with a local BS ("cell"), which covers a geographic area. The BS further relays the data to the Internet via the LTE core. A mobile device has uplink (device$\rightarrow$BS) and downlink (BS$\rightarrow$device) transmissions. In 4G LTE, data transfer uses scheduling-based mechanisms, where a BS schedules radio resources for each device in the cell for its uplink and downlink data transfer.

We next exemplify some representative latency-sensitive applications over the mobile networks.

Mobile VR A mobile virtual reality (VR) app typically involves 3D scenes and associated graphical engines [10, 15, 29, 31]. Standalone VR headsets such as Google Daydream [17] render 3D scenes locally. However, due to limited computation resources and high power consumption on mobile devices, high-quality VR applications typically need the edge/cloud servers to offload the rendering task [44]. In this client-server scheme, the mobile headsets or pads provide sensory/control data, while the server renders the 3D scene in the form of graphical frames. The server coordinates multiple devices, renders the VR graphical frames based on the device's input, and constructs the appropriate 3D scene for each given device.

• *Showcase VR prototype:* Following the above paradigm, we have built an example VR game with Unity 3D engine [42] on Android phones to study LTE latency. It has three modules: the controller at the device, the camera controller at the server, and the streaming component. The Android controller app acquires the device rotation data from the gyroscope sensor to control the in-game camera rotation. The GPS location is fed into the VR game so that the virtual character moves with the player's location updates. Upon receiving the player's sensory data, the camera controller at the server processes them and makes corresponding position and rotation movements for the virtual camera. We implement the streaming module with open-sourced libraries Unity Render Streaming [46] and WebRTC for Unity [18]. With the streaming module, the camera view is rendered and streamed back in 60FPS to the player with WebRTC. Players open the camera stream with the Web browser on the phone to get the real-time camera view.

Mobile sensing Smartphones today are equipped with multiple sensors: accelerometer, gyroscope, camera, to name a few. Many mobile sensing apps collect sensory data and upload them at runtime to the cloud for processing. For example, a localization app sends the GPS data to the cloud for realtime navigation. All such sensing apps are latency sensitive.

Mobile gaming In multi-player mobile games, the device acts as a controller that collects user motion, while the remote server processes the game logic. The server further provides proper synchronization and coordination among players. Moreover, pure cloud-based gaming (with rendering being processed in the cloud) is also trendy [29]. It is a new gaming paradigm being pushed by companies [40].

Cloud/edge-assisted machine learning Mobile apps with machine learning features (e.g., image/object recognition or speech understanding [14, 22]) also pose latency requirements. Network latency becomes a bottleneck for smart assistants, such as Alexa [7] and Siri [45]. Users may tolerate at most 200ms response time, while deep learning based local transcriptions take only 10ms [13].

Networking usage patterns by these mobile apps All the above representative mobile apps involve *frequent and regular uplink* data transfer. The mobile VR, sensing, and gaming [49, 50] applications collect data from device sensors and upload them to the server for subsequent actions. These sensors typically produce small data *periodically*. The user can only configure the sampling periodicity through the API provided by the mobile OS [21]. The machine learning based apps also have predictable traffic. They typically perform local computations with predictable latency before an uplink data transfer. For example, face recognition apps process a video frame locally using a fixed-sized neural network (NN). A user can gauge the delay based on the NN size. Emerging robotic or drone-based applications perform local tasks for a certain duration (e.g., scanning the surrounding environment for a few seconds [6]) before uploading the result. Such apps also exhibit uplink traffic that can be accurately predicted.

3 Demystifying LTE Latency in Mobile Apps

In this section, we empirically analyze where the application-perceived LTE latency stems from. We address two issues:

• *How large can LTE uplink latency be over operational 4G networks?* We use measurements to quantify it in §3.1.

• *Why is the uplink latency prohibitively high over LTE?* We break down this latency into multiple elements. We quantify their impact, identify root causes, and share insights in §3.2.

App	Latency	AT&T	T-Mobile	Verizon	Sprint
PUBG	UL Net	10.7	9.9	10.0	17.7
	DL Net	5.0	5.0	5.0	5.0
	UL/Total	68.2%	66.4%	66.7%	78.0%
VR	UL Net	N/A [1]	18.4	23.8	N/A
	DL Net	N/A	8.5	10.6	N/A
	UL/Total	N/A	68.4%	69.2%	N/A

Table 1: LTE latency (ms) for two mobile apps.

3.1 Measuring LTE Latency

We quantify the uplink latency over operational LTE networks via measurements and trace analysis.

Methodology We analyze the traces from our showcase VR game and another popular mobile application PUBG Mobile [40]. Our VR application uploads user motion packets (∼60Bytes) and receives 60FPS, 5Mbps downlink video stream. The downlink data packets are sent from the deployed server to the device over LTE. PUBG is a mobile game with frequent uplink data (∼40ms interval) and downlink responses. Both uplink and downlink packets are small (<100Bytes). The latency due to server processing is less than 1ms. The mobile devices (a Pixel 2 and a Pixel XL) run the apps. We collect both app logs and LTE signaling traces via MobileInsight [32]. We carry out our experiments over four US mobile carriers from 12/2019 to 09/2020. The tests cover static, low-mobility (∼1m/s), and high-mobility (∼30mph) cases, with varying signal strength (-120∼-80dBm).

Results We first measure the LTE uplink latency. We monitor the device buffer and compute the latency for each data packet. This information is available in the MobileInsight message "LTE MAC UL Buffer Status Internal". Despite small packet size, the uplink latency turns out to be non-negligible, as shown in Table 1. For all four carriers, the uplink latency (UL NET) ranges from 9.9-17.7ms for PUBG and 18.4-23.8ms for VR. These latency values might not meet the requirements of a number of latency-sensitive apps [5].

Who is the latency bottleneck? We further discover that, instead of downlink, the uplink latency poses as a major component in overall latency. We compute the downlink latency from logs of "MAC DL Transport Block" in MobileInsight. The results (DL NET) are in Table 1. We see that, uplink latency accounts for 66.4-78.0% in PUBG and 68.4-69.2% in VR. Surprisingly, even for the downlink-heavy VR app, uplink latency still contributes to a large portion of the overall latency. Recent techniques (e.g., MIMO and carrier aggregation) and 5G further reduce the DL latency with faster PHY designs. In contrast, as we will see later, the scheduling design employed for the uplink will likely be retained in 5G. As a result, we will focus on the uplink latency in this paper.

[1]VR cannot run on AT&T and Sprint, since their firewalls block the traffic.

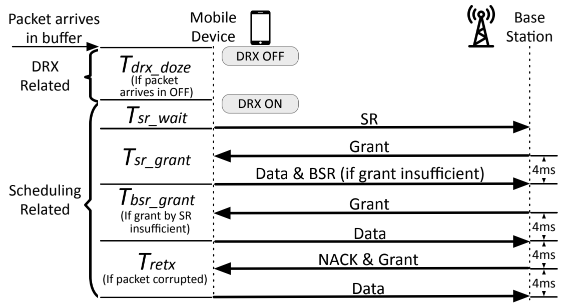


Figure 2: LTE uplink procedure & latency elements.

Latency (ms)	AT&T	T-Mobile	Verizon	Sprint
T_{drx_doze}	29.7	31.9	28.3	29.2
T_{sr_wait}	4.4	4.4	4.6	9.0
T_{sr_grant}	8.2	8.5	8.0	10.1
T_{bsr_grant}	0.03	0.00	0.03	0.16
T_{retx}	0.17	0.14	0.32	0.72

Table 2: Measured latency elements for VR application. T_{drx_doze} is the average value when present.

3.2 Why Long Latency: Breakdown Analysis

We next analyze the root causes for long network latency in 4G LTE. We identify various latency elements for the LTE uplink latency by analyzing the 3GPP standards [1, 2].

We breakdown the uplink latency as shown in Figure 2. The average number of each latency element is shown in Table 2. We can observe that, the major uplink latency bottlenecks are T_{drx_doze}, T_{sr_grant}, and T_{sr_wait}, while T_{bsr_grant} and T_{retx} are one magnitude smaller compared to other elements. We will see how each latency element acts and why it poses or does not pose as the latency bottleneck to our applications.

3.2.1 DRX Doze Latency.

Power-Saving Mode through DRX The power-saving mechanism DRX (Discontinuous Reception) in LTE may also affect latency. In a nutshell, DRX is a technique for a device to save power over LTE (see Figure 3). Instead of continuously waking up for potential downlink delivery from the BS, the device might sleep in the absence of data transfer, thus reducing its energy consumption. In DRX, a device has three states: Long DRX Cycle, Short DRX Cycle, and Continuous Reception (CRX) [2]. In CRX, the device wakes up during the ON period to monitor downlink channels. In long/short DRX, the device only wakes up for a short period of time (set by the onDuration Timer) at the start of each DRX cycle. It dozes off during the OFF period for the remaining time.

The DRX state transition is shown in Figure 3. In the Long/Short cycle state, if any downlink data is received during the ON period, the device enters the CRX state and starts the drx-InactivityTimer. Upon sending an uplink data, the device initiates an SR request. It then switches to the CRX state as well. If the device receives downlink data or initiates another SR request, the timer restarts. The short DRX state

Parameters		AT&T	T-Mobile	Verizon	Sprint
T_{sr_grant}	8ms	96.6%	96.5%	98.8%	0
	10ms	0	0.2%	0.1%	98.1%
	others	3.4%	3.3%	1.2%	1.9%
$T_{sr\_periodicity}$	10ms	94.0%	98.1%	92.3%	11.9%
	20ms	6.0%	1.9%	0	48.9%
	40ms	0	0	7.7%	39.%
$T_{inactivity\_timer}$	200ms	100.0%	99.5%	99.6%	84.5%
	others	0	0.5%	0.4%	15.5%

Table 3: Critical LTE parameters for uplink latency.

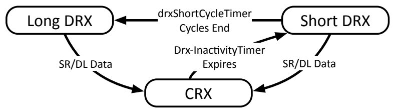


Figure 3: State transition for LTE DRX power-saving.

is entered once the drx-InactivityTimer expires. In this state, the device enters long DRX after the number of drxShortCycleTimer short cycles. All such involved timer parameters are negotiated between the device and the BS during connection setup through RRC.

How downlink DRX incurs long uplink latency DRX is designated for power saving over *downlink* transmissions. It should not block any uplink transfer. In fact, the 3GPP specification [2] stipulates that, upon the uplink sending an SR, downlink DRX should enter the CRX state as if receiving a downlink data packet. However, we found that this is not the case in practice. A new data packet refuses to invoke an SR if the device is in the doze mode. Instead, it continues to doze for a while (the time is denoted as T_{drx_doze}). It then waits for an SR slot to initiate the SR, while migrating the device to the CRX state. Table 2 shows that, T_{drx_doze} is 28.3-31.9ms on average in the four carriers. The maximum latency is 59ms with the 90th percentile being 42ms.

Note that the DRX doze latency is different from the known downlink packet delay due to waiting for DRX ON state. 3GPP [1, 2] does not mandate to prepare for SR at the DRX state. Although this latency element is not standardized, it is common for vendors as they use DRX doze to save energy. The DRX-induced doze timer is hinted in Qualcomm patents [52], where the device defers its SR during DRX OFF for energy savings. We also indirectly validate this behavior in a ZTE Z820 with Mediatek Chipset. For packets with an interval of 1 second, the measured average RTT is 35ms longer than that of packets with a small interval.

Insight: A packet keeps the device at the CRX state for $T_{inactivity\_timer}$. The idea is to reduce the DRX doze latency by sending a dummy message in advance. This way, the device is kept in the ON period before data arrival. The data packet can thus be sent without deferring until the doze period ends.

3.2.2 Scheduling Latency.

Uplink/downlink scheduling in LTE: In LTE, the uplink and downlink data transfers take different approaches:

• *Uplink data transfer over LTE* As shown in Figure 1, the uplink data transmission is through PUSCH (Physical Uplink Shared Channel). All data transmissions are regulated by the BS, which allocates resource blocks (RBs) for the actual transfer. An RB is the smallest unit allocated for a device.

In the scheduling-based LTE design, uplink data *cannot* be immediately sent out before the device is granted resource. This is done via the *request and grant* mechanism. Specifically, the device sends an SR (Scheduling Request) through PUCCH (Physical Uplink Control Channel). SR is a signaling message notifying new data arrival at the mobile device. Moreover, an SR signal cannot be sent at any time instantly. It can only be sent during certain subframes (called SR occasions). The periodicity of SR occasions is notified by the BS during connection setup. Upon receiving an SR, the BS returns an uplink Grant (i.e., *grant*) to the device. A grant specifies what RBs and modulation the device could use *4ms later*. The number of RBs in response to an SR depends on the BS configuration, since SR is just a message stating "device has data to send" without specifying the amount.

• *Downlink data transfer over LTE* LTE still uses the scheduling-based operations for its downlink. However, BS directly allocates RBs for each device upon data arrival, since BS knows what data to transmit to which device.

How scheduling incurs long latency The device also suffers from its uplink scheduling latency. It must wait for an SR occasion before receiving a grant from the BS to upload its data packet. The latency element, denoted as T_{sr_wait}, is thus affected by the periodicity of an SR occasion $T_{sr\_periodicity}$. The device then waits for a grant, which the device could use 4ms later. The latency from sending the SR to sending the data packet is denoted as T_{sr_grant}. The two elements of scheduling are shown in Figure 2. We measure them in Table 2. The SR waiting latency T_{sr_wait} is 4.4ms for AT&T, 4.4ms for T-Mobile, 4.6ms for Verizon, and 9.0ms for Sprint. Sprint has the largest T_{sr_wait} because it has the longest SR cycle. T_{sr_grant} is 8.2ms, 8.5ms, 8.0ms, and 10.1ms for the four carriers. The accumulative latency is denoted as $T_{scheduling} = T_{sr\_wait} + T_{sr\_grant}$.

Insight: This scheduling latency can be reduced. If a grant is pending at the device, a new arriving data packet can use it for transfer. Therefore, we may use a dummy message to request for a grant in advance, so that the data packet can use this grant for actual transfer without delay.

3.2.3 Other Latency Elements.

Buffer status report (BSR) SR is an indicator that informs the BS of new pending data, without specifying *how much*. When the packet that triggers SR is large, the initial grant might be insufficient. The device then sends a BSR (Buffer Status Report) together with the data packets in the scheduled RBs. Unlike SR, a BSR includes the info on how much data still remains in the device buffer. Upon receiving

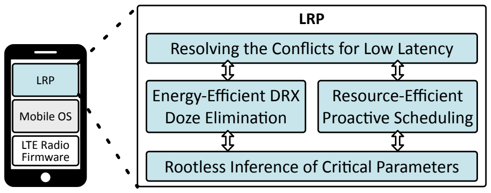

LRP
Resolving the Conflicts for Low Latency
Energy-Efficient DRX Doze Elimination
Resource-Efficient Proactive Scheduling
Rootless Inference of Critical Parameters
LRP
Mobile OS
LTE Radio Firmware

Figure 4: An overview of LRP.

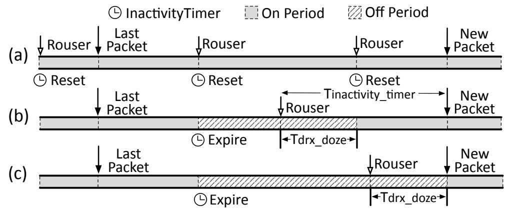


Figure 5: Component solution to DRX doze latency.

the BSR, the BS will process it and respond with sufficient grants for the buffered uplink data.

We note BSR's impact on uplink latency is negligible for most applications in §2. The latency between a BSR and the time to use the grant (denoted as T_{bsr_grant}) is illustrated in Figure 2. Conceptually, it is the request processing time + 4ms, similarly to T_{sr_grant} ($\approx 10ms$). However, it equals to 0 when the initial grant is sufficient. The measurement results are in Table 2. The BSR latency is less than 1ms on average for four US carriers. This is because a base station usually provides a large grant (>100B) sufficient for our apps in response to SR,

Retransmission in LTE An uplink data packet might be corrupted during transfer. Upon receiving a corrupted packet, the BS notifies the device by sending a NACK and a grant. The device uses the grant to retransmit the corrupted data. Similar to BSR latency, the retransmission has limited impact on the uplink latency for apps in §2. The ReTx latency for uplink data packet is fixed at 8ms if needed [1] and 0 otherwise. We denote this latency as T_{retx} and the procedure is shown in Figure 2. Among all data packets, 2.1% in AT&T, 1.7% in T-Mobile, 4.0% in Verizon, and 9.1% in Sprint perceive ReTx latency. Less than 1ms latency is incurred on average, shown in Table 2. Unlike downlink with up to 10% retransmissions [48], uplink packets are small and less prone to corruption.

4 LRP Overview

We devise LRP, an in-device software solution to latency reduction for mobile apps. Figure 4 shows LRP's components. It runs as a user-space daemon at the device, **without** requiring system/root privilege, firmware modification, or hardware support. It is applicable to both Android and iOS. LRP masks the LTE latency elements in §3.2 for applications by *proactively* requesting the needed radio resources and high-speed transfer mode, while still retaining low energy and data consumption overhead. As an application-layer solution, LRP cannot directly control the low-level LTE mechanisms (that require root privilege or firmware access). Instead, it indirectly regulates the LTE uplink transfer with well-crafted *dummy packets*. LRP complements solutions designed to reduce other non-network latency elements [19, 51, 54]. While conceptually simple, LRP must address three challenges:

• **Accurate timing control for each latency element (§5.1–5.2):** Initializing the dummy packets at the right time is crucial to both reducing latency and minimizing energy consumption, signaling overhead, and radio resource usage (thus billing). The proper timing depends on the traffic pattern *and* the unique characteristics of each latency element. To this end, LRP customizes the timing control for critical latency elements, including the DRX doze and scheduling (§3.2).

• **Conflict handling for overall latency reduction (§5.3):** Simply reducing each latency element does not suffice to reduce the overall latency. Due to the complex interactions between LTE latency elements, reducing one latency element may increase other latency elements. Moreover, the dummy packets may compete radio resources with the legitimate data, incurring additional data latency. To this end, LRP devises resolution and avoidance schemes for both types of conflicts.

• **Rootless inference of critical LTE parameters (§5.4):** To be readily usable by *every* device, we design LRP as a user-space software daemon *without* requiring system/root privilege. The challenge is that, LRP's latency reduction requires the fine-grained knowledge of low-level LTE parameters inside the hardware modem chipset. Existing solutions to directly access them (e.g., MobileInsight [32] and QXDM [41]) require root privilege or external hardware support. According to [28], only 7.5% of global mobile devices are rooted. We propose a novel approach to infer these parameters without any system privilege or firmware/hardware modification.

5 The LRP Design

We next elaborate on LRP's design. We first propose component designs to reduce each latency element (§5.1–§5.2), and resolve potential conflicts among them (§5.3). To realize the components without root privilege, we propose a novel inference method at the application layer (§5.4). We analyze LRP and extend the discussion to irregular traffic and 5G (§5.5).

5.1 Energy-Efficient DRX Doze Elimination

To reduce the DRX doze latency in §3.2.1, LRP should ensure the device is in ON period when a data packet arrives at the device buffer. As an application layer solution, LRP cannot directly switch the device to the CRX state (that needs firmware modification). Instead, it sends a dummy packet (*rouser*) *before* the data packet's arrival.

Despite being straightforward at the first glance, a *rouser* is only effective if being sent at the right time. An imprudent *rouser* can either incur unacceptable energy waste or cannot help reduce latency. Therefore, timing control is crucial to balancing latency and energy cost. We first discuss some naive solutions with limitations, and then present our design.

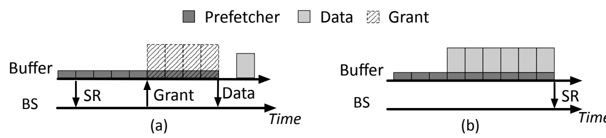


Figure 6: Impact of *prefetcher* Timing.

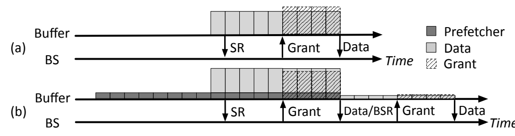


Figure 7: Corner case: a *prefetcher* increases latency.

Naive timing control One naive solution is to keep DRX at CRX state at all times by frequently sending *rouser*s. As shown in Figure 5(a), this can be achieved by sending a *rouser* every $T_{inactivity\_timer}$. Unfortunately, this results in unacceptable energy waste, as the device never enters the doze mode.

A better choice is to send a *rouser* with the time in advance, denoted as t_r, being set to $t_r = T_{inactivity\_timer}$ (Figure 5(b)). On one hand, as the packet keeps the ON period for $T_{inactivity\_timer}$ after dozing, $t_r = T_{inactivity\_timer}$ ensures that the data packet enters the buffer during the ON period. On the other hand, this saves power compared to the first naive choice, since the extra ON period is capped at $T_{inactivity\_timer}$ for each packet at most. However, extra energy consumption is still incurred. Since $T_{inactivity\_timer}$ ($\sim$200ms) is typically much larger than T_{drx_doze} ($\sim$30ms) in reality, the ON period between wakeup from the doze mode and the data packet is unnecessary.

`LRP`'s approach `LRP` prioritizes latency over marginal energy waste with proper timing control. Instead of frequent *rouser*s in naive solutions, `LRP` only sends a *rouser* for the time T_{drx_doze} in advance. We thus keep updating the maximum T_{drx_doze}, denoted as $T_{drx\_doze\_max}$. The timing to send the *rouser* is $t_r = T_{drx\_doze\_max}$. If the device enters the ON period during doze, i.e. $t_r > T_{drx\_doze}$, the *rouser* finishes dozing before the data packet arrives, thus eliminating the doze latency for the data packet. It is also likely that T_{drx_doze} for a *rouser* exceeds t_r. In this case, the packet enters the buffer and endures the dozing latency together with the *rouser*. Although the doze latency is not eliminated, the *rouser* reduces it by t_r.

5.2 Resource-Efficient Proactive Scheduling

`LRP` next seeks to mask the round trips of the scheduling in §3.2.2 for the mobile app. The idea is to send a scheduling request (SR) *before* the arrival of the data, so that the data does not need to wait for the radio grants. As an application-layer solution, `LRP` cannot directly trigger the SR early (which requires modifying the firmware). Instead, it requests a grant from the BS in advance by sending a dummy message, named *prefetcher*. This is feasible since the grant is not tied with the packet that requests it. Moreover, since the BS responds to each SR regardless of the pending data size, a small dummy message can receive a grant that allows for much-larger-size transmission than itself, thus sufficing to accommodate the followup data packet transfer in a single transmission.

Similar to the DRX doze elimination in §5.1, an effective *prefetcher* also needs accurate timing control. As shown in Figure 6, imprudent timing can offset the latency reduction, and/or waste radio resources. We next discuss both naive solutions in Figure 6, and then show `LRP`'s approach.

Naive timing control A too early *prefetcher* might result in both resource waste and prolonged latency as shown in Figure 6(a). The *prefetcher* is sent too early so that the timing to use the returned grant is already passed when the data packet arrives. The resource is thus wasted, while the data packet misses the opportunity to reduce its scheduling latency.

Similarly, a late *prefetcher* could also miss the opportunity to reduce the scheduling latency for the data packet, as shown in Figure 6(b). If the *prefetcher* is sent too late after a potential SR that could reduce latency, the data packet might have to wait for scheduling latency as if no *prefetcher* is issued. In the worst case for both early and late *prefetcher*, it may result in missed latency savings up to $T_{sr\_periodicity} + T_{sr\_grant}$.

`LRP`'s approach `LRP` aims at reducing the scheduling latency at marginal radio resource cost. Let a *prefetcher* be sent t_p before the data packet. The parameter t_p must meet two requirements. First, we should ensure $t_p \geq T_{sr\_grant}$. Note that, an SR can only request a grant to be used at T_{sr_grant} after the SR. Therefore, $t_p \geq T_{sr\_grant}$ guarantees that the SR is sent only if it helps to reduce the scheduling latency for the data packet. Second, we must ensure $t_p \leq T_{sr\_grant}$. This is to let the requested grant be used to transmit the data packet. No resource waste or premature SR is incurred.

Consequently, our timing design is to set the time advance as $t_p = T_{sr\_grant}$, which meets both requirements. Note that T_{sr_grant} is typically constant for a BS, being the accumulative latency of SR processing latency + 4ms, where 4ms is a standardized parameter in [1]. In our experiment, more than 96.5% of T_{sr_grant} is identical under a BS regardless of the carrier. If T_{sr_grant} changes after handover to a new BS, we update T_{sr_grant} immediately. Even if T_{sr_grant} may vary, our solution is no worse than the current practice.

Impact of the data packet size A *prefetcher* helps reduce scheduling latency if the data packet size $\leq$ grant - *prefetcher* size, which is common in reality as >99% of initial grants in our experiments exceed 100B in all operators, while the uplink sensory data is smaller than half of that. Therefore, a *prefetcher* initiates an SR, and gets a returned grant that suffices for the data packet to be sent with the *prefetcher*.

However, a corner case arises when the grant in response to SR is enough for the data packet, but not for a *prefetcher* + the data packet. As shown in Figure 7(b), the device could only send the *prefetcher* and a portion of the data packet. A BSR further requests a grant for the remaining data. The data packet thus suffers extra BSR latency compared to the case without *prefetcher* (Figure 7(a)). In the worst-case scenario, this latency increases by T_{sr_grant} ($\sim$8ms). We discuss the

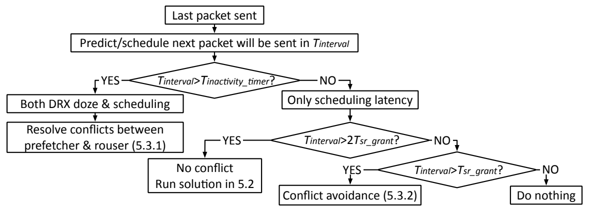

Last packet sent
Predict/schedule next packet will be sent in Tinterval
Tinterval>Tinactivity_timer?
YES
NO
Both DRX doze & scheduling
Only scheduling latency
Resolve conflicts between prefetcher & rouser (5.3.1)
Tinterval>2Tsr_grant?
YES
NO
No conflict
Run solution in 5.2
Tinterval>Tsr_grant?
YES
NO
Conflict avoidance (5.3.2)
Do nothing

Figure 8: The workflow of conflict handling in `LRP`.

probability of this case in Appendix B. However, even in this corner case, the worst case happens only when the data and the *prefetcher* arrive in the same SR period, with probability T_{sr_grant}/T_{sr_wait}. For other conditions in the corner case, the latency is the same as vanilla LTE.

5.3 Handling the Conflicts for Low Latency

`LRP` further resolves several conflicts for overall latency reduction. Figure 8 illustrates the workflow of `LRP`. Let $T_{interval}$ be the time interval between the last and the next expected packet. `LRP` thus reduces various latency elements. It handles improper interplay between latency elements, and between dummy and data packets.

5.3.1 Conflict Resolution Between Latency Elements

`LRP` issues two types of dummy packets for latency reduction: *rouser*s for DRX-induced doze latency, and *prefetcher*s for scheduling latency. Figure 9(a) illustrates their conflicts. A *rouser* itself is a dummy message that needs to be sent before a *prefetcher*. Once turning the device to DRX ON, it asks for the grant, which could carry both *rouser* and *prefetcher*. Therefore, the *prefetcher* is sent by the grant requested by the *rouser*. The grant-induced scheduling latency is not reduced at all. The latency penalty can be as large as $T_{sr\_periodicity} + T_{sr\_grant}$ compared to no-conflict case in Figure 9(b).

To resolve this conflict, we refine the timing control to ensure both dummy packets' effectiveness. Specifically, we should make sure a *rouser* is sent when a *prefetcher* hits the device buffer, so that the *prefetcher* can take effect and reduce the scheduling latency. A *rouser* takes at most $T_{sr\_periodicity} + T_{sr\_grant}$ to be sent out as a dummy message and a *prefetcher* needs to be sent T_{sr_grant} before the data packet. Therefore, we adapt the timer from $t_p = T_{drx\_doze\_max}$ to $T_{drx\_doze\_max} + T_{sr\_periodicity} + 2T_{sr\_grant}$ to ensure a *rouser* is sent before a *prefetcher*. The *rouser* thus endures $T_{drx\_doze\_max}$ that guarantees the doze is completed and then sent out.

5.3.2 Conflict Avoidance Between Dummy and Data

The next conflict arises between `LRP`'s dummy packets and the last legitimate data packet. If a *rouser* conflicts with the last packet, this does not pose an issue: the *rouser* can still help the device to remain in the ON period for $T_{inactivity\_timer}$. We thus only discuss where a *prefetcher* intervenes with the last packet. We show how `LRP` adapts this for latency reduction.

There are two instances when a *prefetcher* arrives in the buffer *before* the last data packet being completely sent out, shown in Figure 10. In case (a), the *prefetcher* does not provide any latency reduction. The grant for the last packet has enough room to carry the *prefetcher*, which will be sent together. There is no *prefetcher*-requesting grant for the next data packet. In case (b), a *prefetcher* may increase the latency. The grant for the last data packet cannot accommodate the piggy-backed transmission of the *prefetcher*. A BSR request is thus triggered by the device to request for more grants. Since BSR specifies the size for the dummy message *prefetcher*, the returned grant does not suffice to transmit the data packet. This subsequently invokes another round of BSR-grant operations. The data packet might suffer from extra BSR latency.

To avoid the conflicts, `LRP` adjusts the timing of a *prefetcher*. It leaves enough time for the last packet to complete its transmission before the *prefetcher*. Recall that the theoretical maximum uplink latency that the last packet would experience after optimization is T_{sr_grant}. The dummy *prefetcher* is then sent at least T_{sr_grant} after the application sent its last packet. Specifically, if the time gap (between the last packet arrival and the next packet arrival) is larger than $2T_{sr\_grant}$, we send a *prefetcher* T_{sr_grant} before the next packet. This is the timing we designed in §5.2; it will not break the above condition. Otherwise, we send a *prefetcher* T_{sr_grant} after the last packet. This choice will reduce less latency compared to the timing in §5.2 without conflicts. However, we avoid the cases of Figure 10, where a conflict negatively affects the latency.

5.4 Rootless Inference of Critical Parameters

As shown in §5.1–5.3, `LRP` relies on knowing certain LTE parameters for latency reduction. Obtaining such parameters through the root privilege can definitely work. However, such an approach limits the applicability of `LRP`. To let `LRP` work with *every* commodity device, we seek to infer these parameters at the application layer. Note that existing tools typically require system privilege (e.g., MobileInsight [32]) or additional hardware (e.g., QXDM [41]).

To infer these critical LTE timers, `LRP` exploits packet pairs for probing. Figure 11 shows the general procedure. `LRP` sends two adjacent probing requests and records their interval t_1. Upon receiving the responses to both packets, `LRP` compares the responses' intervals t_2 with t_1, and estimates the corresponding timers. This approach is based on the premise that, the difference between t_1 and t_2 mainly arises from the different uplink LTE latency experienced by two packets. This premise largely holds in practice, because latency fluctuations from the base station are much larger than those in the core network or servers[2]. Compared with the conventional packet-pair technique, `LRP` customizes probing packets with the LTE

[2]We have validated this premise in operational LTE. We send a pair of DNS requests at $t_1 = 0$. A UL grant suffices to send both requests; they arrive at the BS simultaneously. t_2 is solely affected by the core network and DL. The results show that $t_2 < 1ms$ for >99% responses.

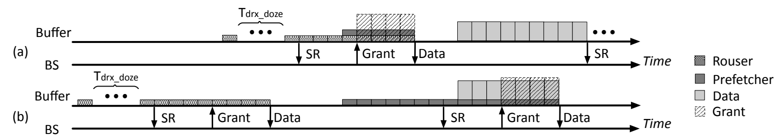


Figure 9: Improper timing control causes conflicts between components.

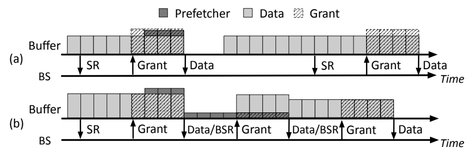
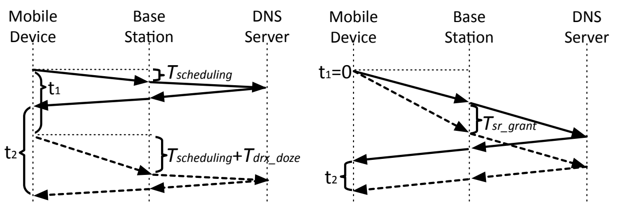


Figure 10: Conflicts: dummy msgs & data.



(a) T_{drx_doze} (b) T_{sr_grant}

Figure 11: Inferring parameters at application layer.

domain knowledge for accurate inference.

Inferring DRX-related parameters To reduce DRX doze latency, LRP should know $T_{drx\_doze\_max}$ (§5.1). Recall that the DRX doze latency is only present when the packet interval is large. The idea is to let t_1 be large enough so that the first response packet cannot keep the second request in DRX ON. The second packet in the pair experiences UL DRX doze latency, while the first does not as we immediately start next pair after one is done. We can thus use the interval difference $t_2 - t_1$ to infer DRX doze latency. Consider that two requests can also be different in terms of scheduling latency, we repeat the pair for 10 times and take the interval difference average. Figure 11(a) illustrates this procedure.

One caveat is that we need to know how large t_1 is so that the second request suffers from DRX doze latency. We increase the interval t_1 gradually until a certain spike appears in measured RTT for the second request, caused by DRX doze latency ($\approx$30ms as shown in §3). The time interval between the first response and the second request that triggers such spike infers $T_{inactivity\_timer}$. We can thus infer $T_{drx\_doze} = t_2 - t_1$. We take the max in multiple rounds as $T_{drx\_doze\_max}$.

Inferring scheduling-related parameters LRP needs T_{sr_grant} to reduce the scheduling latency (§5.2). Figure 11(b) shows how LRP infers it. We let t_1 as 0 by sending both requests together. As we just showed, the grant is sufficient for a single request packet. We increase the size of the second request so that the grant will not be sufficient for both request packets. According to the scheme, the first request experiences only scheduling latency while the second experiences the same scheduling latency plus T_{bsr_grant}, which equals to T_{sr_grant} under a same BS. We thus can derive LRP optimization parameter T_{sr_grant} from the measured t_2 as $T_{sr\_grant} = t_2$.

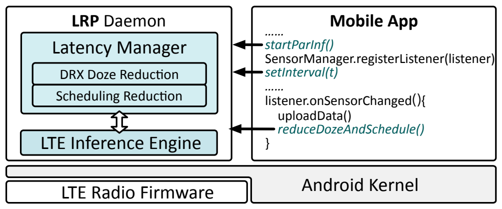
5.5 Miscellaneous Issues

Energy Analysis LRP incurs extra energy overhead from four sources. First, transmitting a *rouser* incurs a longer ON period. The time to send a *rouser* can be as long as $T_{sr\_periodicity} + T_{sr\_grant}$. It incurs $T_{sr\_periodicity}/2 + T_{sr\_grant}$ on average. Second, T_{drx_doze} is not predictable so we select $T_{drx\_doze\_max}$ to prioritize latency over energy. The extra ON period is $\delta = T_{drx\_doze\_max} - T_{drx\_doze}$ for each *rouser*. If the packet arrives during DRX OFF, $T_{drx\_doze\_max}$ equals to T_{drx_doze} and $\delta = 0$. Otherwise, $T_{drx\_doze} = 0$ and $\delta = T_{drx\_doze\_max}$. The expectation of δ is thus $p_{on} \cdot T_{drx\_doze\_max}$, where p_{on} is the probability of a packet arriving during DRX ON period. If we ignore background traffic and assume the packet arrives in the buffer at a random time, p_{on} = onDurationTimer / DRX cycle. Third, an early *rouser* (due to inaccurate estimation) also causes a longer ON period. Denote ε as the estimation error. When the *rouser* arrives during DRX OFF, the extra ON period is ε. Otherwise, ε incurs no extra ON period. Finally, sending extra small messages incurs extra energy waste.

Impact on the spectrum efficiency For every data packet, we define its spectrum efficiency $SE = \frac{\text{sizeof (data packet)}}{\text{sizeof (Total UL resource granted)}}$. When a data packet suffers from doze latency and LRP sends a *rouser*, it reduces *SE* by half: the *rouser* and the *prefetcher* initiate two grants, while the legacy LTE only requests for one. The extra grant occupies $\approx$2 RB in commercial networks. LRP trades-off *SE* for low latency. When LRP sends a *prefetcher* only, two scenarios arise. In the normal case, the grant from SR can carry both the data packet and a *prefetcher*. Therefore, LRP requests no extra grant and *SE* is the same as the legacy LTE. In the corner case discussed in §5.2, the BS allocates at most sizeof (*prefetcher*) extra grant. One extra RB is thus wasted, since a single RB is sufficient to carry a *prefetcher*. *SE* is reduced by $\frac{\text{sizeof } (prefetcher)}{\text{sizeof } (prefetcher + \text{grant from SR})}$. This value multiplying the probability of the corner case (see Appendix B) yields the expectation of *SE* reduction.

Impact of background traffic LRP still reduces latency in the presence of background traffic. No matter whether the background packet is sent before a *rouser* or between a *rouser* and a data packet, the *rouser* will keep the data packet at the DRX ON state, thus eliminating the DRX doze latency. On scheduling latency, if the background traffic is sent after the data packet, it does not affect the *prefetcher*. If the background traffic is in between, the *prefetcher* reduces its latency, which indirectly reduces latency for the real data packet. It still does not increase the latency compared to legacy LTE without LRP. When the background traffic is sent before a *prefetcher*, it will be sent out through BSR before the data packet in the worst case, equivalent to no optimization.

What if the uplink traffic is not strictly regular? While mobile sensors produce regular data packets, the actual uplink



```
......
startParInf()
SensorManager.registerListener(listener)
setInterval(t)
......
listener.onSensorChanged(){
    uploadData()
    reduceDozeAndSchedule()
}
```

Figure 12: Implementation of LRP in Android.

data packets might not be strictly periodic. This can be caused by mobile OS overhead, prediction inaccuracy, or sensor periodicity variance. LRP still guarantees no worse latency than legacy LTE, and saves LTE latency in most scenarios. We show the following Theorem 5.1 and prove it in Appendix C.

Theorem 5.1. For the data packet that should have arrived at $T_{interval}$ but actually arrives at T, LRP does not incur extra latency compared with the legacy 4G LTE.

LRP for non-regular traffic For those ML/AI apps of §2 with irregular but predictable uplink traffic, LRP works equally well. For others with irregular yet unpredictable uplink traffic, we do not recommend LRP for such apps. If users intend to use our APIs, latency reduction cannot be ensured.

Applicability to 5G In principle, LRP is applicable to 5G, which has three usage cases. Enhanced Mobile Broadband (eMBB) extends the current 4G technology. Massive Machine Type Communication (mMTC) is for cellular IoT devices. Its design is based on LTE-M and NB-IoT [8]. The scheduling mechanisms of both modes largely remain unchanged [3, 4]. LRP is still applicable. Ultra Reliable Low Latency Communications (URLLC) targets low-latency communication. The potential grant-free scheduling might partially achieve LRP's latency reduction, but LRP's DRX doze latency reduction will still help the URLLC applications.

Network impact LRP incurs little overhead on the network side. The overhead stems from processing extra signaling, which is marginal compared with normal operations. This is because the BS monitors the control and data channels continuously, regardless of whether it receives an SR.

Impact on other users If the devices under a BS all use LRP, they will still benefit from LRP. The core idea of LRP is to schedule a device's allocated resources in advance if its data arrival can be predicted. The procedure does not sacrifice other users' access in general. Moreover, if certain device does not adopt LRP, its latency may be slightly prolonged. This arises when the BS assigns the last available resource to an LRP user who advances its scheduling, while this resource could have been available to the non-LRP user. However, the impact is minimal, as the BS will serve the user the next subframe (in 1ms) and the throughput is not affected.

6 Implementation

We implement LRP as a standalone user-space daemon with Android NDK. A similar implementation is also feasible for iOS. Figure 12 shows its key components, including a latency manager for latency reduction with conflict resolution in §5.1–5.3, an inference engine that offers key parameters for LRP based on the solutions in §5.4, and a set of APIs for latency-sensitive applications. To use LRP, a latency-sensitive mobile app requests LRP service using its APIs as detailed below. At runtime, LRP first detects if the device connects to a new base station by checking the change of serving cell ID. Upon cell changes, LRP starts to infer the key LTE parameters for this new cell. Once the key parameters are obtained, LRP initiates its latency manager to reduce DRX doze latency in §5.1 and scheduling latency in §5.2, and resolves the conflicts in §5.3.

APIs LRP provides easy-to-use application-layer APIs for mobile application developers. Figure 12 showcases these APIs with a mobile VR application. The app first calls startParInf() so that LRP daemon starts and infers the LTE parameters relevant to latency reduction components. The daemon detects possible parameter changes (say, upon handover) and re-runs the inference procedure whenever necessary. As our VR application uploads periodic sensory data packets, it calls setInterval(t) to inform LRP such periodicity. Whenever a data packet is sent, the application calls reduceDozeAndSchedule() for LRP to reduce latency for the next packet.

Latency manager It realizes the latency reduction in §5.1–5.2 and conflict resolution in §5.3. A practical issue to realize them is to optimize the dummy packet's construction and delivery for low cost. Both *prefetcher* and *rouser* messages in LRP should be as small as possible so that extra data overhead is minimized. In addition, a smaller *prefetcher* will decrease the likelihood of the corner case discussed in §5.2. The smallest packet we could generate in the Android device without root is an ICMP ping packet with IP header only via system command. Our implementation issues only one small ICMP packet to the local gateway in LTE that serves the users.

LTE inference engine It infers the key LTE parameters for LRP's latency reduction based on the approaches in §5.4. We use DNS requests/responses as probing packets, which have low deployment cost (by using LTE's readily-available DNS servers) and higher accuracy (compared to other probing packets delivered with low priority such as ICMP). For DNS servers, LTE assigns its own in-network DNS server when the device attaches to it, which provides fast and stable service. We use such DNS servers for our experiment.

Moreover, we note that simply running the inference in §5.4 may be inaccurate in practice, since it is sensitive to the noises from background traffic, vendor-specific base station behaviors, and server load. To this end, we optimize our implementation to mitigate these noises and improve the inference accuracy. Specifically, we add a few filters to get rid of the noises. For instance, when measuring the scheduling-related parameters, we know that T_{rtt} should be greater than 4ms in reality, therefore, if the packet response pair is received within 4ms, we ignore this round of experiment.

App		AT&T			Verizon			T-Mobile			Sprint			China Mobile		
		Leg	LRP	η	Leg.	LRP	η	Leg.	LRP	η	Leg.	LRP	η	Leg.	LRP	η
Mobile VR	Med.	N/A	N/A	N/A	12.0	8.0	0.5×	11.0	6.0	0.8×	N/A	N/A	N/A	N/A	N/A	N/A
	95%	N/A	N/A	N/A	28.0	15.0	0.9×	40.0	14.0	1.9×	N/A	N/A	N/A	N/A	N/A	N/A
Gaming	Med.	10.0	6.0	0.7×	9.0	7.0	0.3×	9.0	7.0	0.3×	17.0	11.0	0.5×	4.0	3.0	0.3×
	95%	17.0	15.0	0.1×	15.0	15.0	0×	15.0	15.0	0×	27.0	21.0	0.3×	10.0	5.0	1.0×
Localization	Med.	38.0	5.0	6.6×	50.0	14.0	2.6×	42.0	5.0	7.4×	30.5	14.0	1.2×	11.0	3.5	2.1×
	95%	46.0	14.0	2.3×	59.0	23.0	1.6×	48.0	10.0	3.8×	61.7	25.8	1.4×	22.0	6.0	2.7×
Object Detection	Med.	23.0	7.0	2.3×	38.0	9.0	3.2×	33.0	5.0	5.6×	30.0	15.0	1.0×	14.0	6.0	1.3×
	95%	47.8	16.0	2.0×	51.0	15.3	2.3×	45.0	10.0	3.5×	59.0	27.5	1.1×	22.0	17.0	0.3×

NOTE: Mobile VR is evaluated under Verizon and T-Mobile only. Other operators' firewalls block the VR traffic. Leg: Legacy LTE. Med: Median.

Table 4: Uplink network latency (ms) reduction by LRP in evaluations with four apps. η=(Legacy-LRP)/LRP.

7 Evaluation

We assess how LRP improves the overall latencies and QoEs for emergent mobile applications, evaluate the effectiveness of solution components in LRP, and quantify LRP's overhead.

Experimental setup We run LRP on Google Pixel, Pixel 2, Pixel XL, and Pixel 5. We quantify the latency reduction in both US and China over AT&T, Verizon, T-Mobile, Sprint, and China Mobile. The evaluation covers 375 unique cells. We repeat the tests in static, walking (∼1m/s), and driving (∼30mph) scenarios. We do experiments mostly in metropolitan areas while driving tests cover rural areas as well. The radio signal strength varies from -120 to -80dBm, covering good (>-90dBm), fair ([-105, -90dBm]), and bad (<-105dBm) conditions. To quantify LRP's latency reduction, we use MobileInsight [32] to extract the ground truth of fine-grained per-packet latency breakdown from the chipset.

To gauge LRP's impact on the network side, we build a USRP-based testbed. A server with Intel i7-9700k CPU and 32G RAM runs srsLTE [20] for the functions of core network and BS processing. A USRP B210 connects to the server and provides wireless access for the devices. We plug sysmoUSIM [47] into the test phones, and register them.

7.1 Overall Benefits for the Applications

We showcase LRP's latency reduction and QoE improvements with four representative emergent mobile applications:

○ *Mobile VR.* We use the showcase VR game as described in §2. We measure the latency of the sensor data and control data, and use it to gauge how our design reacts to VR games.

○ *Localization.* We write an Android app that uploads the periodic GPS location status to the cloud via the Android API [21]. We encode each location update in 22 bytes and send it to the cloud every second.

○ *Object recognition.* We prototype an object recognition app using MobileNetV2 [43], a phone-based deep learning model. The app processes camera frames and uploads the recognition result to the cloud. The typical inference time is 250ms.

○ *Gaming.* We evaluate its latency by replaying the traces from PUBG Mobile [40], one of the most popular multi-player online mobile games. Since PUBG traffic is not strictly regular, we use it to demonstrate the effectiveness of LRP as discussed in §5.5. We use the traffic emulator to send data packets based on the trace.

Overall LTE latency reduction Table 4 and Figure 13 show LRP's latency reduction for these apps in static settings with fair-good signal strength; other scenarios have similar results as detailed in §7.2. On average, LRP achieves 4-5ms (0.5-0.8×) latency reduction in mobile VR, 8-37ms (1.2-7.4×) reduction in localization, 8-29ms (1.0-5.6×) reduction in object detection, and 1-6ms (0.3-0.7×) reduction in gaming for all 5 LTE carriers. Our breakdown analysis further shows these apps suffer from different latency bottlenecks. For the localization and object detection, the majority of data packets suffer from both DRX doze and scheduling latency. For the VR and gaming with more frequent packets, the scheduling latency is the major latency bottleneck. LRP can reduce both bottleneck latencies and thus benefit all these applications.

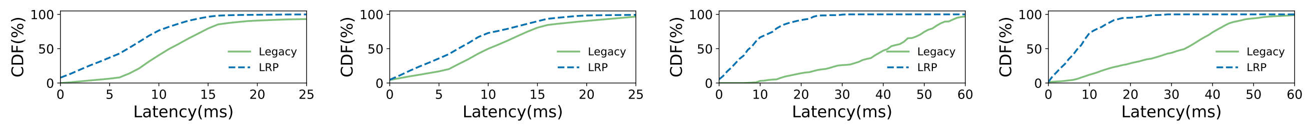
QoE improvement To showcase the impact of LRP on the mobile VR, we conduct a user study with 10 participants to evaluate the subjective experiences of using VR with/without LRP. Figure 14 shows the average Mean Opinion Score (MOS) on three aspects: graphical visual quality, responsiveness, and overall experience. Participants rate 1 (Bad) to 5 (Excellent) on these three aspects of the VR game with constant head position changes. The results show that LRP can improve the visual quality by 8% (3.1→4.0), responsiveness by 63% (2.4→3.9), and overall experience by 46% (2.4→3.5).

5G latency reduction We evaluate how LRP reduces 5G latency under AT&T 5G network. Since we do not have access to its fine-grained data-plane traces, we measure RTT at the application layer. LRP reduces RTT by 4.6ms for Gaming, 20.5ms for Localization, and 19.8ms for Object Detection. The results are similar to the latency reduction in AT&T 4G.

7.2 Micro-Benchmarks

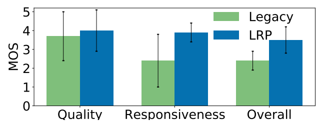
We next assess LRP's solution components under various signal strengths and user mobility patterns.

DRX-induced latency reduction (§5.1) As shown in §5.1, LRP helps reduce the DRX doze latency if the inter-packet interval larger than $T_{inactivity\_timer}$ (otherwise the DRX doze latency is always 0 with/without LRP). Figure 15 shows LRP's



(a) Mobile VR (b) Gaming (c) Localization (d) Object Detection

Figure 13: LTE latency with and without `LRP` in representative apps.



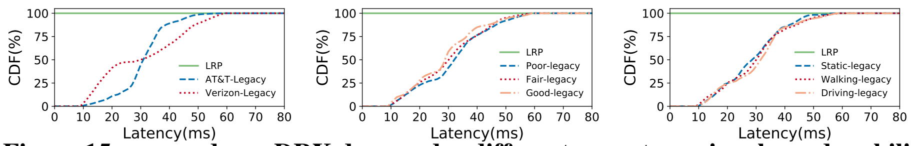


Figure 14: MOS of mobile VR. **Figure 15: `LRP` reduces DRX doze under different operators, signals, and mobility.**

Scenario		AT&T			Verizon			T-Mobile			Sprint			China Mobile		
		Leg.	`LRP`	η	Leg.	`LRP`	η	Leg.	`LRP`	η	Leg.	`LRP`	η	Leg.	`LRP`	η
Static-Poor	Med.	12.0	5.0	1.4×	12.0	9.0	0.3×	11.0	7.0	0.6×	20.0	12.0	0.7×	16.0	5.0	2.2×
	95%	17.0	11.0	0.5×	17.0	17.0	0×	17.0	16.0	0.1×	30.0	26.0	0.2×	26.0	23.0	0.1×
Static-Fair	Med.	13.0	8.0	0.6×	11.0	8.0	0.4×	17.0	13.0	0.3×	18.0	14.0	0.3×	14.0	4.0	2.5×
	95%	17.0	15.0	0.1×	17.0	13.0	0.3×	12.0	6.0	1.0×	32.0	29.0	0.1×	24.0	10.0	1.4×
Static-Good	Med.	13.0	6.0	1.2×	10.0	5.0	1.0×	8.0	6.0	0.3×	13.0	7.0	0.9×	13.0	4.0	2.3×
	95%	17.0	11.0	0.5×	17.0	16.0	0.1×	16.0	11.0	0.5×	27.0	18.0	0.5×	24.0	9.0	1.7×
Walking	Med.	11.0	8.0	0.4×	13.0	6.0	1.2×	12.0	7.0	0.7×	19.0	13.0	0.5×	16.0	9.0	0.8×
	95%	17.0	11.0	0.5×	16.0	16.0	0×	17.0	16.0	0.1×	30.0	26.0	0.2×	30.0	26.0	0.2×
Driving	Med.	14.0	8.0	0.8×	14.0	8.0	0.8×	12.0	8.0	0.5×	17.0	13.0	0.3×	17.0	10.0	0.7×
	95%	18.0	17.0	0.1×	17.0	11.0	0.5×	17.0	16.0	0.1×	29.0	27.0	0.1×	37.0	28.0	0.3×

Table 5: Scheduling latency (ms) in five mobile carriers. η=(Legacy-`LRP`)/`LRP`.

DRX latency reduction under various signal strengths and mobility patterns. We run this test under the most popular setting of $T_{inactivity\_timer}$ = 200ms (Table 3) when the inter-packet interval is $1.5 \cdot T_{inactivity\_timer}$. We also test other intervals and get similar results. In all scenarios, `LRP` reduces the DRX doze latency to 0 for all LTE carriers. This results in 21–41ms mean latency reduction and 40–57ms 95% latency reduction.

Scheduling latency reduction (§5.2) We next quantify the reduction in uplink scheduling latency. The latency reduction ratio, η, is defined as that of the reduced latency and the `LRP` latency. Table 5 shows the results in different carriers, signal strengths, and mobility patterns. In all these scenarios, `LRP` reduces the median scheduling latency by 0.3-2.5×, and reduces the 95th latency by up to 1.7×.

Conflict handling for latency reduction (§5.3) We confirm the effectiveness of `LRP`'s conflict resolution/avoidance. We adapt `LRP`'s APIs to enable/disable the conflict handling in §5.3. Table 6 compares the overall latency with/without `LRP`'s conflict handling. We first illustrate `LRP` can resolve *rouser* and *prefetcher* conflict. We use Localization as its traffic pattern satisfies the condition (long interval) for potential conflict. Compared with no conflict resolution, `LRP` reduces extra 8.82-60.0% latency in all operators. We next evaluate how `LRP` handles data and dummy packets conflicts. The heavy traffic in the Gaming application potentially causes such conflict. We run the Gaming application with `LRP` and the APIs without conflict avoidance. Compared with no conflict avoidance, `LRP` reduces up to 20% extra latency.

Accuracy of critical parameter inference (§5.4) We finally check how accurate our LTE parameter inference is.

Conflicts		AT&T	Ver.	T-M.	Spr.	C. M.
rouser & *prefetcher*	w/o Res.	28.0	30.0	34.0	10.0	13.0
	`LRP`	33.0	36.0	37.0	16.0	16.0
	Extra Red.	17.9%	20.0%	8.82%	60%	23.0%
Data & dummy	w/o Res.	3.5	2.0	2.0	5.0	2.0
	`LRP`	4.0	2.0	2.0	6.0	2.0
	Extra Red.	14.3%	0.0%	0.0%	20%	0.0%

Table 6: Latency (ms) reduction with conflict handle.

	AT&T	Verizon	T-Mobile	Sprint	C. Mobile
Infer T_{rtt}	3.2%	1.5%	2.0%	3.0%	2.0%
Infer $T_{drx\_doze\_max}$	3.0%	1.3%	3.0%	1.3%	2.5%

Table 7: Error rate of `LRP` parameter inference.

For each cell, we first collect ground truth by analyzing the physical/link/RRC-layer signaling messages from MobileInsight. We then use `LRP` component to infer the parameters and compare them with the ground truth. We calculate the average error rate in terms of inference. The results are shown in Table 7. As we can see, the inference error rate is at most 3.2% for both parameters in all 5 operators. `LRP` inference is accurate as argued in §5.4 and §6.

7.3 Overhead

Overhead of dummy messages The dummy messages may incur additional data usage and thus billing. Table 8 shows that `LRP` incurs no more than 0.6KB data per second under all carriers. The data overhead depends on the frequency of calling LRP APIs. For heavy traffic applications (VR, Gaming), the extra overhead is 0.33KB/s while the number for the other two apps is 0.05KB/s. The overhead is acceptable in typical data plans and the extra data is only incurred when `LRP` APIs are called. As explained in §6, `LRP` has minimized the use of dummy for efficient latency reduction.

	AT&T	Verizon	T-Mobile	Sprint	C. Mobile
Extra Data (KB/s)	0.20	0.15	0.41	0.23	0.60
Extra Sig. Msg	3.8%	3.7%	4.3%	3.3%	1.1%
Energy Overhead	1.7%	4.2%	5.8%	2.1%	4.7%

Table 8: Overhead of LRP.

Extra signaling message The dummy messages incur extra signaling between the device and the BS. We measure this overhead as shown in Table 8. LRP incurs up to 4.3% messages, which are marginal compared with the total volume of signaling messages. Reducing latency for apps with DRX doze generates more messages. LRP incurs on average 1.6 extra signaling messages per second for Location and Object Detection. While the other two apps with LRP generate 0.8 extra message every second on average.

Energy consumption While LRP exploits the DRX for lower latency, it still respects the LTE's energy saving with accurate timing control and incurs marginal energy cost. We first compare the percentage of the extra ON period with and without LRP. We track the CDRX events with MobileInsight. As shown in Table 8, for all carriers, at most 5.8% of extra ON period is invoked. Furthermore, we fully charge the device and run Object Detection (with DRX doze) and VR (no DRX doze) applications for one hour, and compare the energy consumption with or without LRP. With LRP, two applications incur 2.5% (16.12% to 16.52% of total battery) and 1.0% (37.04% to 37.40% of total battery) extra battery consumption, respectively. This overhead is marginal, as we adjust the timing of *rouser*s to reduce unnecessary energy waste.

Network impact We measure the network impact of LRP in our SDR testbed. Even in the absence of data transfer, the server spends 0.055ms on average to process the collected signal in every subframe (1ms). In contrast, processing LRP's extra signaling costs 0.002ms, about 3.6% extra overhead.

Impact on other users We next examine whether LRP affects those non-LRP users. We test a two-device scenario, with both running the Gaming app. Device A never uses LRP, whereas device B turns on/off LRP in the test. When B does not run LRP, A's average uplink network latency is 15.79ms, and the 95th percentile is 24.0ms. When B activates LRP, A's average latency becomes 15.84ms and the 95th percentile is 24.0ms. Both numbers are not visibly affected. Therefore, the latency of non-LRP device is not affected, regardless of whether the other runs LRP or not.

8 Related Work

Many cross-layer techniques have been designed to improve user experience and application performance in mobile networks (see [19] for a survey). They use lower-layer information to improve video streaming [54], to optimize Web access [12, 26, 34, 35, 51], to name a few. Most such solutions seek to boost the application-perceived throughput. Other recent proposals detect whether LTE is the bottleneck for applications [9], estimate the radio link speed [11], or examine how LTE configurations affect applications [25]. In contrast, we focus on devising LTE latency-oriented reduction solutions.

Early efforts are also made to reduce the LTE network latency. They analyze the latency for Web access over LTE [39, 53], devise application-specific solutions to LTE scheduling latency with modified modem firmware [48], measure the impact of DRX upon LTE from the energy perspective [23], and adjust the RRC parameters to reduce data-plane latency with infrastructure update [38]. Recent work [9, 30] also makes device-based throughput prediction for performance improvement. Our work differs from them since we work on the latency elements that cannot be eliminated with higher throughput. Authors from [16, 33] target reducing one-time connection setup latency, while LRP reduces latency elements for every data packet in the connected state. Other recent efforts seek to refine the 4G/5G network infrastructure [24, 37]. In contrast, we propose an effective solution without root privilege, device firmware change, or infrastructure upgrade.

9 Conclusion

Reducing latency is critical to many delay-sensitive applications, such as mobile AR/VR, mobile gaming, sensing, machine learning, and robot/drone-based image/speech recognition. In mobile networks like 4G LTE, reducing uplink latency is more challenging than its downlink counterpart, since it involves multiple latency elements stemming from power-saving, scheduling, on-demand resource allocation, etc.

We have designed and implemented LRP, a device-based solution to LTE latency reduction without any infrastructure changes. LRP does not require root privilege at the device and works with mobile apps directly. It ensures the network latency is no worse than the legacy 4G LTE, and is applicable to the upcoming 5G. By design, LRP uses small dummy messages with proper timing control and conflict handling, in order to eliminate unnecessary latency components from scheduling and power-saving operations. Our experiments have confirmed its effectiveness with a variety of mobile apps.

In the broader context, reducing latency poses a more challenging problem than improving throughput for the networked system community. In the mobile network domain, various tricks have been invented for boosting throughput (e.g., massive MIMO, more sophisticated modulation, mmWave, etc.). This is not the case for latency. Both its fundamental theory and effective practice are lacking. Moreover, exploring pure device-based solution, which does not require root privilege and has direct access to user application-level information, offers a nice complement to the infrastructure-centric design, which typically takes years to be deployed.

Acknowledgments We greatly appreciate our shepherd Dr. Lin Zhong and the anonymous reviewers for their constructive feedbacks and advices to help improve the quality of this work. We also thank Mr. Kaiyuan Chen and other members at UCLA WiNG for their early effort and feedback. This work is supported in part by NSF CNS-1910150 and CNS-2008026.

References

[1] 3GPP. TS36.213: Evolved Universal Terrestrial Radio Access (E-UTRA); Physical layer procedures, Sep. 2019.

[2] 3GPP. TS36.321: Evolved Universal Terrestrial Radio Access (E-UTRA); Medium Access Control (MAC) protocol specification, Sep. 2019.

[3] 3GPP. TS38.331: NR; Radio Resource Control (RRC); Protocol specification, Oct. 2019.

[4] 3GPP. TS36.331: Radio Resource Control (RRC), 2020.

[5] Michael Abrash. What VR Could, Should, and almost certainly Will be within two years. http://media.steampowered.com/apps/abrashblog/Abrash%20Dev%20Days%202014.pdf, 2014.

[6] Mikhail Afanasov, Alessandro Djordjevic, Feng Lui, and Luca Mottola. Flyzone: A testbed for experimenting with aerial drone applications. In *Proceedings of the 17th Annual International Conference on Mobile Systems, Applications, and Services*, pages 67–78, 2019.

[7] Amazon. Amazon Alexa. https://www.amazon.com/Amazon-Echo-And-Alexa-Devices/b?ie=UTF8&node=9818047011, Mar. 2020.

[8] Pilar Andres-Maldonado, Pablo Ameigeiras, Jonathan Prados-Garzon, Jorge Navarro-Ortiz, and Juan M Lopez-Soler. Narrowband iot data transmission procedures for massive machine-type communications. *IEEE Network*, 31(6):8–15, 2017.

[9] Arjun Balasingam, Manu Bansal, Rakesh Misra, Kanthi Nagaraj, Rahul Tandra, Sachin Katti, and Aaron Schulman. Detecting if lte is the bottleneck with bursttracker. In *The 25th Annual International Conference on Mobile Computing and Networking*, pages 1–15, 2019.

[10] Tristan Braud, Farshid Hassani Bijarbooneh, Dimitris Chatzopoulos, and Pan Hui. Future networking challenges: The case of mobile augmented reality. In *2017 IEEE 37th International Conference on Distributed Computing Systems (ICDCS)*, pages 1796–1807. IEEE, 2017.

[11] Nicola Bui, Foivos Michelinakis, and Joerg Widmer. Fine-grained lte radio link estimation for mobile phones. *Pervasive and Mobile Computing*, 49:76–91, 2018.

[12] Yi Cao, Javad Nejati, Aruna Balasubramanian, and Anshul Gandhi. Econ: Modeling the network to improve application performance. In *Proceedings of the Internet Measurement Conference*, pages 365–378, 2019.

[13] Jiasi Chen and Xukan Ran. Deep learning with edge computing: A review. *Proceedings of the IEEE*, 107(8):1655–1674, 2019.

[14] Tiffany Yu-Han Chen, Lenin Ravindranath, Shuo Deng, Paramvir Bahl, and Hari Balakrishnan. Glimpse: Continuous, real-time object recognition on mobile devices. In *Proceedings of the 13th ACM Conference on Embedded Networked Sensor Systems*, pages 155–168, 2015.

[15] Open AR Cloud. Open AR Cloud. https://www.openarcloud.org/, Mar. 2020.

[16] Edith Cohen and Haim Kaplan. Prefetching the means for document transfer: A new approach for reducing web latency. In *INFOCOM 2000*, volume 2, pages 854–863. IEEE, 2000.

[17] Google Daydream. https://arvr.google.com/daydream/.

[18] WebRTC for Unity. https://github.com/Unity-Technologies/com.unity.webrtc.

[19] Bo Fu, Yang Xiao, Hongmei Julia Deng, and Hui Zeng. A survey of cross-layer designs in wireless networks. *IEEE Communications Surveys & Tutorials*, 16(1):110–126, 2013.

[20] Ismael Gomez-Miguelez, Andres Garcia-Saavedra, Paul D Sutton, Pablo Serrano, Cristina Cano, and Doug J Leith. srsLTE: An open-source platform for LTE evolution and experimentation. In *Proceedings of the Tenth ACM International Workshop on Wireless Network Testbeds, Experimental Evaluation, and Characterization*, pages 25–32, 2016.

[21] Google. Google API for Android, Mar. 2020.

[22] Giulio Grassi, Kyle Jamieson, Paramvir Bahl, and Giovanni Pau. Parkmaster: An in-vehicle, edge-based video analytics service for detecting open parking spaces in urban environments. In *Proceedings of the Second ACM/IEEE Symposium on Edge Computing*, pages 1–14, 2017.

[23] Junxian Huang, Feng Qian, Alexandre Gerber, Z Morley Mao, Subhabrata Sen, and Oliver Spatscheck. A close examination of performance and power characteristics of 4g lte networks. In *Proceedings of the 10th international conference on Mobile systems, applications, and services*, pages 225–238, 2012.

[24] Aman Jain, NS Sadagopan, Sunny Kumar Lohani, and Mythili Vutukuru. A comparison of sdn and nfv for redesigning the lte packet core. In *2016 IEEE Conference on Network Function Virtualization and Software Defined Networks (NFV-SDN)*, pages 74–80. IEEE, 2016.

[25] Fabian Kaup, Foivos Michelinakis, Nicola Bui, Joerg Widmer, Katarzyna Wac, and David Hausheer. Assessing the implications of cellular network performance on mobile content access. *IEEE Transactions on Network and Service Management*, 13(2):168–180, 2016.

[26] Conor Kelton, Jihoon Ryoo, Aruna Balasubramanian, and Samir R Das. Improving user perceived page load times using gaze. In *14th USENIX Symposium on Networked Systems Design and Implementation*, pages 545–559, 2017.

[27] Ronny Krashinsky and Hari Balakrishnan. Minimizing energy for wireless web access with bounded slowdown. In *Proceedings of the 8th annual international conference on Mobile computing and networking*, pages 119–130, 2002.

[28] Kaspersky Lab. Rooting your android: Advantages, disadvantages, and snags. `https://www.kaspersky.com/blog/android-root-faq/17135/`, Jun. 2017.

[29] Zeqi Lai, Y Charlie Hu, Yong Cui, Linhui Sun, Ningwei Dai, and Hung-Sheng Lee. Furion: Engineering high-quality immersive virtual reality on today's mobile devices. *IEEE Transactions on Mobile Computing*, 2019.

[30] Jinsung Lee, Sungyong Lee, Jongyun Lee, Sandesh Dhawaskar Sathyanarayana, Hyoyoung Lim, Jihoon Lee, Xiaoqing Zhu, Sangeeta Ramakrishnan, Dirk Grunwald, Kyunghan Lee, et al. PERCEIVE: Deep Learning-based Cellular Uplink Prediction Using Real-Time Scheduling Patterns. In *Proceedings of the 18th International Conference on Mobile Systems, Applications, and Services (MobiSys'20)*, pages 377–390, 2020.

[31] Kyungmin Lee, David Chu, Eduardo Cuervo, Johannes Kopf, Yury Degtyarev, Sergey Grizan, Alec Wolman, and Jason Flinn. 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*, pages 151–165, 2015.

[32] Yuanjie Li, Chunyi Peng, Zengwen Yuan, Jiayao Li, Haotian Deng, and Tao Wang. Mobileinsight: Extracting and analyzing cellular network information on smartphones. In *Proceedings of the 22nd Annual International Conference on Mobile Computing and Networking*, pages 202–215, 2016.

[33] Yuanjie Li, Zengwen Yuan, and Chunyi Peng. A control-plane perspective on reducing data access latency in lte networks. MobiCom '17, pages 56–69, New York, NY, USA, 2017. ACM.

[34] Ravi Netravali and James Mickens. Prophecy: Accelerating mobile page loads using final-state write logs. In *15th USENIX Symposium on Networked Systems Design and Implementation*, pages 249–266, 2018.

[35] Ravi Netravali, Vikram Nathan, James Mickens, and Hari Balakrishnan. Vesper: Measuring time-to-interactivity for web pages. In *15th USENIX Symposium on Networked Systems Design and Implementation*, pages 217–231, 2018.

[36] Ravi Netravali, Anirudh Sivaraman, James Mickens, and Hari Balakrishnan. Watchtower: Fast, secure mobile page loads using remote dependency resolution. In *Proceedings of the 17th Annual International Conference on Mobile Systems, Applications, and Services*, pages 430–443, 2019.

[37] Akanksha Patel, Mythili Vutukuru, and Dilip Krishnaswamy. Mobility-aware vnf placement in the lte epc. In *2017 IEEE Conference on Network Function Virtualization and Software Defined Networks (NFV-SDN)*, pages 1–7. IEEE, 2017.

[38] Guillermo Pocovi, Ilaria Thibault, Troels Kolding, Mads Lauridsen, Rame Canolli, Nick Edwards, and David Lister. On the suitability of lte air interface for reliable low-latency applications. In *2019 IEEE Wireless Communications and Networking Conference (WCNC)*, pages 1–6. IEEE, 2019.

[39] Behnam Pourghassemi, Ardalan Amiri Sani, and Aparna Chandramowlishwaran. What-if analysis of page load time in web browsers using causal profiling. *Proceedings of the ACM on Measurement and Analysis of Computing Systems*, 3(2):1–23, 2019.

[40] PUBG. PUBG. `https://www.pubg.com/`, Mar. 2020.

[41] Qualcomm. QxDM Professional - QUALCOMM eXtensible Diagnostic Monitor. `http://www.qualcomm.com/media/documents/tags/qxdm`.

[42] Unity Technologies Report. `https://www.tweaktown.com/news/74682/index.html`.

[43] Mark Sandler, Andrew Howard, Menglong Zhu, Andrey Zhmoginov, and Liang-Chieh Chen. Mobilenetv2: Inverted residuals and linear bottlenecks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 4510–4520, 2018.

[44] Shu Shi, Varun Gupta, and Rittwik Jana. Freedom: Fast recovery enhanced vr delivery over mobile networks. In *Proceedings of the 17th Annual International Conference on Mobile Systems, Applications, and Services*, pages 130–141, 2019.

[45] Apple Siri. Siri. https://www.apple.com/siri/, Mar. 2020.

[46] Unity Render Streaming. https://github.com/Unity-Technologies/UnityRenderStreaming.

[47] sysmocom. sysmoUSIM-SJS1 SIM + USIM Card (10-pack). http://shop.sysmocom.de/products/sysmousim-sjs1, 2016.

[48] Zhaowei Tan, Yuanjie Li, Qianru Li, Zhehui Zhang, Zhehan Li, and Songwu Lu. Supporting mobile vr in lte networks: How close are we? *Proceedings of the ACM on Measurement and Analysis of Computing Systems*, 2(1):1–31, 2018.

[49] Uber. Uber. https://www.uber.com/, Mar. 2020.

[50] Waze. Waze. https://www.waze.com/, Mar. 2020.

[51] Xiufeng Xie, Xinyu Zhang, and Shilin Zhu. Accelerating mobile web loading using cellular link information. In *Proceedings of the 15th Annual International Conference on Mobile Systems, Applications, and Services*, pages 427–439, 2017.

[52] Ming Yang and Tom Chin. Scheduling request during connected discontinuous reception off period, July 20 2017. US Patent App. 14/996,153.

[53] Zengwen Yuan, Yuanjie Li, Chunyi Peng, Songwu Lu, Haotian Deng, Zhaowei Tan, and Taqi Raza. A machine learning based approach to mobile network analysis. In *2018 27th International Conference on Computer Communication and Networks (ICCCN)*, pages 1–9. IEEE, 2018.

[54] Anfu Zhou, Huanhuan Zhang, Guangyuan Su, Leilei Wu, Ruoxuan Ma, Zhen Meng, Xinyu Zhang, Xiufeng Xie, Huadong Ma, and Xiaojiang Chen. Learning to coordinate video codec with transport protocol for mobile video telephony. In *The 25th Annual International Conference on Mobile Computing and Networking*, pages 1–16, 2019.

APPENDIX

A Notations

Notation	Explanation
T_{drx_doze}	The DRX Doze latency for an uplink packet between it enters buffer during DRX OFF and DRX enters ON state
$T_{drx\_doze\_max}$	The maximum doze latency T_{drx_doze} measured under a cell
T_{sr_wait}	The latency of waiting for an uplink scheduling request (SR)
T_{sr_grant}	The time difference between an SR and sending data using the requested grant
T_{bsr_grant}	The time between the first segment of a packet being sent and the last segment being sent through grants via BSR
T_{retx}	The latency of uplink packet retransmission
$T_{scheduling}$	$T_{scheduling} = T_{sr\_wait} + T_{sr\_grant}$
$T_{inactivity\_timer}$	A new data transmission will restart this timer and keep the device in DRX ON state until this timer expires
$T_{sr\_periodicity}$	The periodicity of subframes where a device can initiate an SR
$T_{interval}$	The time interval between the last and the next expected packet

Table 9: Notation table.

B Discussion on the Corner Case

The corner case happens when the grant from an SR is sufficient for the data packet, but insufficient for the data packet and its *prefetcher*. In this section, we discuss the size of the grant from an SR and the probability of this corner case.

Since the size of a data packet and a *prefetcher* is fixed, the occurrence of the corner case depends on the grant from an SR. The BS assigns a grant for the SR according to 3GPP standard [1]. However, it has the freedom to determine the size of the grant. It can assign certain RBs to the user. The number of the RBs is denoted as N_{PRB}. The RB amount is not sufficient to determine the grant size, which is also affected by the modulation index, denoted as I_{MCS}. A BS sends a grant with I_{MCS}, which is affected by channel condition, device power, etc. N_{PRB} can be selected from a subset of discrete values from {1, ...,110}, depending on the channel bandwidth. I_{MCS} can be selected from a subset of discrete values {0, ..., 63}, depending on the modulation capability of the device. Multiple I_{MCS} can map to the same modulation scheme. The UL data that can be sent using this grant is a function of both N_{PRB} and I_{MCS}. This discrete function, denoted as $F(N_{PRB}, I_{MCS})$, is shown in a 110x44 table in 3GPP 36.213 [1].

$F()$ is monotonically increasing with either N_{PRB} or I_{MCS}. Suppose the selection of N_{PRB} and I_{MCS} are independent. Let the probability of the BS selecting $P(N_{PRB} = j) = p_j$, where $j \in \{1,...,110\}$ and $\sum p_j = 1$. Similarly, let $P(I_{MCS} = i) = q_i$, where $i \in \{0,...,63\}$ and $\sum q_i = 1$. Let the size of the data packet be a and the size of a *prefetcher* be a'. $(j,i) \in X_1$ if $a \leq F(j,i) < a + a'$. Otherwise, $(j,i) \in X_2$ Therefore, $p_{corner} = \sum_{(j,i)\in X_1} p_j \cdot q_i$.

From the operational traces, a BS tends to assign $N_{PRB} =$ 2 or 3 in response to an SR. When $N_{PRB} = 2$, any $I_{MCS} \geq 3$ can guarantee $F(N_{PRB}, I_{MCS}) > 100$. When $N_{PRB} = 3$, any $I_{MCS} \geq$ 2 can guarantee $F(N_{PRB}, I_{MCS}) > 100$. In our experiments, >99% of initial grants exceed 100B in all operators. This is sufficient for a small uplink sensory data packet and a small *prefetcher* message.

C Proof for Theorem 5.1

Proof. LRP operates based on the value of $T_{interval}$. When $T_{interval} \geq T_{inactivity\_timer}$, LRP sends a *rouser* to eliminate DRX doze latency. When the next packet arrives later than expected ($T > T_{interval}$), LRP is still very likely to reduce DRX doze latency as the *rouser* sent T_{drx_doze} before the next packet will keep the device in ON state for $T_{inactivity\_timer}$. Therefore, as long as $T \leq T_{interval} - T_{drx\_doze} + T_{inactivity\_timer}$, LRP still reduces DRX doze latency. If $T > T_{interval} - T_{drx\_doze} + T_{inactivity\_timer}$, the device might have already turned to DRX OFF when the next packet arrives. In this situation, the DRX doze latency still exists but LRP does not add extra latency source. Similarly, when the next packet arrives early ($T < T_{interval}$), LRP still reduces doze latency when the *rouser* precedes the data packet, namely $T \geq T_{interval} - T_{drx\_doze}$. LRP cannot eliminate the entire DRX doze latency as the optimal solution, but can still reduce doze latency to $T - (T_{interval} - T_{drx\_doze})$. Otherwise, LRP does not send any *rouser* and the latency is the same compared to no LRP. In summary, if the next packet actually arrives in T where $T_{interval} - T_{drx\_doze} \leq T \leq T_{interval} - T_{drx\_doze} + T_{inactivity\_timer}$, LRP still eliminates or reduces the DRX doze latency. The margin allowed for error (T_{drx_doze} and $T_{inactivity\_timer} - T_{drx\_doze}$) can be 30-80ms in reality depending on common LTE parameters. Otherwise, LRP does not increase the latency.

When $T_{interval} < T_{inactivity\_timer}$, LRP sends a *prefetcher* only (T_{sr_grant} before the next packet) to reduce scheduling latency. If data arrives later, the *prefetcher* is still possible to save its latency if its requested grant can be used by the next packet, which is $T < T_{interval} + T_{sr\_wait}$. Similarly, if data arrives earlier, the *prefetcher* reduces scheduling latency if it is sent before the real data packet, i.e., $T > T_{interval} - T_{sr\_grant}$. When the next scheduled packet arrives in $T_{interval}$ where $T_{interval} - T_{sr\_grant} < T < T_{interval} + T_{sr\_wait}$, LRP still reduces the LTE uplink scheduling latency. This margin allowed for error (T_{sr_grant} and T_{sr_wait}) is usually 8-20ms in reality depending on LTE parameters. Otherwise, the scheduling latency is not reduced but LRP does not incur extra latency. □