Energy Optimization for Mobile Applications by Exploiting 5G Inactive State

Zhi Ding, Yuxiang Lin, Weifeng Xu, Jiamei Lv, Yi Gao, *Member, IEEE*, and Wei Dong, *Member, IEEE*

***Abstract*—The high energy consumption of 5G New Radio (NR) poses a major challenge to user experience. A major source of energy consumption in User Equipments (UE) is the radio tail, during which the UE remains in a high-power state to release the radio sources. Existing energy optimization approaches cut radio tails by forcing the UE to enter a low-power state. However, these approaches introduce extra promotion delays and energy consumption with soon-coming data transmissions. In this paper, we first conduct an empirical study to reveal that the 5G radio tail introduces significant energy waste on UEs. Then we propose 5GSaver, a two-phase energy-saving approach that utilizes the inactive state of New Radio to better eliminate the tail phenomenon in 5G cellular networks. 5GSaver identifies the end of App communication events in the first phase and predicts the next packet arrival time in the second phase. With the learning results, 5GSaver can automatically help the UE determine which radio resource control state to enter for saving energy. We evaluate 5GSaver using 15 mobile Apps on commercial smartphones. Evaluation results show that 5GSaver can reduce radio energy consumption by 9.5% and communication delay by 12.4% on average compared to the state-of-the-art approach.**

***Index Terms*—5G, energy saving, machine learning.**

I. Introduction

The fifth-generation (5G) is a promising technology that is being widely used in smartphones. It has been reported that the unit shipments of 5G-enabled smartphones reach 1.5 billion by 2025 [1]. For end users, 5G is meant to deliver high multi-Gbps throughput, ultra-low latency, and massive network capacity. Higher performance and improved efficiency of 5G can significantly improve mobile user experiences.

However, 5G also brings non-negligible energy costs. Though 5G New Radio is energy-efficient, its energy consumption per unit time can be up to three times more than legacy radios due to its higher data rates, more antennas, and more powerful radio frequency modules [2], [3], [4]. Efficient energy management is extremely important for 5G smartphones with limited battery capacity. Extensive research on 3G/4G radio energy management [5], [6], [7], [8] reveals that a large proportion of energy consumed by radio resources is due to the *radio tail* phenomenon, which refers to the period during which the radio interface keeps high-power states after the completion of data transmissions [5]. Many studies try to cut the 3G/4G radio tail by adopting the fast dormancy mechanism [9], under which a smartphone can ask for an immediate release of radio links and leave the high-power states rapidly after the End Of Communication (EOC) events.

Though there exist preliminary measurement studies [2], [10] and configuration researches [11] on 5G energy consumption, we are the first to focus on energy optimization for mobile applications on 5G smartphones with 5G tail-cutting while existing work has focused more on 5G measurements. This paper intends to answer the following questions.

What are the energy consumption characteristics of 5G radio on smartphones? In particular, whether the energy tail phenomenon still exists in 5G, and what is its impact? To investigate the energy consumption of 5G NR, we have conducted an empirical study on smartphones. The study has verified that the energy consumption of 5G NR on User Equipments (UEs) is almost twice that of 4G LTE. Furthermore, our empirical study shows that the *radio tail* phenomenon still exists in 5G NR. Surprisingly, we find that the high throughput of 5G has instead made the energy waste of radio tails more significant than legacy 3G/4G radios due to the relatively shorter data transmission time of 5G. For example, it takes only 4 seconds to download a 50 MB file in 5G networks, but the radio tail takes nearly 16 seconds, leading to an 80% radio time waste and a serious waste of battery energy in UEs. This ratio can further increase with more small transmissions in the network.

Are existing 3G/4G tail cutting approaches [12], [13] still effective for 5G? Existing 4G tail cutting approaches that simply cut the tails without considering the following traffic pattern would cause non-negligible energy costs and delays in 5G scenarios. No matter when to invoke fast dormancy, a new transmission may start just after an EOC event. In this case, the radio interface needs extra promotion energy and long promotion delays to switch from idle states to active states for the required data transmission. From our empirical study, we find that the inactive state *RRC_INACTIVE* [14], which is a newly introduced Radio Resource Control (RRC) state in 5G NR, can help solve

Manuscript received 24 June 2023; revised 4 January 2024; accepted 1 March 2024. Date of publication 19 March 2024; date of current version 3 October 2024. This work was supported in part by the National Natural Science Foundation of China under Grant 62272407 and Grant 62072396, in part by the "Pioneer" and "Leading Goose" R&D Program of Zhejiang under Grant 2023C01033, in part by the National Youth Talent Support Program, and in part by the Information Technology Center and State Key Lab of CAD&CG, Zhejiang University. Recommended for acceptance by C. M. Pinotti. *(Corresponding authors: Jiamei Lv; Yi Gao.)*

Zhi Ding, Yi Gao, and Wei Dong are with the College of Computer Science, Zhejiang University, Hangzhou, Zhejiang 310027, China (e-mail: dingzhi@zju.edu.cn; gaoyi@zju.edu.cn; dongw@zju.edu.cn).

Yuxiang Lin is with the Alibaba Group, Hangzhou, Zhejiang 311121, China (e-mail: bizhi.lyx@alibaba-inc.com).

Weifeng Xu and Jiamei Lv are with the School of Software Technology, Zhejiang University, Hangzhou, Zhejiang 315048, China (e-mail: xuwf@zju.edu.cn; lvjm@zju.edu.cn).

Digital Object Identifier 10.1109/TMC.2024.3377696

the above issue. The *RRC_INACTIVE* state [14] has low energy consumption and allows the UEs to fast transit to active states for data transmission. However, existing tail cutting approaches are ineffective for 5G NR since they cannot determine whether to enter the inactive state or the idle state after detecting an EOC event. Besides, though previous work [15], [16] that explore the impact of the RRC idle state on the energy consumption of UEs leverage the state to optimize the tradeoff between energy efficiency and packet delay in 5G scenarios, they didn't take radio tail into consideration, which leads to more ratio time waste in 5G networks. Instead, we effectively utilize this inactive state for 5G energy savings by predicting the next packet arrival time after EOC events.

How can we devise better energy optimization approaches by using RRC_INACTIVE*state and what are the implications for future mobile systems/Apps designs?* The addition of an intermediate state like *RRC_INACTIVE*, which enables energy-saving in 5G, poses certain challenges. One of the key challenges is related to the latency and responsiveness. The *RRC_INACTIVE* state introduces additional latency compared to an always-connected state. When a UE transitions from *RRC_INACTIVE* to an active state, there may be a delay in reestablishing the connection and resuming data transmission. Minimizing this latency and ensuring a responsive user experience are crucial challenges. Besides, the introduction of the *RRC_INACTIVE* state requires careful network planning and optimization.

In this paper, we propose 5GSaver, a 5G radio energy-saving approach for mobile Apps. To minimizing the overall latency of transitions from *RRC_INACTIVE* to an active state, 5GSaver identifies the EOC Events and fast recovery events to help UE enter correct RRC states so that the number of transitions from the *RRC_INACTIVE* state to the active state of the UE is as small as possible. 5GSaver adopts a two-phase approach: 1) in the first phase, 5GSaver learns the features of EOC Events (cIEOC) and identifies EOC Events with Random Forest (RF), and 2) in the second phase, if an EOC event is identified, 5GSaver predicts the next Packet Arrival Time (*PPAT*) with Deep Forest (DF) by exploiting program execution patterns as well as per-application statistics. Considering that the frequency of EOC events is much lower than that of non-EOC events for many Apps, the two-phase approach yields high identification accuracy and low computational overhead. We use the *data item* that contains these features over a predefined period as the basic training and recognition unit. Compared to packet granularity, data items can both capture temporal correlation and further reduce computation overhead. At runtime, 5GSaver can intelligently determine which NR state to enter with the learning results. When 5GSaver estimates the transmission will continue, the UE will stay in the active state. When the next packet arrival time is predicted to be close to the EOC event, 5GSaver guides the UE to enter the inactive state. Otherwise, 5GSaver invokes fast dormancy to enter the idle state. In 5GSaver, parameters such as timers, thresholds, and handover mechanisms are also fine-tuned to achieve the desired balance between energy savings and network performance.

We implement and evaluate 5GSaver using 15 different mobile Apps on 3 smartphones and conduct extensive trace-driven experiments. Evaluation results show that compared to commercial strategies, 5GSaver can achieve significant savings on energy (38.4% on average) while introducing acceptable delays. Compared to a state-of-the-art approach RadioJockey, 5GSaver can reduce the radio energy cost by 9.5% while reducing the communication delay by 12.4% on average.

In summary, we make the following key contributions:

1) We present an extensive empirical study on App energy consumption on 5G smartphones. We observe that 5G NR consumes much more energy than 4G LTE and the radio tail phenomenon has a greater impact on the energy consumption of 5G NR.
2) We reveal that existing tail cutting approaches for 4G are not efficient in optimizing 5G energy. We identify two specific reasons leading to the inefficiencies: 1) do not consider the soon coming next data transmissions; 2) do not consider the newly introduced inactive state of NR.
3) We propose a two-phase energy-saving approach, 5GSaver. Evaluation results show 5GSaver outperforms existing tail optimization approaches for both energy savings and communication delays.

II. Related Work

In cellular networks, the radio tail phenomenon contributes a large fraction of radio energy consumption [2], [8], [17], [18]. Existing approaches for mitigating the radio tail phenomenon for 3G/4G radio can be classified into 1) Tail aggregation approaches and 2) Tail cutting approaches. We also introduce some early efforts in 5G energy measurement and optimization.

A. Tail Cutting

Tail cutting approaches aim to directly shorten the radio tail time. Early researches mainly focus on setting an appropriate inactivity timer [19], [20]. Many commercial smartphone models typically invoke fast dormancy with a fixed short inactivity timer, ranging from 3 s to 5 s [13]. Falaki et al. [19] observe that 95% of inter-packet arrival times lie within 4.5 seconds and set the inactivity timer to this fixed value. However, the tail duration is application-dependent and should be adaptable to the target Apps. A fixed inactivity timer is not suitable for diverse and complex mobile Apps. Even if a reasonable tail duration is found, the tail time still exists and causes considerable energy waste in these approaches. On the contrary, 5GSaver can enter the correct states right after detecting state-related features.

Recently, several studies [12], [13], [17], [21] use learning-based approaches to dynamically terminate the high-power tail. Top [13] provides a tail removal API for applications to leverage the fast dormancy feature. Deng et al. [21] invokes fast dormancy using a dynamic inactivity timer, whose value is set based on traffic pattern information. SmartCut [17] trains an AutoRegressive Move Average (ARMA) model to invoke fast dormancy and promote the radio interface in advance to save energy. RadioJockey [12] uses a decision tree model to predict EOC events and invokes fast dormancy if an EOC event has been detected. However, no matter what inactivity timer value is chosen in the above approaches, new network communications may start soon after the EOC events due to the flattened distributions of packet intervals (shown in Section III). When transmitting data with short-term sleep, these approaches may even waste more energy due to frequent state transitions.

To address the above issues, 5GSaver utilizes the inactive state of NR to handle the situation where the UE needs to quickly resume the communication. When 5GSaver predicts that the next packet will arrive right after the EOC events, the UE will be guided to enter the *RRC_INACTIVE* state instead of the *RRC_IDLE* state. This inactive state ensures the UE can save energy by remaining in a relatively low-power inactive state while having the ability of fast recovery.

B. Tail Aggregation

Tail aggregation approaches reschedule transmission in batch to reduce the tail energy consumption of small data transmissions. For example, TailEnder [5] delays transmissions and prefetches data for delay-tolerant Apps. TailTheft [18] performs batching and prefetching in the tail time to save energy. Bartendr [22] prefetches data under good signal strength for syncing and streaming Apps. PerES [8] considers not only the tail energy reduction but also the data transmission energy optimization.

These approaches usually introduce extra delays in batched packet transmissions. 5GSaver focuses on latency-sensitive interactive applications that require real-time interactivity, so we go for a tail-cutting approach for better performance. Specifically, 5GSaver uses learning-based approaches to determine whether to switch states based on currently extracted state-related features and thus does not delay packet transmissions. Besides, the tail aggregation approach designs need to reproduce the App with their own delay-tolerant timers. 5GSaver can be implemented without any changes to the App code. It is worth noting that the tail aggregation approaches are orthogonal to 5GSaver and can help further improve the energy efficiency of 5GSaver in multi-app scenarios. Besides, for some applications that do not need interactivity and run in the background, tail aggregation might work well.

C. 5G Energy Measurement and Optimization

Some recent measurement work [2], [10] conduct measurement studies on 5G energy consumption and propose 5G mode selection-based power management schemes. However, these schemes simply determine whether to use 4G or 5G radios based on the instantaneous traffic intensity and do not solve the tail problem, which is an essential source of energy consumption for 5G phones. Besides, the hardware-based measurement in Narayanan A. et al. [10] is hard to integrate into 5GSaver-equipped cell phones. Lastly, our measurement work under 5G networks in different frequency bands in mainland China complements the findings of Narayanan A. et al. [10].

There also exist some early works on 5G energy optimization [11], [23], [24]. For example, Khlass et al. [11] propose an RRC state handling framework to optimize the state transition in the RAN/CN. However, these works cannot adapt to the diverse and changeable traffic patterns of different Apps. To the best of our knowledge, 5GSaver is the first to utilize the *RRC_INACTIVE* state to solve the tail problem for optimizing the energy consumption for 5G mobile Apps.

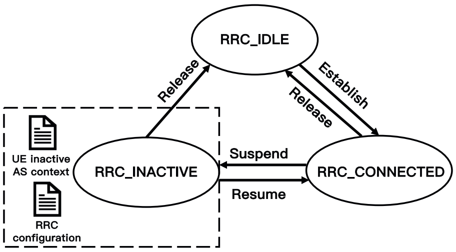


Fig. 1. Three different RRC states and state transitions of UE in 5G NR.

III. Empirical Study

In this section, we present an empirical study that considers four questions about 5G NR energy consumption: (1) How much more energy does 5G NR consume compared to 4G LTE?; (2) Does the radio tail phenomenon still exist in 5G NR?; (3) How do traffic patterns affect existing tail cutting approaches? (4) When is the inactive state more efficient than the idle state? The empirical study results provide important new insight into the 5G NR energy consumption. Besides, in answering these questions, we provide the motivation and countermeasures during the development of 5GSaver.

A. Background

5G NR has three RRC states, i.e., *RRC_IDLE*, *RRC_INACTIVE*, and *RRC_CONNECTED*, as defined in Rel-16 TS 38.331 [14]. Compared to LTE, the *RRC_INACTIVE* state is a notable feature. As shown in Fig. 1, a UE operates simultaneously at a specific RRC state and can transit among three states. To reduce the signaling overhead, NR introduces an *RRC_INACTIVE* state. The station can initiate the state transition to suspend the RRC connection and let the UE enter the *RRC_INACTIVE* state. When entering the *RRC_INACTIVE* state, the UE stores the UE inactive Access Stratum (AS) context and RRC configuration [14]. At the *RRC_INACTIVE* state, when the UE needs to transmit packets, it resumes the suspended RRC connection, restore the above information, and fast transit to the *RRC_CONNECTED* state. Otherwise, the RRC connection is released, and the UE enters the *RRC_IDLE* state when the context is no longer valid or there is no more data transmission. The tail duration is determined by inactivity timers, which are used to control the release of radio resources. Whether the UE should enter the *RRC_IDLE* state or the *RRC_INACTIVE* state is determined based on the requirements of the current service. By default, if there is no specific request or trigger to enter the *RRC_INACTIVE* state, the UE typically transitions directly from the *RRC_CONNECTED* state to the *RRC_IDLE* state. In summary, the advantages of *RRC_INACTIVE* state are that it can significantly reduce signaling overhead, energy consumption, and connection delay for massive 5G devices.

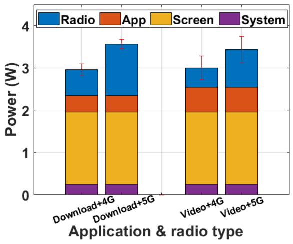


Fig. 2. Power consumption of different applications and radios.

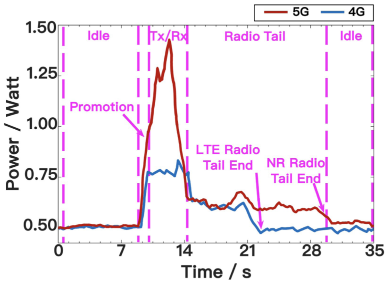


Fig. 3. Example trace of radio tail phenomenon in 4G and 5G network.

B. Experimental Setup

Our experiments are based on three different commercial Android smartphones, ZTE Axon10 Pro, Mi 10, and OnePlus 9. We use the mid-band 5G network provided by China's two largest network carriers, China Mobile and China Telecom.

We develope a battery status monitoring App based on Android APIs to obtain instantaneous current and voltage, and then calculate the instantaneous energy consumption. When running target Apps, our monitoring App runs in the background at a sampling rate of 500 Hz. Due to its simple logic, the extra energy consumption of our monitoring App is negligible (the evaluation of the monitoring App's energy consumption is shown in Section V-D).

C. How Much More Energy Does 5G NR Consume Compared to 4G LTE?

We measure the smartphone's energy consumption when running 2 typical Apps: HTTP download and online video player. Refer to [2], we breakdown the overall 5G/4G energy cost into 4 parts as follows: (1) To get the system consumption, we turn off the screen and turn on the "airplane" mode to kill all background Apps. (2) We then measure the screen element at the maximum brightness with other settings unchanged. (3) To obtain the power consumption of the App alone, we load the App's contents in advance and run the Apps offline. (4) Finally, we record the energy trace of the 4G/5G radio interface at normal operations. We turn on and turn off the 5G mode to measure the energy consumption of 5G NR and 4G LTE, respectively. For each App, we repeat the collection process 10 times. We repeat the evaluation on three smartphones and achieve similar evaluation results on all three phones, where the results on the ZTE Axon10 Pro are shown in Fig. 2. Fig. 2 shows that the 5G NR radio accounts for 30.1% on average of the total energy consumption, which is almost twice the energy consumption of 4G LTE radio. We can also see that the radio energy consumption has a strong positive correlation with the App data rate, which is similar to what is also observed earlier by [2], [10], [25]. This is due to the following two observations. (1) The same radio type in apps with higher data rates (e.g., HTTP download) consumes more energy. (2) The energy consumption variation is more significant in Apps with higher throughput vibration (e.g., online video players).

5G can provide higher throughput than 4G, so we investigate how throughput affects device power consumption over 5G and the energy efficiency of 4G versus 5G at different throughputs. Through control experiments, we measure the device power consumption when transferring data over 4G and 5G with different download throughputs. We run UDP data transfers using iPerf3 and vary the target throughput. Fig. 5 shows the relationship between network throughput and energy efficiency (energy per bit) and compares 4G/LTE and 5G. We can find that for both 4G and 5G, the energy consumption per data transfer unit decreases with increasing throughput. We can also conclude the higher efficiency when transferring at higher throughput under 5G. 5G can be less efficient than 4G at a low throughput but more when the throughput is high. When mobile phone users use 5G daily, they cannot always fully utilize the network's bandwidth resources. Therefore, in actual daily use, 5G may still consume more energy when transmitting data of the same size. Hence, 5GSaver is significant in optimizing the energy consumption of 5G cell phones.

Implication: Compared to 4G LTE, 5G NR radio consumes more than twice the energy consumption. Radio energy consumption is highly related to the App throughput.

D. Does the Radio Tail Phenomenon Still Exist in 5G NR?

Fig. 3 shows an example trace of different states by downloading the file in 5G NR and 4G LTE networks. Our evaluations in 5G networks from different operators yield similar results. From the evaluation, we can observe that since there is no specific request or trigger to enter the *RRC_INACTIVE* state, the UE transitions directly from the *RRC_CONNECTED* state to the *RRC_IDLE* state without transitions to the *RRC_INACTIVE* state. We can observe that the radio tail phenomenon is present in 4G [26], but it is increasing in the 5G era. 5G suffers from a longer radio tail phase than 4G when releasing the RRC connection from *RRC_CONNECTED* and entering *RRC_IDLE*, which leads to additional energy waste. Due to the high throughput of 5G, the radio tail accounts for up to 80% of the active time, which is much higher than 4G when transmitting the same data. Though the power consumption during radio tail is less than that of data transmission, it still consumes up to 49.4% of the total radio energy. Our measurement of radio tail shows that combined with the ratio of radio power consumption (e.g., ∼30% shown in Fig. 2), the power waste of radio tail is about 12%-20% of the total power consumption of the smartphone. Depending on the settings of different operators and the usage habits of different

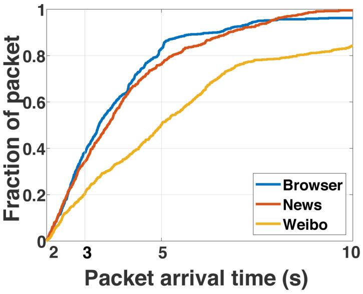


Fig. 4. CDF of next packet arrival times.

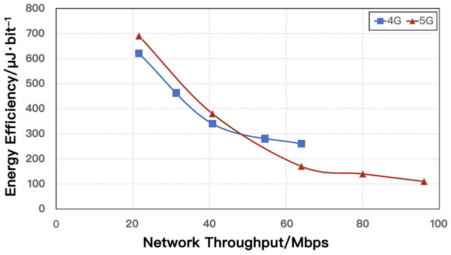


Fig. 5. Energy efficiency in 4G and 5G network.

TABLE I
PARAMETERS OF 5G NR RADIO STATES AND TRANSITIONS IN AN HTTP DOWNLOAD APPLICATION

Transition/State	Power (mW)	Duration (s)
IDLE to CONNECTED	400.5	1.2
INACTIVE to CONNECTED	161.5	0.2
Default tail	292.5	16
CONNECTED state	1026.4	/
IDLE state	112.5	/
INACTIVE state	182.5	/

users, the proportion of radio tail's power waste varies. These proportions further increase when the amount of transmission data decreases.

Though this evaluation is conducted on 5G SA networks, the conclusions still hold even on the 5G NSA networks. Because though the long-tail phase has existed since the 4G era [26], due to the 5G NSA architecture, to complete the switchover from *RRC_CONNECTED* to *RRC_IDLE*, the 5G module must first experience the 4G state machine via LTE RRC reconfiguration. This process is equivalent to activating the LTE tail again, increasing the tail energy overhead. Therefore, how eliminating the radio tail of 5G NR is a critical problem in smartphone energy optimization.

Implication: Massive power waste of NR attributes to the radio tails which have greater impacts on power consumption than LTE.

E. How Do Traffic Patterns Affect Existing Tail Cutting Approaches?

Existing tail cutting approaches mostly identify EOC events and invoke fast dormancy to enter the idle state if an EOC event is detected [12], [13]. However, if a packet arrives just after the EOC event, the UE needs to immediately wake up the radio, thus introducing extra state transition delays and energy consumption. Therefore, these approaches only work well in Apps with aggregated data transfer. In practice, however, most mobile Apps have flattened distributions of time intervals between EOC events and the next network communications. To validate this phenomenon, we extract the distribution of the next packet arrival times after EOC events during normal uses of Apps. In the evaluation, we use these Apps according to our daily habits. In these applications, we consider an EOC event to occur when there is no packet transmission in the next 1 s. Fig. 4 shows the CDF of the next packet arrival times for three typical Apps. As seen, the time intervals span a large range of values. No matter what interval value is chosen for an EOC event, new network communications are likely to start soon. Furthermore, the investigation results of 15 popular mobile Apps show that a large proportion (up to 72.28%) of EOC events is followed by soon coming data transmissions (shown in Table II). These results indicate that existing tail cutting approaches cannot work well in modern mobile Apps.

Implication: Efficient energy consumption approaches must be able to respond to new network communications that may arrive right after EOC events.

F. When is the Inactive State More Efficient Than the Idle State?

To obtain the radio characteristics of NR, we run an HTTP download application on the smartphone. Refer to [7], [10], [27], we use a network-based approach to derive the parameters of RRC states and transitions for 5G NR. Specifically, we deliberately control whether to download files and the interval between two download tasks. By measuring the RTTs and energy levels for different download intervals, we can identify different states and transitions, and calculate the timers of these transitions. We have repeated the experiments 10 times to eliminate the impact of measurement noise. Table I shows the average power consumption and communication delays of different radio states and transitions. We observe that UE in the inactive state consumes slightly higher energy than in the idle state, which means the UE should enter the idle state if it has no data transmission for a long time. However, it consumes much less energy when resuming the RRC connection from the inactive state. The state promotion power of the inactive state is less than half of that of the idle state. More importantly, the INACTIVE-CONNECTED state promotion time is only one-sixth of the IDLE-CONNECTED state promotion time. Therefore, when an application requires short-term sleep, it is more efficient to enter the inactive state rather than the idle state.

Implication: For modern mobile applications that experience frequent short-term sleep (shown in Section III-E), the inactive state has great potential for reducing their communication delays and energy consumption.

IV. SYSTEM DESIGN

The objective of 5GSaver is to identify EOC events and predict the arrival time of the next packet for a specified application. The main benefit of 5GSaver comes from using the *RRC_INACTIVE* state of 5G NR, so it is important to accurately predict the EOC

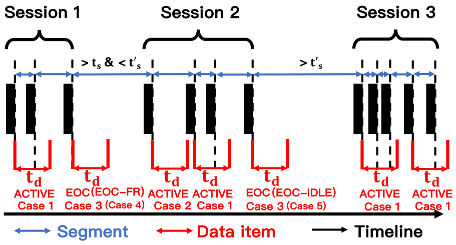


Fig. 6. Extracting data items from the packet traces.

events for different applications and guide 5GSaver to enter the *RRC_INACTIVE* state. To this end, 5GSaver has been designed with a number of refinements to predict application-specific EOC events. Based on these results, 5GSaver can determine 1) whether to leave the *RRC_CONNECTED* state (Section IV-C), and 2) whether to enter the *RRC_IDLE* state or *RRC_INACTIVE* state (Section IV-D).

A. *Definition & Notations*

As shown in Fig. 6, in 5GSaver, we define a *data item* that contains the features over a predefined period as the basic training and recognition unit. We define the period between any two consecutive packets as a *segment*. We define a network *session* as a time period over which the application sends or receives packets such that any two consecutive packets have a small inter-packet time (less than 1 s in our analysis). We use a period t_s to indicate when the UE needs to release the RRC connection, i.e., triggering an EOC event. There are energy savings compared to a scenario where the radio remains on between the two sessions. Since NR supports the efficient *RRC_INACTIVE* state, we set an additional threshold of t'_s in 5GSaver to distinguish whether the UE should enter the inactive state or the idle state. Specifically, if the next packet arrives between t_s and t'_s, the UE should enter the inactive state to suspend the RRC connection for later fast recovery. If the next packet arrives after t'_s, the UE entirely enters the *RRC_IDLE* state. Detailed settings of t_s and t'_s are illustrated in Section V-A. We use t_{seg} to denote the duration of an arbitrary segment. Then we extract *data items*. The time period is t_d ($t_d < t_s$) seconds from these segments as the basic data unit here. Note that t_d can vary across applications since different applications have different traffic patterns. We empirically select an appropriate t_d for a specific application. Detailed energy-saving performance with different data item windows is evaluated in Section V-C.

We define an ACTIVE data item based on the presence of packet transmission. Otherwise, we label the data item as an EOC data item. As shown in Fig. 6, the extraction of data items can be divided into the following three cases.

Case 1. $t_{seg} < t_d$: The duration of a large number of segments is very short due to the frequent packet transmission during data communication. In 5GSaver, we use data items rather than packets (i.e., segments) to make a tradeoff between computation overhead and accuracy. Besides, the use of data items can mitigate the inconsistency of inaccurate timestamps among packet traces and other related feature traces collected by smartphones. In this case, we concatenate as many such short consecutive segments as needed to extract one ACTIVE data item.

Case 2. $t_d < t_{seg} < t_s$: If the duration of a segment is between t_d and t_s, we extract an ACTIVE data item whose time period is from the observed packet up to t_d seconds. We truncate every data item to t_d seconds in length because we want to restrict our learning to features that lie within a short time of observing a packet. This truncation ensures that we can extract features close to packet transmission events.

Case 3. $t_{seg} > t_s$: If the duration of a segment is larger than t_s, we extract an EOC data item whose time period is from the observed packet up to t_d seconds. Without the truncation, 5GSaver may learn rules for EOC events using features over a long time period after observing a packet. Such rules match long after the actual EOC event, thus increasing unnecessary radio on time.

For the task of predicting the arrival time of the next packet, we have further calculated the duration from the end of each EOC data item to the beginning of the next data item. For all EOC data items, we further divide them into two cases based on their durations.

Case 4. $t_s < t_{seg} < t'_s$: If the duration after an EOC data item is between t_s and t'_s, the UE is supposed to be in the INACTIVE state. The communication is fast recovered and the EOC data item is labeled as EOC-FR. The identification of EOC-FR data items is important for enabling a rapid bridge for the reversion.

Case 5. $t_{seg} > t'_s$: When the duration is larger than t'_s, we consider that the application really ends its data transmission and should convert to the idle state. We create an EOC-IDLE data item for this case.

B. *Extracted Features*

We develop a feature extraction module to collect application-related data at a sampling rate of 500 Hz in the background and extract the following features in each data item.

System function call. Internal system function calls are invoked by an App in response to its logic and data. We use STRACE with the process ID to collect system call traces of the specified App. Similar to [12], we use binary to represent whether a certain system call existed or not. For each system call, we use 1 to indicate it is invoked in the current data item, and 0 otherwise. The number and types of used system calls are different in different Apps.

Packet information. We extract the length of the packet and inter-arrival time between adjacent packets in each data item. In particular, for case 1, we extract the packet information of the last packet in the data item. We first use TCPDUMP to dump all network packets[1] and use NETSTAT to periodically get the port numbers opened by the App. Then we filter the packets using these port numbers. It is worth pointing out that in the subsequent energy consumption measurements, we measure the energy consumption when using TCPDUMP and NETSTAT to grab information separately from the energy consumption when not using them to grab information and subtract the two

[1] TCPDUMP does not support filtering packets by process ID

TABLE II
DETAILED INFORMATION OF THE 15 EXPERIMENTAL MOBILE APPLICATIONS

App Type	App	Avg session time/s	Avg inter-session time/s	# of data items	Avg inter-data item time/s	# of EOC	# of EOC-FR	# of EOC-FR /# of EOC	# of features
Information	Ume Browser [28]	2.9437	2.9610	6514	1.1996	968	443	45.76%	78
	Tencent News [29]	4.0752	3.2055	7385	0.9975	784	329	41.96%	84
	Sina Weibo [30]	3.0820	3.5580	3463	2.8508	817	181	22.15%	74
	QQ Browser [32]	6.4939	4.0997	9014	0.4173	499	232	46.49%	90
	Toutiao [33]	5.0955	2.9910	8205	0.4986	512	292	57.03%	85
Video	Kwai [34]	5.4267	3.1718	10878	0.3386	625	407	65.12%	90
	TikTok [35]	2.7338	2.8010	4803	1.0851	1006	449	44.63%	77
	Bilibili [36]	2.9565	3.2758	5059	0.9958	924	269	29.11%	79
	Xigua Video [37]	3.1536	3.3888	4369	1.1886	890	345	38.76%	74
Music	Kuwo Music [38]	2.7318	2.9854	1820	2.7945	801	150	18.73%	83
	Himalaya [39]	2.3493	2.6803	3468	1.5844	1128	349	30.94%	78
	CloudMusic [40]	2.6825	3.5485	4017	1.2524	898	255	28.40%	76
	QQ Music [41]	2.3327	4.5991	3260	1.6282	841	196	23.31%	81
Others	Baidu Netdisk [42]	2.9595	3.2688	4252	1.2710	947	365	38.54%	89
	Soul [43]	2.9019	2.7406	5330	0.8486	938	678	72.28%	86

to exclude the effect of TCPDUMP and NETSTAT (considering the large amount of I/O and computation that sampling may trigger) on the energy consumption measurements.

Application resource usage: Application data transmission affects its resource usage. We use DUMPSYS to record the memory and battery information and use TOP to collect the CPU usage of the application. Then we extract the resource usage close to the end of each data item.

Log information: Smartphones expose all application logs. Although there exist developer-dependent logs, some native system logs may be closely related to the application usage that reflects the start and end of communications. We use LOGCAT and the process ID to get the number of logs and the priority of the last log in each data item.

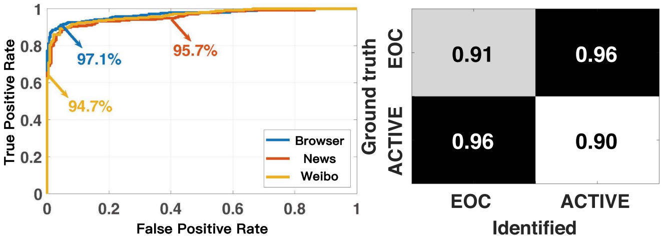
Previous data item state: As there are temporal correlations between consecutive data items, we also record the actual state of the previous data item (1 represents ACTIVE and 0 represents EOC) for learning temporal rules.

In summary, for each data item, we collect features including a variable number of system function calls, 2 packet features, 11 battery features, 8 CPU features, 7 memory features, 2 log features, and 1 previous state. The detailed feature numbers of experimental Apps are shown in the last column of Table II. Though the above features are highly correlated with packet arrivals and EOC events, we still cannot use a mathematical model to characterize these correlations. Therefore, we use learning-based approaches in 5GSaver.

C. IEOC Phase: Identifying EOC

5GSaver uses the RF classifier as its EOC estimation tool. We select RF due to its capabilities of requiring low memory and computation overhead during online estimation, modeling complex relationships among multi-source app-related features, and avoiding overfitting. An important advantage of using RF is it naturally supports feature selection. As some features may not have a strong correlation with the running state of a specific application, it is better to filter out these features during training. For example, log features can not be selected to train the RF model of Ume Browser, which has a relatively poor log quality.

The input to the RF classifier is data items whose labels are ACTIVE and EOC. Note that the number of ACTIVE data items is much more than that of EOC data items, making the dataset significantly unbalanced. We undersample the ACTIVE data items to eliminate the imbalance of sample sizes. Then we normalize all features to accelerate the convergence of the RF model. We use 70% of data items and their corresponding labels as the training set and the rest as the test set. We perform a grid-based search on the training set to automatically find the optimal hyperparameters with a five-fold cross-validation strategy to maximize the classification accuracy. Specifically, we test the following key parameters in RF for an application: decision tree number, max features per tree, min samples leaf, max depth, and min samples split. Finally, we use the tuned optimal parameters to construct the final RF model, which can be applied to the application of interest.

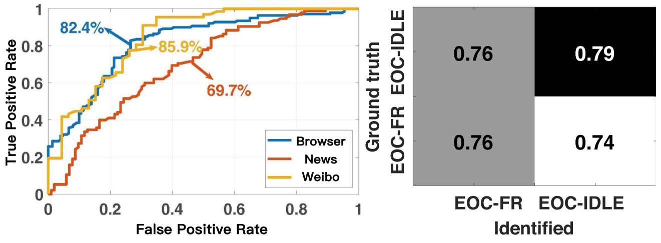


(a) ROC curve

(b) Confusion matrix of the Ume Browser Application

Fig. 7. EOC estimation results.

Fig. 7 shows the EOC estimation results of three typical applications, including Ume Browser [28], Tencent News [29], and Sina Weibo [30]. Other mobile applications have similar EOC estimation accuracy. For each application, we have collected 120 data traces on each mobile phone, each of which lasts 60 seconds. We collect a total of about 54.0GB of traces from a total of 15 Apps. As shown in columns 7 and 8 of Table II, EOC events account for an average of about 15.37% of all traces, and EOC-FR events account for an average of about 6.04% of all traces. In Fig. 7(a), we use the Receiver Operating Characteristic (ROC) curve, which is the comparison of the true positive rate and false positive rate as the criterion changes, to show the classification results since it is not sensitive to sample imbalance. The closer the ROC curve is to the upper left corner, the better the classification performance. We assess the model's performance



(a) ROC curve

(b) Confusion matrix of the Ume Browser Application

Fig. 8. Next packet arrival time prediction results.

with Area Under Curve (AUC), which quantifies the area under the ROC curve. The AUC value of each ROC curve is also shown in the figure. The closer the AUC value is to 1, the better the classification performance. We also show the confusion matrix of classification results of the browser application in Fig. 7(b). Results show that 5GSaver achieves an average AUC of 95.8% in Fig. 7, which means our EOC estimation approach can well capture EOC events even for complex applications such as a mobile browser. Note that this is also the accuracy for the UE to determine whether to leave the active RRC state.

D. PPAT Phase: Predicting Next Packet Arrival Time

As shown in Fig. 6, after identifying that the current data item should be labeled as EOC, 5GSaver further determines whether to enter the idle or inactive state based on t'_s. Instead of obtaining the absolute prediction value, we only need to predict whether the arrival time of the next packet is larger than t'_s. Hence, we turn the prediction task into a binary classification problem that is relatively easy to solve.

Many different machine learning methods can be used to predict the arrival time of the next packet, but our empirical study shows that the DF method exhibits the best performance compared to other methods (such as SVM, DNN, LSTM, etc.). This may be because the parameters required by 5GSaver in order to save energy represent the state of the mobile phone during operation, and the correlation between the parameters is not as strong as some other fields suitable for machine learning methods (such as natural language processing), so the performance of methods including LSTM, neural network, etc. is slightly inferior to the DF method. In view of this, 5GSaver uses DF [31] as the prediction model since it can achieve highly competitive performance to deep neural networks, whereas the performance is quite robust to hyper-parameter settings. Therefore, DF is supposed to perform well with different mobile applications. Similar to RF, DF also integrates a feature selection module to remove irrelevant features and reduce runtime overhead.

Fig. 8 shows the next packet arrival prediction results of the above three applications in the form of ROC and confusion matrix. Other experimental mobile applications listed in Table II have similar prediction accuracy. Results show that for most applications, 5GSaver achieves AUCs of more than 82%, which is also the accuracy for the UE to determine whether to enter the idle state or the inactive state. For the News application, we can observe that the packet arrival prediction accuracy is relatively lower due to its more diverse news content and more frequent user interaction. However, these prediction results can still provide important guidance for subsequent energy savings.

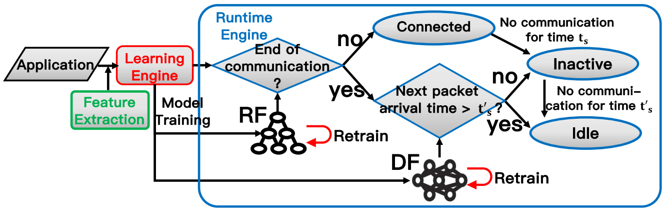


Fig. 9. 5GSaver’s two-phase workflow.

E. 5GSaver’s Workflow

The two-phase workflow of 5GSaver is shown in Fig. 9. During the training phase of 5GSaver, we continuously extract application-related features described in Section IV-A. Then the learning engine of 5GSaver trains the RF classifier and DF prediction models, and feed them into the runtime engine.

We have found that our offline learning engine has a good turnaround time. For complex applications such as Ume Browser, the RF classifier can usually be trained within 0.21 seconds on a typical smartphone (Qualcomm Snapdragon 855, 8G RAM) using 6,514 data items involving 78 features, which is sufficient to achieve a good EOC identification accuracy (shown in Fig. 7). The DF prediction model can also be trained within 1.64 seconds due to its relatively fast training speed compared to other deep learning algorithms [31]. These short training times help us implement an *online learning engine* that combines the pre-trained offline training models with online learning algorithms to capture dynamic application features.

During online learning, 5GSaver continuously monitors the misclassification and misprediction rates of the trained RF and DF models using a sliding time window. The learning engine uses the data items of the current App instance to retrain the RF (or DF) model if the misclassification (or misprediction) rate exceeds a threshold, which means the application’s traffic pattern has changed significantly. Since traffic pattern usually changes due to major version updates of the App or significant changes in user habits, our online learning engine does not introduce much retraining overhead. 5GSaver’s online learning model also allows us to train with each App only once and then generalize the prediction model to users with different usage habits. The online learning performance with different traffic dynamics is evaluated in Section V-C. As a result, the online learning engine can automatically update the RF and DF models in case Apps change their behavior over time.

At runtime, whenever an application sends or receives a packet (Packet 1) after a long idle time, the runtime engine resets the running state, forms a new data item, and starts collecting related features in the data item. If the application further sends or receives the next packet (Packet 2) within the time of a data item t_d, we identify that the current period (from Packet 1 to Packet 2) is ACTIVE, and restart collecting traces as part of the data item started by Packet 2. On the other hand, if there is no network transmission for the time window t_d since Packet 1, the runtime engine needs to use the trained RF classifier and the currently extracted data item to identify whether the application has reached an EOC event. If the data item is identified as ACTIVE, the UE remains in the *RRC_CONNECTED* state. If the data item indicates that an EOC event occurs, 5GSaver needs to further use the DF model to predict whether the application will restart a new communication soon. If so, the UE should suspend the RRC connection and transit to the *RRC_INACTIVE* state. Otherwise, the engine uses fast dormancy to force the radio to enter the *RRC_IDLE* state.

To deal with incorrect classifications/predictions, 5GSaver also integrates a connected-inactive timer and an inactive-idle timer to reduce the influence. These timers make the UE does not stay in the connected and inactive states if there is no more packet transmission. Specifically, in the *RRC_CONNECTED* state, if the application does not send or receive data for the inactive timer t_s, the UE enters into the *RRC_INACTIVE* state. In the *RRC_INACTIVE* state, if the application doesn't send or receive data for the idle timer t'_s, the UE releases the RRC connection and transit to the *RRC_IDLE* state. Hence, compared to existing idle timer-based energy-saving strategies, 5GSaver does not significantly increase the energy consumption. In summary, 5GSaver can achieve energy savings by identifying whether enter low energy consumption states and reduce the communication delay by entering the *RRC_INACTIVE* state.

5GSaver can also be tailored to make it compatible with traditional cellular networks (e.g., 4G LTE) without the *RRC_INACTIVE* state. Specifically, for legacy cellular networks, 5GSaver only uses the EOC estimation module to identify EOC events. If 5GSaver detects an EOC event, it will enter the *RRC_IDLE* state. Otherwise, 5GSaver will keep the radio on to wait for the coming network communications. Detailed performance of the tailored 5GSaver is evaluated in Section V.

V. Evaluation

In this section, we first present the implementation of 5GSaver. Then we show our evaluation results using trace-driven experiments across 15 mobile applications.

A. Methodology

The feature extraction module is implemented as an Android daemon. The learning engine and runtime engine are implemented by Python in Termux, which supports running Python scripts in Android.

Devices and Applications: Our experiment involves three 5G phone models: ZTE Axon10 Pro (with Snapdragon 855, about US$150), OnePlus 9 5G (with Snapdragon 888, about US$320), and Mi 10 (with Snapdragon 865, about US$420). We select these three mobile phones because they cover different price ranges and different vendors to exclude the impact of possible proprietary energy savings solution implemented by the vendor and the hardware performance differences. Like other 5G NR energy-related researches [2], [10], [44], 5GSaver requires the root privilege of Android phones to enable the feature extraction module on these phones. We select 15 Android applications (Table II) that are most downloaded in the ZTE Android market to evaluate 5GSaver. The phones in the evaluation use SIM cards provided by China's two largest network carriers, China Mobile and China Telecom. These two network carriers may have their own configurations such as RRC tail timers (usually 10–20 s). Tests across diverse cellular networks help us more fully evaluate the performance of 5GSaver and understand the 5G energy problem more comprehensively. We recruit 3 volunteers and provide them with phones installed with the 15 applications. In our evaluation, each volunteer uses the aforementioned three mobile phones for the experiment. Each volunteer who participats in the evaluation conducts evaluations on three mobile phones for about 30 hours respectively. In the experiment, each volunteer experiments with each application for 2 hours on each mobile phone. During their normal daily use, our feature extraction module is running in the background. These 2 hours of experimental data are sliced into ∼120 traces where each trace lasts 60 seconds, during which the users are interacting with the corresponding application. Then, we remove traces with missing feature data (due to the feature collection daemon is killed by mistake). Each trace lasts 60 seconds, during which the users are interacting with the corresponding application (we identify whether the user is using an application based on whether it has data transmission).

As the inactive timer t_s and idle timer t'_s are key parameters that dictate the behavior of 5GSaver and associated gains, we adaptively set them for different Apps. For an App, we set the idle timer t'_s to its average inter-session time, which is larger than most session intervals and only smaller than a few long session intervals due to the long-tail App traffic patterns [12]. A session interval larger than t'_s means the UE should enter the idle state for saving more energy. Then we empirically set the inactive timer $t_s{=}\alpha t'_s$ ($\alpha \in (0, 1)$) in 5GSaver, where α is App-dependent and is fixed for a specific App. We show the impact of different α's in Section V-C. Using these settings, we can observe that in all EOC events (indicate temporary suspensions of communication), the proportion of EOC-FR events (will be fast recovered) can be up to 72.3%, which means we must consider the soon coming data transmissions for saving energy in modern mobile Apps.

Metrics: We use two end-to-end metrics to evaluate 5GSaver: energy savings and communication delay. To get the radio energy consumption of each App, we use the breakdown method introduced in [2] to subtract the fixed system and App energy consumption. Though user-experienced delay may include multiple types of delays (e.g., rendering delay, computation delay, etc.), we focus on the radio-induced communication delay here. The communication delay mainly comes from the radio promotion time. For each App, we use the transition duration listed in Table I to calculate its overall communication delay.

Comparison Approaches: We compare 5GSaver with six baseline energy-saving approaches. (1) Commercial timeout-based fast dormancy approach (*Commercial*). As mentioned previously, commercial smartphone models typically invoke fast dormancy with a fixed short inactivity timer, ranging from 3 s to 5 s. This approach can reduce the energy consumed due to the radio tail and is widely adopted by current devices [13]. We use

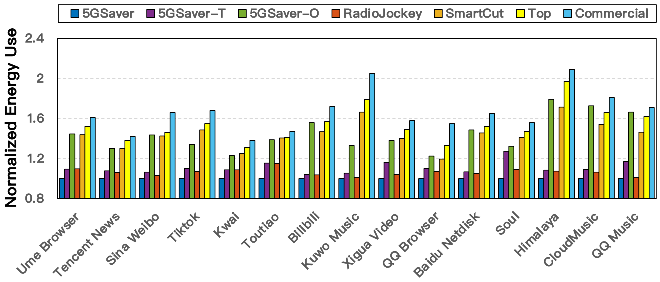


Fig. 10. Energy savings comparison between 5GSaver and baselines. 5GSaver performs best compared to baselines. All volunteers use these Apps and each volunteer use each App on each mobile phone for 2 hours according to their daily habits.

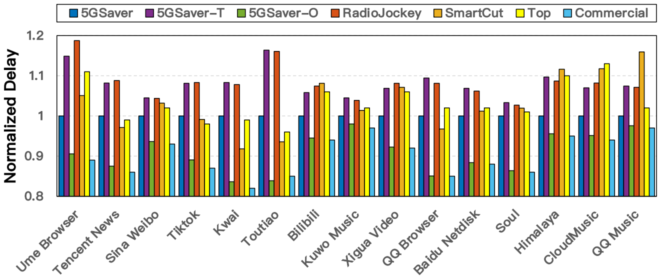


Fig. 11. Communication delay comparison between 5GSaver and baselines. The commercial approach has the minimum delay since it trades significantly higher energy consumption (shown in Fig. 10) for delay reduction.

a three-second inactivity timer-based fast-dormancy strategy, which is widely adopted by current smartphones and has been evaluated to achieve a good tradeoff between energy savings and signaling overhead in [12]. (2) *RadioJockey* [12]. RadioJockey uses a C5.0 decision tree classifier to identify the EOC events and invokes fast dormancy if an EOC event is detected. (3) *SmartCut* [17]. SmartCut uses an ARMA model to estimate packet intervals to cut tails and take advance radio promotion. (4) *Top* [13]. Top performs traffic predictions on 3G networks and provides a tail removal API for applications to leverage the fast dormancy feature. (5) 5GSaver-Tailored (*5GSaver-T*). 5GSaver-T is the tailored version of 5GSaver without considering the inactive state (introduced in Section IV-E). (6) 5GSaver-OnePhase (*5GSaver-O*). 5GSaver-O is the one-phase classification version of 5GSaver and only uses a DF model to identify whether the current data item is ACTIVE, EOC, or EOC-FR. This approach is to demonstrate the effectiveness of our two-phase design. In our evaluation, we implement the RadioJockey [12], SmartCut [17], and Top [13] to evaluate the performance of 5GSaver.

B. 5GSaver's Overall Performance

Figs. 10 and 11 show the normalized energy consumption and communication delay of all evaluated Apps for each approach, respectively. To better evaluate and demonstrate the performance improvements of the 5GSaver, all values are normalized to that of 5GSaver. In the evaluation, all volunteers use these Apps according to their daily habits, and each volunteer use each App on each mobile phone for 2 hours. We have the following seven key observations: (1) Compared to the currently adopted scheme, 5GSaver can save 24.0%-50.7% (38.4% on average) of energy. (2) 5GSaver performs consistently better than state-of-the-art energy optimization approaches. Compared to RadioJockey and SmartCut, 5GSaver improves the energy savings by 3.7%-21.5% (9.5% on average) and 18.2%-44.4% (29.9% on average), respectively. This is mainly attributed to the introduction of the *RRC_INACTIVE* state in 5GSaver. (3) Even without the inactive state, 5GSaver-T performs slightly better than RadioJockey since RF is more efficient than C5.0 decision trees when learning EOC-related features. Specifically, C5.0 decision trees are a single decision tree algorithm, while Random Forests are an ensemble of decision trees. Random Forests often provide better performance and generalization, especially when dealing with complex and noisy datasets, but they are less interpretable compared to individual decision trees. (4) Compared to 5GSaver-O, 5GSaver can significantly reduce energy consumption by 30.2% on average. We find this improvement is due to the imbalance of training events (the number of ACTIVE events is much higher than EOC events). The trained DF model prefers to identify events as ACTIVE and leads to higher energy consumption. (5) Though the performance improvements for some Apps are modest, the performance improvement of 5GSaver is significant for chatty Apps with a high proportion of EOC-FR events. For example, for Apps with a low proportion of EOC-FR events, e.g., Kuwo Music, the reduced energy consumption is around 6.6% compared to RadioJockey since the UE does not need frequent state switching with continuous data transmission. On the other hand, 5GSaver can reduce the energy consumption of Soul, which is the experimental App with the highest proportion of EOC-FR events, by 21.5% compared to RadioJockey. This is because, for such chatty Apps with more intermittent data transmissions, 5GSaver guides UEs to enter the inactive state and save promotion energy consumption when EOC-FR events occur. As chatty Apps (e.g., instant messaging, social networking, and online games) become more and more popular in modern society [45], 5GSaver can achieve a significant energy consumption reduction on smartphones that use these Apps frequently. (6) The 5GSaver consistently achieves better energy savings under different operator networks. Different operators may have different default timer values, which could affect the performance of energy-saving mechanisms like 5GSaver. Therefore, to exclude the influence of network configuration differences between different operators (such as default timer values), we evaluate the performance of 5GSaver separately using network environments provided by two operators. The evaluation results of Figs. 10 and 11 are the average values under 5G networks provided by two different operators. The evaluation results prove that there may be differences in the performance when using networks provided by different operators, but 5GSaver remains effective. This proves that different operators' policies may slightly impact 5GSaver's performance, but the resulting effect is not significant, and 5GSaver is consistently more energy efficient on different operators' networks. (7) 5GSaver has a better energy-saving effect for both push-based applications (e.g., Kwai) and pull-based applications (e.g., QQ Music) than baseline approaches. In fact, 5GSaver's PPAT is used to predict the arrival time of the next packet, which can predict both uplink and downlink packets. Not surprisingly, for applications that run in the background and keep on fetching data periodically (pull-based), 5GSaver works well. For applications that are primarily push-based (e.g., Kwai, Tiktok, Baidu Netdisk shown in Figs. 10 and 11), 5GSaver still demonstrates better performance. This proves that for both push-based and pull-based applications, 5GSaver can save energy in terms of better generalization performance.

As shown in Fig. 11, compared to RadioJockey and SmartCut, 5GSaver can reduce the communication delay by 12.4% and 4.7% on average, respectively. The reduced delay of 5GSaver is mainly due to the faster promotion time of the inactive state. Without the inactive state, the communication delay of 5GSaver-T is also comparable to that of RadioJockey. We can observe that there is a gap between the delay results of 5GSaver and Optimal. This is because the next packet arrival time prediction results are still modest. For applications with frequent EOC-FR events (e.g., Ume Browser, Toutiao), 5GSaver may be easier to introduce UE into the inactive state by mistake. If entering the wrong state, the UE needs extra time to return to the correct state. It is promising to further improve the delay performance by designing more accurate prediction algorithms. We will further investigate other efficient deep learning-based prediction models (e.g., DA-RNN [46]) in our future work. This indicates that there are often new network communications soon after EOC events. An interesting observation is that SmartCut may have a relatively lower delay for chatty applications (e.g., Kwai, Toutiao). This is because SmartCut makes the UE more likely to stay in the active state based on its estimated short packet interval, which helps reduce the state promotion time at the cost of higher energy consumption. This phenomenon is further amplified in 5GSaver-O approaches. We can observe that the communication delays of 5GSaver-O approaches are much better than that of all learning-based energy-saving approaches, especially for applications with frequent EOC-FR events (e.g., Kwai, Toutiao, Soul). This is because these approaches trade significantly higher energy consumption for communication delay using long inactivity timers. For data transmissions with short-term sleep, these approaches are likely to stay in the connected state while 5GSaver enters the inactive state, which introduces extra, although slight, inactive-connected promotion delays when recovering communications.

C. Impact Factors

We have extensively evaluated the impact of different factors on 5GSaver's performance using all test applications. Due to the page limit, we use an example application Ume Browser to show the detailed evaluation results. The results for other applications follow the same pattern.

1) Data Item Window: Next, we focus on our choice of data item window t_d in 5GSaver. We vary t_d during the feature extraction process. Fig. 12 plots the energy savings of 5GSaver and other existing approaches with different data item windows of Ume Browser. In Fig. 12, all curves are normalized to that of the commercially adopted approach. We found that when the data item is small, the energy savings of 5GSaver increase monotonically as the data item window increases. When the data item window approaches the inflection point (i.e., $t_d = 0.3$ s), the energy savings of 5GSaver is minimized. After passing the inflection point, the energy savings linearly decrease as the data item window further increases. This is because when the data item window is small, there is too little data in a data item for 5GSaver to train good learning models and make accurate predictions, leading to lower energy savings. Beyond the inflection point, the learning results are accurate enough, and thus the energy savings become a linear function of how long 5GSaver must wait to make an estimation/prediction. This waiting time is necessary for feature extraction and is exactly equal to the data item window. We also find RadioJockey works well at low data item windows since its simple C5.0 decision tree can already be trained with these data. However, its energy savings are still higher than 5GSaver.

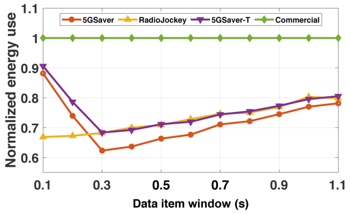


Fig. 12. Energy savings with different data item window.

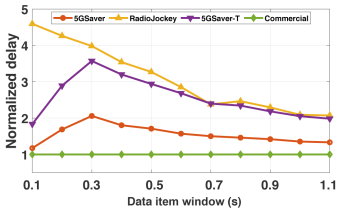


Fig. 13. Communication delays with different data item window.

We have also shown the normalized communication delays of 5GSaver with different data item windows in Fig. 13 and these curves show an opposite trend to the energy-saving curves. However, the additionally introduced communication delay of 5GSaver is consistently much less than RadioJockey. Compared to RadioJockey, 5GSaver can decrease more than 12.2% of communication delays.

The inflection point is relatively stable for a specific App based on experiments across different network conditions. On the other hand, the energy savings and delay reduction of other mobile Apps follow the same pattern as in Figs. 12 and 13. However, different Apps can have different inflection points due to their unique traffic patterns. Therefore, 5GSaver first identifies the inflection point for each App. Our empirical research shows that although different users have different behaviors when using the same application, the traffic characteristics of the applications are similar when viewed in milliseconds. So, once the inflection point of an App has been learned, it can be used by all other users.

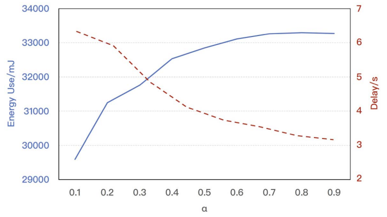


Fig. 14. 5GSaver performance with different inactive timers.

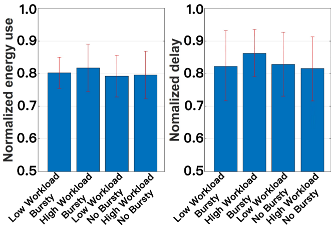


Fig. 15. 5GSaver performance with different traffic dynamics.

TABLE III
DETAILED INFORMATION OF THE 5 SETS OF MULTIPLE APPLICATIONS SCENARIOS

App Group	# of Apps	Applications
Group A	3	QQ Browser, Tencent News, Sina Weibo
Group B	3	Kwai, Kuwo Music, Himalaya
Group C	3	Soul, Baidu Netdisk, Xigua Video
Group D	5	Ume Browser, Tiktok, Toutiao, Bilibili, Soul
Group E	5	Kuwo Music, Himalaya, Tencent News, Baidu Netdisk, QQ Browser

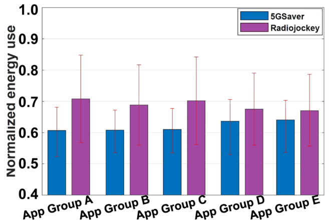


Fig. 16. Performance comparison when running multiple applications.

2) Inactive Timer: As mentioned in Section V-A, 5GSaver uses α to control the inactive timer t_s for an application with a specific idle timer t'_s, where $t_s{=}\alpha t'_s$. Fig. 14 shows the total energy consumption and communication delay of 5GSaver with different α when running the Ume browser application for a total of 60 seconds. We can see that a larger α, which means a larger inactive timer, leads to higher energy consumption and lower communication delay. This is because a large inactive timer leads the UE to enter the inactive state in many cases that should have entered the idle state, resulting in a significant delay reduction at the cost of small extra energy consumption. Taking a tradeoff between energy consumption and communication delay, we set $\alpha = 0.3$ for Ume Browser. It is worth pointing out that the curves of energy and delay with α may vary from application to application due to the traffic characteristics of the application, but the overall trend is similar to that of Ume Browser. α is also stable for a specific application and only needs to be learned once.

3) Traffic Dynamics: 5GSaver integrates an online learning strategy to adapt to traffic dynamics that are introduced by different user behaviors. We deliberately imitate four different traffic dynamics (low workload and bursty, high workload and bursty (instantaneous packet rate $>$ 1000 packets/s), high workload and no bursty, high workload (average packet rate $>$ 500 packets/s) and no bursty) to access the content shown in the browser. Fig. 15 shows the normalized 5GSaver performance with different traffic dynamics. In the figure, the situation where the 5GSaver energy saving algorithm is not used to save energy is normalized to 1. We can observe that both the energy savings and delays have been reduced by up to 20.8% and 17.9% with online learning, respectively compared to the scenario without any power saving algorithm. Among the four scenarios, the high-workload and bursty scenarios perform slightly worse, because the EOC is found to be less accurate under high-load scenarios. While in other cases, for low workload scenarios, 5GSaver finds EOC and saves energy well, while in high workload but no bursty scenarios, the EOC discovery algorithm also works well. This experiment validates the robustness of 5GSaver across different traffic dynamic scenarios. Once 5GSaver has been trained for a specific application by the application provider, phone platform, or even individual users, it can be shared with other users to reduce the application consumption.

4) Multiple Applications: In practice, multiple applications may run simultaneously on the phone and generate data traffic. We use the data traces to simulate five sets of scenarios (Table III) where multiple applications are running simultaneously. Each set of application combinations has been carefully selected to ensure that each set of App combinations covers a different type of Apps in Table II, and all five App combinations cover all participating App types in Table II. To ensure the normal operation of each App, we remain in the connected state when any application is actively communicating. When all applications are identified to meet EOC events, 5GSaver uses the last data item of each application to predict the next packet arrival time. If any App predicts that a packet will arrive soon, we enter the inactive state. Otherwise, we invoke fast dormancy to enter the idle state.

We compare the energy savings and delay reduction between 5GSaver and RadioJockey, which invokes fast dormancy when neither application is classified as active [12]. Fig. 16 shows a performance comparison between 5GSaver and RadioJockey when running multiple Apps. In the figure, the situation where the 5GSaver and RadioJockey algorithm is not used to save energy is normalized to 1. Fig. 16 shows 5GSaver saves an average of 9.8% more energy than RadioJockey in the scenario running 3 Apps, and 7.5% in the scenario running 5 Apps. This is because the number of EOC events that require fast recovery has increased due to the interleaving between applications. When an application generates an EOC event, new network communications of other applications can immediately start to wake up the radio. In these scenarios, the inactive state plays a more important role in saving energy and reducing communication delays.

TABLE IV
PREDICTION ACCURACY EVALUATION RESULTS FOR CROSS-APPLICATION OF 5GSAVER

App Type	App	DF Accuracy	RF Accuracy
Information	Ume Browser	70.13%	83.56%
	Tencent News	58.39%	85.41%
	Sina Weibo	71.94%	82.59%
	QQ Browser	62.45%	81.92%
	Toutiao	82.50%	83.69%
Video	Kwai	85.81%	86.46%
	Tiktok	82.14%	82.14%
	Bilibili	87.21%	84.53%
	Xigua Video	79.59%	80.93%
Music	Kuwo Music	82.43%	83.34%
	Himalaya	79.95%	81.02%
	CloudMusic	86.18%	82.48%
	QQ Music	81.97%	78.54%
Others	Baidu Netdisk	61.57%	76.45%
	Soul	73.54%	74.79%

TABLE V
AVERAGE POWER, CPU UTILIZATION, AND MEMORY COST OF DIFFERENT MODULES OF 5GSAVER

Module	Power (mW)	CPU (%)	Memory (MB)
Feature extraction	26.3	1.50	5.7
Learning engine	269.4	2.54	8.9
Runtime engine	29.4	1.51	7.9

Besides, considering that the features used to predict EOC events for different applications are generally similar, there are some minor differences in the specific features used to predict EOC events for different applications. Therefore, we still utilize the strategy of learning and predicting the EOC event features of different applications separately when evaluating the energy saving effect when multiple applications are running simultaneously. Our evaluation shows that even with an application-agnostic holistic optimization (with the same learning techniques), 5GSaver only decreases the energy savings by 2.1% compared to the aforementioned per-app optimization. performance, which is still better than RadioJockey. This may be due to the fact that the events used to predict the EOC characteristics of different applications are mostly similar, and thus 5GSaver achieves better performance even with application-agnostic holistic optimization.

5) Cross Applications: To evaluate the generalization performance of 5GSaver, we perform leave-one-out cross-validation of 5GSaver in each category of Apps (i.e., Information, Video, Music, and Others) according to the types of the 15 Apps that participate in the evaluation, and the evaluation results are shown in Table IV. Our leave-one-out cross-validation is performed within each category of Apps rather than on all 15 Apps. For example, for the leave-one-out test results of Ume Browser shown in Table IV, we first train the model on the other four Information Apps (i.e., Tencent News, Sina Weibo, QQ Browser, and Toutiao) and test it on Ume Browser. We adopt such a leave-one-out cross-validation approach because, among the 15 Apps mentioned above, the traffic patterns of Apps in different categories tend to have significantly different characteristics. In contrast, the traffic patterns of Apps in the same category tend to have similar characteristics. For example, users of Tiktok and Kwai may like to use it to watch live video streams or short videos, and thus, Tiktok and Kwai may have continuous downlink packets for downloading video streams in their traces. Whereas users of Tencent News and Toutiao may prefer to use it to watch news in text form, so the traces of Tencent News and Toutiao may have EOC events for a few moments after the end of a downlink packet (which may be because after the news that the user is currently browsing has been downloaded, the user needs to spend some time to read the news during this time, the App does not have other downloading behaviors). Therefore, the results of the leave-one-out test within similar Apps can better reflect the generalization performance of 5GSaver, considering the significant differences in the traffic characteristics of different categories of Apps.

According to Table IV, the prediction models trained for each type of Apps have lower prediction accuracy than those trained on a per-App basis in predicting the traffic behavior of that class of Apps. This may be because the prediction model trained for each App can better learn the traffic characteristics specific to different Apps and thus has better prediction results. In addition, the leave-one-out cross-validation results for Video and Music Apps are relatively better, possibly due to the higher similarity of traffic features across Apps in Video and Music categories. In contrast, although some browsers (e.g., Ume Browser and QQ Browser) and news Apps (e.g., Tencent News) are categorized as Information Apps, they may be affected by different users' usage habits. Their traffic characteristics differ (e.g., users may still use browsers to download files). Therefore, the leave-one-out cross-validation results for Information Apps are relatively weak. In addition, the traffic behavior of Others Apps is more different, so the results of Leave-one-out cross-validation are poorer. Overall, our leave-one-out cross-validation evaluation proves that 5GSaver has better generalization performance, and there is still potential room for improvement in model generalization and model cross-domain in future research, which also serves as a direction for us to improve 5GSaver in the future.

D. 5GSaver's Overhead

We have evaluated 5GSaver's overhead of feature extraction and runtime learning on ZTE Axon10 Pro when running Apps in the background. We first only run the feature extraction module and measure its overhead. Then we measure the cost of the learning engine and runtime engine in the runtime learning module, respectively. We calculate their cost by subtracting the cost of the feature extraction module from the total cost. We analyze the power consumption data in the three smartphones in the previous evaluation. Table V shows the average CPU utilization. Note that the measured CPU utilization is relative to a single core. As all of the Snapdragon chips of different brand of the smartphone in our evaluation have eight cores, here we divide the measured utilization by 8.

As seen, the total extra CPU consumption of feature extraction and runtime engine is 3.01% and the total extra memory consumption is 13.6 MB, which is negligible for commercial smartphones. We observe that the power consumption for model learning is relatively higher. However, despite the use of an online learning strategy, 5GSaver does not introduce much additional cost for model training because, as mentioned above, the model update is not frequent during online learning. Besides, as the runtime learning module can also be migrated to high-performance servers, its overhead can be further ignored. It is worth pointing out that with such low resource consumption, 5GSaver reduces communication latency by 12.4% and radio energy consumption by 9.5% compared to RadioJockey. The model used for prediction in 5GSaver is lightweight and can be run on mobile phones with lower overhead and negligible latency. the pre-trained model size of 5GSaver ranged from 3.7 MB to 20.8 MB depending on the parameter settings. Due to the use of lightweight learning models, 5GSaver's CPU and memory overheads are acceptable for common smartphones.

We have also evaluated the extra energy consumption of our energy monitoring App. Our evaluation shows that the power cost of the monitor module is around 28.3 mW, which is negligible on the 5G phone. By attaching a battery tester externally to the cellular device, we measure the overhead energy incurred when our monitoring App runs on the cellular phone. We first test the energy overhead on the cell phone when the measurement module is not running. Then, we test the cell phone energy overhead when the measurement module is running, controlling all other variables to remain constant between the two evaluations. The difference between the two energy overheads is used to calculate the overhead of our monitoring App. We prefer the software-based energy consumption measurement method in 5GSaver. This is because software-based measurements not only measure the energy consumption overhead but also allow for easy measurement of system application layer information, which is impossible with hardware-based measurements.

VI. Discussion

In this section, we discuss some design choices, the generalization ability, limitations, and potential future directions of 5GSaver.

What is the in-field implementation method of 5GSaver? One caveat in 5GSaver is that although 5GSaver does not need to modify the 5G infrastructure, it requires the cooperation of operators to enable active RRC state transitions. To the best of our knowledge, no commercial smartphone provides a programming API for actively transiting RRC states at the UE side. According to the most recent version of RRC protocols [14], UE sides need the cooperation of operators to establish, suspend, resume, and release RRC connections. To obtain support from operators for the widespread use of 5GSaver, more on-site verification and more research on large-scale radio resource scheduling within and between base stations are both needed. In our future work, we plan to implement 5GSaver in 5G private networks to evaluate the performance and scalability of 5GSaver further.

Why do we need a new online learning model for 5G energy saving? Existing methods in the 4G era [12], [17] are inefficient in optimizing 5G energy because they don't consider the upcoming data transmission and the *RRC_INACTIVE* state. Attempts can be made to modify the existing methods to take advantage of the *RRC_INACTIVE* state of 5G NR. However, existing methods lack the prediction of future packets, so they may not facilitate a better choice of whether to enter the *RRC_IDLE* state or the *RRC_INACTIVE* state. Moreover, existing works [12], [17] can not dynamically adapt to changes in App traffic patterns. The online learning engine in 5GSaver is necessary to adapt to traffic dynamics introduced by different user behaviors and can automatically update the models if Apps change their behavior over time. These two key points allow 5GSaver to achieve better energy savings and communication delays than existing approaches. Therefore, though it's possible to modify the existing methods to take advantage of the *RRC_INACTIVE* state, compared to this, the new learning mode adopted by 5GSaver is better adapted to the variable traffic patterns of different Apps in the real world.

Why don't we save energy better by understanding the QoE needs of different Apps and applying earlier state transitions when needed? Though 5GSaver focuses on latency-sensitive interactive applications that require real-time interactivity, for several applications (like email, WhatsApp, etc.), Quality of Experience (QoE) will not hampered much if the message is being fetched after a few seconds. Since considering QoE requirements for different applications is valuable, understanding QoE of Apps and making early state transitions for Apps whose additional latency due to state transition will affect the QoE can potentially enhance performance. However, given the vast diversity of mobile applications and the varying QoE requirements even for the same application among different users, implementing a comprehensive understanding of QoE for each application becomes challenging. Furthermore, the associated overhead in analyzing QoE requirements for diverse applications may outweigh the benefits. Therefore, 5GSaver aims to provide a generalized energy-saving method applicable to a broad spectrum of applications, ensuring practicality and ease of extension to different use cases.

VII. Conclusion

In cellular networks, a large proportion of battery energy is wasted in the radio tail. In this paper, we conduct an empirical energy consumption study and validate that the radio tail phenomenon still exists in 5G cellular networks and wastes a more significant amount of energy. To better eliminate the radio tail phenomenon, we propose a two-phase energy-saving approach 5GSaver that integrates the inactive state of 5G NR. 5GSaver uses application-related features to train an RF model for identifying EOC events and train a DF model for predicting the next packet arrival time. Based on the learning results, the UE can enter the appropriate states for saving energy. Extensive trace-driven experimental results on 15 mobile applications demonstrate the effectiveness of 5GSaver in achieving better energy savings and communication delays compared to existing tail optimization approaches.

Acknowledgement

The authors would like to thanks all the reviewers for their valuable comments and helpful suggestions.

REFERENCES

[1] Statista, "Forecast 5G-enabled smartphone unit shipments worldwide from 2019 to 2025," Retrieved Mar. 2021, 2020. [Online]. Available: https://www.statista.com/statistics/792394/5g-smartphone-shipments-worldwide/

[2] D. Xu et al., "Understanding operational 5G: A first measurement study on its coverage, performance and energy consumption," in *Proc. ACM Annu. Conf. ACM Special Int. Group Data Commun. Appl., Technol., Architectures, Protoc. Comput. Commun.*, 2020, pp. 479–494.

[3] M. Koziol, "5G's waveform is a battery vampire," Retrieved Mar. 2021, 2019. [Online]. Available: https://spectrum.ieee.org/5gs-waveform-is-a-battery-vampire

[4] I. Chih-Lin, S. Han, and S. Bian, "Energy-efficient 5G for a greener future," *Nature Electron.*, vol. 3, no. 4, pp. 182–184, 2020.

[5] N. Balasubramanian, A. Balasubramanian, and A. Venkataramani, "Energy consumption in mobile phones: A measurement study and implications for network applications," in *Proc. ACM SIGCOMM Conf. Internet Meas.*, 2009, pp. 280–293.

[6] F. Qian, Z. Wang, A. Gerber, Z. M. Mao, S. Sen, and O. Spatscheck, "Characterizing radio resource allocation for 3G networks," in *Proc. ACM SIGCOMM Conf. Internet Meas.*, 2010, pp. 137–150.

[7] J. Huang, F. Qian, A. Gerber, Z. M. Mao, S. Sen, and O. Spatscheck, "A close examination of performance and power characteristics of 4G LTE networks," in *Proc. 10th Int. Conf. Mobile Syst., Appl., Serv.*, 2012, pp. 225–238.

[8] Y. Cui et al., "Performance-aware energy optimization on mobile devices in cellular network," *IEEE Trans. Mobile Comput.*, vol. 16, no. 4, pp. 1073–1089, Apr. 2017.

[9] UE, "Fast Dormancy," behavior. 3GPP discussion and decision notes R2-075251, 2007.

[10] A. Narayanan et al., "A variegated look at 5G in the wild: Performance, power, and QoE implications," in *Proc. ACM SIGCOMM Conf.*, 2021, pp. 610–625.

[11] A. Khlass, D. Laselva, and R. Jarvela, "On the flexible and performance-enhanced radio resource control for 5G NR networks," in *Proc. IEEE 90th Veh. Technol. Conf.*, 2019, pp. 1–6.

[12] P. K. Athivarapu et al., "RadioJockey: Mining program execution to optimize cellular radio usage," in *Proc. 18th Annu. Int. Conf. Mobile Comput. Netw.*, 2012, pp. 101–112.

[13] F. Qian, Z. Wang, A. Gerber, Z. M. Mao, S. Sen, and O. Spatscheck, "TOP: Tail optimization protocol for cellular radio resource allocation," in *Proc. 18th IEEE Int. Conf. Netw. Protoc.*, 2010, pp. 285–294.

[14] 3GPP, "TS 38.331 (version 16.2.0, release 16): 5G; NR; Radio resource control (RRC); Protocol specification," 2020. [Online]. Available: https://www.etsi.org/deliver/etsi_ts/138300_138399/138331/16.02.00_60/ts_138331v160200p.pdf

[15] A. Khlass, D. Laselva, and R. Jarvela, "On the flexible and performance-enhanced radio resource control for 5G NR networks," in *Proc. IEEE 90th Veh. Technol. Conf.*, 2019, pp. 1–6.

[16] A. Khlass and D. Laselva, "Efficient handling of small data transmission for RRC inactive UEs in 5G networks," in *Proc. IEEE 93rd Veh. Technol. Conf.*, 2021, pp. 1–7.

[17] G. Xue, H. Zhu, Z. Hu, J. Yu, Y. Zhu, and G. Zhang, "SmartCut: Mitigating 3G radio tail effect on smartphones," *IEEE Trans. Mobile Comput.*, vol. 14, no. 1, pp. 169–179, Jan. 2015.

[18] D. Zhang, Y. Zhang, Y. Zhou, and H. Liu, "Leveraging the tail time for saving energy in cellular networks," *IEEE Trans. Mobile Comput.*, vol. 13, no. 7, pp. 1536–1549, Jul. 2014.

[19] H. Falaki, D. Lymberopoulos, R. Mahajan, S. Kandula, and D. Estrin, "A first look at traffic on smartphones," in *Proc. 10th ACM SIGCOMM Internet Meas. Conf.*, 2010, pp. 281–287.

[20] C.-C. Lee, J.-H. Yeh, and J.-C. Chen, "Impact of inactivity timer on energy consumption in WCDMA and cdma2000," in *Proc. IEEE Wireless Telecommun.*, 2004, pp. 15–24.

[21] S. Deng and H. Balakrishnan, "Traffic-aware techniques to reduce 3G/LTE wireless energy consumption," in *Proc. 8th Int. Conf. Emerg. Netw. Experiments Technol.*, 2012, pp. 181–192.

[22] A. Schulman et al., "Bartendr: A practical approach to energy-aware cellular data scheduling," in *Proc. 16th Annu. Int. Conf. Mobile Comput. Netw.*, 2010, pp. 85–96.

[23] S. Ryoo, J. Jung, and R. Ahn, "Energy efficiency enhancement with RRC connection control for 5G new RAT," in *Proc. IEEE Wireless Commun. Netw. Conf.*, 2018, pp. 1–6.

[24] Y.-N. R. Li, M. Chen, J. Xu, L. Tian, and K. Huang, "Power saving techniques for 5G and beyond," *IEEE Access*, vol. 8, pp. 108675–108690, 2020.

[25] A. Narayanan et al., "A first look at commercial 5G performance on smartphones," in *Proc. ACM Web Conf.*, New York, NY, USA, 2020, pp. 894–905.

[26] J. Huang, F. Qian, A. Gerber, Z. M. Mao, S. Sen, and O. Spatscheck, "A close examination of performance and power characteristics of 4G LTE networks," in *Proc. 10th Int. Conf. Mobile Syst., Appl., Serv.*, New York, NY, USA, 2012, pp. 225–238. [Online]. Available: https://doi.org/10.1145/2307636.2307658

[27] S. Rosen et al., "Discovering fine-grained RRC state dynamics and performance impacts in cellular networks," in *Proc. Int. Conf. Mobile Comput. Netw.*, 2014, pp. 177–188.

[28] Ume Browser, "UME technologies co.,ltd," 2021. Accessed: Mar. 2021. [Online]. Available: https://umeweb.com/

[29] Tencent News, "Tencent technology co.,ltd," 2021. Accessed: Feb. 2021. [Online]. Available: https://news.qq.com/mobile/

[30] Sina Weibo, "Sina corporation," 2021. Accessed: Feb. 2021. [Online]. Available: https://weibo.com/

[31] Z.-H. Zhou and J. Feng, "Deep Forest: Towards an alternative to deep neural networks," in *Proc. Int. Joint Conf. Artif. Intell.*, 2017, pp. 3553–3559.

[32] QQ browser, "Tencent technology co.,ltd," 2021. Accessed: Mar. 2021. [Online]. Available: https://browser.qq.com/

[33] Toutiao, "ByteDance," 2021. Accessed: Mar. 2021. [Online]. Available: https://app.toutiao.com/

[34] Kwai, "Beijing kuaishou technology co., ltd," 2021. Accessed: Mar. 2021 from https://www.kwai.com/

[35] TikTok, "ByteDance," 2021. Accessed: Mar. 2021. [Online]. Available: https://tiktok.com/

[36] Bilibili, "Bilibili Inc," 2021. Accessed: Mar. 2021. [Online]. Available: https://app.bilibili.com/

[37] X. Video, "ByteDance," 2021. Accessed: Mar. 2021. [Online]. Available: https://www.ixigua.com/app/

[38] Kuwo, "Beijing KuWo technology co., ltd," 2021. Accessed: Mar. 2021 [Online]. Available: http://www.kuwo.cn/down

[39] Himalaya, "Shanghai himalaya technology co., ltd," 2021. Accessed: Mar. 2021 [Online]. Available: https://www.ximalaya.com/

[40] NetEase CloudMusic, "NetEase, Inc," 2021. Accessed: Mar. 2021 [Online]. Available: https://music.163.com/

[41] QQ Music, "Tencent, Inc," 2021. Accessed: Mar. 2021 [Online]. Available: https://y.qq.com/

[42] B. Netdisk, "Baidu, Inc," 2021. Accessed: Mar. 2021 [Online]. Available: https://pan.baidu.com/

[43] Soul, "Shanghai arbitrary door technology co., ltd," 2021. Accessed: Mar. 2021 [Online]. Available: https://www.soulapp.cn/

[44] Y. Li, C. Peng, Z. Yuan, J. Li, H. Deng, and T. Wang, "Mobileinsight: Extracting and analyzing cellular network information on smartphones," in *Proc. 22nd Annu. Int. Conf. Mobile Comput. Netw.*, 2016, pp. 202–215.

[45] M. Q. Khan, "Signaling storm problems in 3GPP mobile broadband networks, causes and possible solutions: A review," in *Proc. Int. Conf. Comput., Electron. Commun. Eng.*, 2018, pp. 183–188.

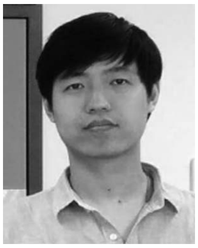
[46] Y. Qin, D. Song, H. Chen, W. Cheng, G. Jiang, and G. Cottrell, "A dual-stage attention-based recurrent neural network for time series prediction," in *Proc. Int. Joint Conf. Artif. Intell.*, 2017,pp. 2627–2633.

Zhi Ding received the BS degree from Zhejiang University, China, in 2021. He is currently working towards the master's degree with Zhejiang University, China. His research interests include 5G networks, edge task scheduling and video QoE.

Yuxiang Lin received the BS degree from the Xidian University, China, in 2016, and the PhD degree from Zhejiang University, China, in 2021. He is currently working with Alibaba Group. His research interests include smart NIC, 5G networks, mobile sensing and ubiquitous computing.

Weifeng Xu received the BS degree from the Zhejiang University of Technology, China, in 2022. He is currently working towards the master's degree with Zhejiang University, China. His research interests include 5G networks and knowledge distillation.

Jiamei Lv received the BS degree from the College of Information Science and Engineering, Ningbo University, in 2018, and the PhD degree from the College of Computer Science, Zhejiang University, in 2023. She is currently a researcher with the School of Software Technology, Zhejiang University. Her research interests include the Internet of Things, blockchain, and 5G networks.



Yi Gao (Member, IEEE) received the BS and PhD degrees from Zhejiang University, in 2009 and 2014, respectively. He is currently a full professor with Zhejiang University, China. From 2015 to 2016, he visited McGill University as a visiting scholar. His research interests include Internet of Things, edge computing, wireless and mobile computing. He is a member of the ACM.

Wei Dong (Member, IEEE) received the BS and PhD degrees from Zhejiang University, in 2005 and 2011, respectively. He is currently a full professor with the College of Computer Science at Zhejiang University. He leads the Emerging Networked Systems (EmNets) lab in Zhejiang University. He has published more than 150 papers in prestigious conferences and journals including MobiCom, INFOCOM, ICNP and *IEEE/ACM Transactions on Networking*, *IEEE Transactions on Mobile Computing*, etc. His research interests include Internet of Things and sensor networks, wireless and mobile computing. He is a member of ACM.