CASTLE over the Air: Distributed Scheduling for Cellular Data Transmissions

Jihoon Lee
University of Colorado Boulder
jihoon.lee-1@colorado.edu

Jinsung Lee
University of Colorado Boulder
jinsung.lee@colorado.edu

Youngbin Im
University of Colorado Boulder
youngbin.im@colorado.edu

Sandesh Dhawaskar Sathyanarayana
University of Colorado Boulder
sadh0344@colorado.edu

Parisa Rahimzadeh
University of Colorado Boulder
parisa.rahimzadeh@colorado.edu

Xiaoxi Zhang
Carnegie Mellon University
xiaoxiz2@andrew.cmu.edu

Max Hollingsworth
University of Colorado Boulder
max.hollingsworth@colorado.edu

Carlee Joe-Wong
Carnegie Mellon University
cjoewong@andrew.cmu.edu

Dirk Grunwald
University of Colorado Boulder
dirk.grunwald@colorado.edu

Sangtae Ha
University of Colorado Boulder
sangtae.ha@colorado.edu

ABSTRACT

This paper presents a fully distributed scheduling framework called CASTLE (Client-side Adaptive Scheduler That minimizes Load and Energy), which jointly optimizes the spectral efficiency of cellular networks and battery consumption of smart devices. To do so, we focus on scenarios when many smart devices compete for cellular resources in the same base station: spreading out transmissions over time so that only a few devices transmit at once improves both spectral efficiency and battery consumption. To this end, we devise two novel features in CASTLE. First, we explicitly consider inter-cell interference for accurate cellular load estimation. Based on our observations, we exploit the RSRQ (Reference Signal Received Quality) and SINR as features in a machine learning algorithm to accurately estimate the cellular load. Second, we propose a fully distributed scheduling algorithm that coordinates transmissions between clients based on the locally estimated load level at each client. Our formulation for minimizing battery consumption at each device leads to an optimized backoff-based algorithm that fits practical environments. To evaluate these features, we prototype a complete LTE system testbed consisting of mobile devices, eNodeBs, EPC (Evolved Packet Core) and application servers. Our comprehensive experimental results show that CASTLE's load estimation is up to 91% accurate, and that CASTLE achieves higher spectral efficiency with less battery consumption, compared to existing centralized scheduling algorithms as well as a distributed CSMA-like protocol. Furthermore, we develop a light-weight SDK that can expedite the deployment of CASTLE into smart devices and evaluate it in a commercial LTE network.

CCS CONCEPTS

• **Networks** → **Mobile networks**; *Packet scheduling*; *Network measurement*;

KEYWORDS

LTE; Cell Load; Distributed Scheduling; Energy Saving

ACM Reference Format:
Jihoon Lee, Jinsung Lee, Youngbin Im, Sandesh Dhawaskar Sathyanarayana, Parisa Rahimzadeh, Xiaoxi Zhang, Max Hollingsworth, Carlee Joe-Wong, Dirk Grunwald, and Sangtae Ha. 2019. CASTLE over the Air: Distributed Scheduling for Cellular Data Transmissions. In *The 17th Annual International Conference on Mobile Systems, Applications, and Services (MobiSys '19), June 17–21, 2019, Seoul, Republic of Korea.* ACM, New York, NY, USA, 13 pages. https://doi.org/10.1145/3307334.3326086

1 INTRODUCTION

The volume of mobile data traffic increases exponentially, partly due to the rapidly growing number of mobile-connected devices active on the Internet. For instance, there will be 11.6 billion mobile-connected devices by 2021. Among these devices, the number of Machine-to-Machine (M2M) connections is expected to grow at a 34% CAGR (compound annual growth rate) and reach 3.3 billion by 2021 [7]. To cope with the traffic demand from these devices, recent work has focused on developing new low power wide area communication technologies for Internet-of-Things (IoT) devices. Such technologies are designed to meet the requirements of long battery life, low device cost, and extended coverage [3], and so generally support low data rates with unreliable connectivity. Many

MobiSys '19, June 17–21, 2019, Seoul, Republic of Korea

ACM ISBN 978-1-4503-6661-8/19/06…$15.00
https://doi.org/10.1145/3307334.3326086

"smarter" services, however, necessitate reliable delivery of large volumes of data, e.g., for applications like autonomous driving, smart cities/infrastructures and connected healthcare [8]. Indeed, by 2021 most mobile devices will have some form of cellular connectivity in order to meet these performance needs [7].

Today's cellular networks are, however, ill equipped to handle the growing number of cellular devices and their data demand. Firstly, a huge number of simultaneously active smart devices[1] can lead to very low per-device throughputs and even short-term unfairness [32]. Specifically, the scheduler implemented in the cellular base station (BS) enforces fair scheduling irrespective of device and application types [5]; it is unaware of the device type and the payload cannot be decoded in the BS due to upper-layer end-to-end encryption (e.g., HTTPS). Secondly, smart devices can incur significant battery consumption due to enlarged transmission times when the cellular load is very high or congested [6].

Given these limitations, many devices would prefer to utilize the network when the cellular load is low and few other devices are active. By doing so, they can significantly increase spectral efficiency, and thus help the network cope with the increased amount of data traffic. However, this requires client devices to estimate the cellular load in real time with negligible energy expenditure, which presents a research challenge. For instance, the cellular load is impacted by the amount of traffic generated by other devices in the network and changes dynamically over time [16], as well as across geographical locations and environmental conditions [29].

Some prior work has presented client-based load estimation in commercial cellular networks, e.g., [6, 28]. The basic idea is that the Reference Signal Received Quality (RSRQ) measured at the mobile client can imply the current load level of the cell serving the client. In this work, we experimentally confirm a high correlation between RSRQ and achievable throughput: user devices with higher RSRQ achieve higher throughput. However, this correlation breaks down at the edge of the cell, where the device may experience interference from neighboring cells. We explicitly consider inter-cell interference, empirically demonstrating that the RSRQ is degraded by interference from the neighbor cells, but that considering the observed Signal to Interference and Noise Ratio (SINR) as well as RSRQ can lead to accurate load estimates even in the cell edge. Our experimental evaluation shows that we can attain 83–91% accuracy in the presence of inter-cell interference.

Even if the cellular load is known, distributed coordination mechanisms are needed to make sure that too many devices do not try to take advantage of low-load times, thus ensuring the load remains low and yielding efficient transmissions. In typical cellular networks, the BS runs a centralized scheduling algorithm for the attached clients based on its own policy, e.g., proportional fairness [22]. By taking link quality and each client's historical throughput into account, it allocates fewer time/frequency resources to existing clients and more to newly joined clients. Thus, as the number of competing devices increases, devices will likely receive downlink data in a discontinuous and bursty manner, leading to high energy consumption, as analyzed in [11]. In addition, existing centralized scheduling algorithms are unaware of application requirements like transfer deadlines and are not designed to minimize device energy expenditure, e.g., maximizing short-term fairness in throughput instead. Hence, a centralized scheduling approach requires significant modification of the Long Term Evolution (LTE) standard to achieve high energy efficiency of devices.

To avoid the disadvantages of typical centralized scheduling algorithms, we design a fully distributed scheduling framework called CASTLE (Client-side Adaptive Scheduler That minimizes Load and Energy) that coordinates transmissions among different clients based on the load level estimated at each device. Our algorithm is qualitatively similar to LoadSense's Peek-n-Sneak protocol [6], in which each client performs a simple CSMA-like operation before obtaining a scheduling opportunity from the BS based on the binary estimation of cellular load. However, we take a *theory-driven* approach that guarantees optimal performance. In particular, we formulate and solve the problem of minimizing the battery consumption of each mobile device, subject to an achievable throughput constraint based on the aforementioned load estimation considering *inter-cell interference*. Our estimation algorithms allow us to dynamically adapt the solution to this optimization problem as the load changes, unlike LoadSense's long observation period of 3 seconds.

We have implemented CASTLE as a light-weight software development kit (SDK) on the Android platform that provides cellular load estimation as well as a distributed scheduling service. Our experimental evaluations show some key results: (i) CASTLE drastically reduces the energy expenditure of User Equipments (UEs) – the downloading time with CASTLE is reduced by 5–53% compared to Peek-n-Sneak, and by 78–87% compared to centralized proportional fair scheduling algorithms; (ii) CASTLE improves the spectral efficiency by 12–17% and 20–57%, compared to Peek-n-Sneak and centralized scheduling, respectively.

Our paper makes the following contributions.

- **We propose a theoretical model for the cell load estimation and a novel scheduling algorithm** (§3) for smart devices in cellular networks. We refine an analytical model of estimating the cellular load to explicitly consider inter-cell interference, unlike existing work such as LoadSense [6]. The scheduling algorithm minimizes devices' battery expenditure in idealized conditions, thus guaranteeing maximal spectral efficiency. Furthermore, each device runs the algorithm in a fully distributed manner using key parameters derived from our optimization framework.
- **We provide a CASTLE SDK** (§4). The CASTLE SDK is available in [17], consisting of an API, channel inference module and scheduler module for Android devices. To minimize the SDK's energy expenditure, as is needed for smart applications, we exploit several techniques including a lookup-based load inference using passive UE measurements.
- **We demonstrate via our fully controllable LTE testbed and a commercial LTE network** (§5) that CASTLE can accurately estimate the cellular load in comparison with the ground truth directly obtained from the eNodeB, which has not been done so far. We also confirm that CASTLE outperforms other schemes such as Peek-n-Sneak and other centralized schedulers in terms of spectral efficiency and

[1] We define "smart devices" as devices with at least a 3G cellular data connection. We also use a device, client, user, and user equipment interchangeably in the rest of our paper.

battery consumption. CASTLE's benefits hold not only in static scenarios but also in case of UE mobility.

In §2, we present our system design and use cases; §3-5 describe our research contributions. §6 discusses remaining issues, and §7 presents related work. We conclude the paper in §8.

2 CASTLE DESIGN

We propose CASTLE, a distributed scheduling framework that jointly optimizes the spectral efficiency of LTE networks and the battery consumption of smart devices. We discuss CASTLE's design challenges (§2.1) before presenting CASTLE's architecture in §2.2.

2.1 Design Challenges

In designing our load- and energy-aware scheduling framework, we solve several technological challenges:

Theory and practice for estimating the current cell load. Our estimation algorithm for the cell load is based on a theoretical inference model, and we also aim for the algorithm to be practical considering inter-cell interference and computational complexity.

- **Consider interference.** Adding base stations has historically been an effective method to increase the capacity of cellular networks, and dense deployments of small cells are a key feature of future wireless networks. Users in such dense base station deployments naturally incur inter-cell interference, which is a major limiting factor in LTE networks. Some prior work has presented client-based load estimation. LoadSense [6] is not generally applicable, since it requires a prior per-site measurement to estimate the cell load. CLAW [28] quantitatively models the relation between the cell load and physical-layer statistics, which does not require the prior per-site measurement. However it may not be practical in multi-cell environment, since no interference is considered. CASTLE analytically predicts the achievable throughput of a UE based on RSRP, RSRQ, and SINR, accurately estimating the cell load requires a machine-learning (ML) approach to take into account the inter-cell interference.
- **Maintain low computational complexity.** Running such an ML algorithm in real time will incur excessive computational overhead. For some small-sized devices, a high computational overhead may not be acceptable. Therefore, CASTLE takes a low computational complexity approach which uses a *Cell Load Lookup Table* with pre-trained results.
- **Be practical.** CASTLE SDK should run on the existing Android OS, and the SDK needs to be easily integrated with any client application using only a few lines of code. piStream [27] and CQIC [19] assumed a cross-layer interaction with the LTE PHY, which is not easily applicable without obtaining privileged control over Android's subsystem.

Distributed transmission scheduling for multiple devices. Our proposed scheduling algorithm is inspired by combining Carrier Sense Multiple Access (CSMA) with CASTLE's cell load estimation.

- **Maximizing battery efficiency.** Our distributed scheduling algorithm aims to minimize UE energy expenditure, as long as the traffic is sent within a given deadline. Minimizing the total energy expenditure of transferring a fixed amount of data is equivalent to minimizing the average energy per bit. Since the energy per bit at any time decreases as the UE achieves higher instantaneous throughput, the problem is equivalent to maximizing the average throughput over all time slots in which UE is active. Given the estimated cell load, the UE needs to decide when to start downloading. Transferring data at a congested time requires more power expenditure.
- **Application-aware scheduling.** Since a UE typically runs multiple applications at the same time, the CASTLE SDK should support per-application scheduling service with different performance requirements (e.g., the data amount to be downloaded, deadline, etc.).
- **Fully distributed coordination.** Scheduling multiple flows in a distributed manner for this objective, however, is quite challenging since the scheduling needs to operate based solely on local information available on each UE.
- **Coexistence.** The coexistence of CASTLE's scheduling and the existing LTE centralized scheduling schemes should be considered. To verify this feature, we used randomly generated background traffic that was not scheduled by CASTLE in §5.4, and we also tested CASTLE on a commercial network in §5.5.

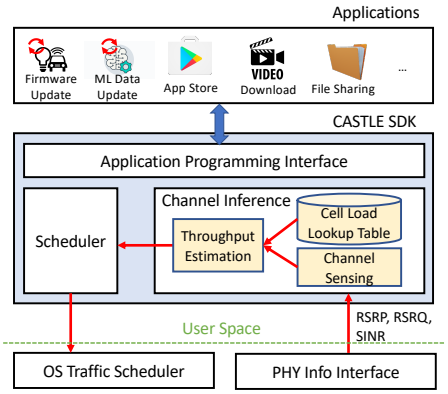


Figure 1: CASTLE architecture. Applications can leverage the CASTLE API to schedule their traffic based on accurate estimates of the cell load.

2.2 Architecture

Figure 1 illustrates the architecture of CASTLE, which is provided as an SDK. Applications can interact with the CASTLE API to benefit from our energy-efficient, load-aware download scheduling algorithm; §6 outlines some types of applications that can particularly benefit from CASTLE. The expected throughput is estimated by the Channel Inference module and fed into the Scheduler module. The Channel Sensing module periodically obtains PHY-layer information (e.g., RSRP, RSRQ, CQI, and SINR)[2]. To minimize the computational overhead, CASTLE uses the *Cell Load Lookup Table*, which populates the cell load level for each RSRP, RSRQ, CQI, and SINR pair by using both our analytical models in §3.4 and machine learning for the inter-cell inference scenario. This enables

[2]The *CellSignalStrengthLte* class on Android provides the LTE PHY-layer information.

an efficient $O(1)$ lookup. The Scheduler module then decides the transmission schedule for the application traffic according to the algorithm presented in §3.2.

3 SYSTEM MODEL

We first provide an overview of CASTLE's scheduling in §3.1. To optimize the parameters used in the scheduling algorithm, we formulate the problem to minimize the UEs' total energy expenditure in §3.2. We then transform this problem into one where each UE estimates its expected throughput and then finds the optimal backoff parameters to be used for its data transfer, in §3.3. §3.4 presents an algorithm to estimate the UE throughput, which §4 extends to handle inter-cell interference.

3.1 CASTLE's Scheduling

Unlike centralized scheduling, a distributed scheduling algorithm depends on random access and thus needs to determine when to send and how long to wait when there is a collision (i.e., when multiple nodes transmit at the same time), analogous to CSMA (Carrier Sense Multiple Access). However, unlike CSMA, in which the transmission happens after a "blind" random backoff based on the number of collisions, CASTLE lets each UE make an informed decision on the backoff parameters based on an estimate of its expected throughput.

Figure 2 illustrates the operation of CASTLE by two UEs[3]. The expected throughput $A(t)$ is estimated every one second time slot, and is compared to the optimal threshold A^*. If the expected throughput is greater than or equal to the threshold, the download can be started. However, there may be many UEs that estimate their expected throughput condition is now satisfied, which could lead to more contentions. Therefore, CASTLE waits for a random backoff interval that is selected uniformly in the range $[0, Rb_{\max}]$ before starting the transmission. After the expiry of this backoff interval, the UE checks this condition again before starting its transmission. Otherwise, CASTLE cancels its transfer and waits until the expected throughput becomes higher than the threshold.

3.2 Problem Formulation

We consider the problem of N UEs connected to an LTE eNodeB over a series of T time slots, indexed by $t = \{1, 2, ..., T\}$. We term a UE "*active*" at time t if it is downloading data at that time[4]. To reflect the resource allocation regime in existing cellular networks, we consider that the BS uses proportional fair scheduling to allocate resources to all active UEs within any given time slot t. Note that the BS' scheduling is done at a much finer timescale (i.e., 1 ms granularity) than the UE activity [5]. Suppose that there are a total number of resource blocks (RBs) that can be allocated at time slot t (denoted by $l_{\text{total}}(t)$), and that each UE has one download flow to complete.

At the beginning of each time slot, each UE can estimate the current load on the network using the method in §3.4. We denote this estimate as $l_{\text{used}}(t)$, with the condition of $0 \le l_{\text{used}}(t) \le l_{\text{total}}(t)$;

[3]CASTLE's scheduling works not on a per-UE basis, but rather on a per-application basis. For simplicity, a single application is assumed in Figure 2.
[4]In this work, we do not consider a upload scenario as most cellular traffic happens in the downlink as measured in [16].

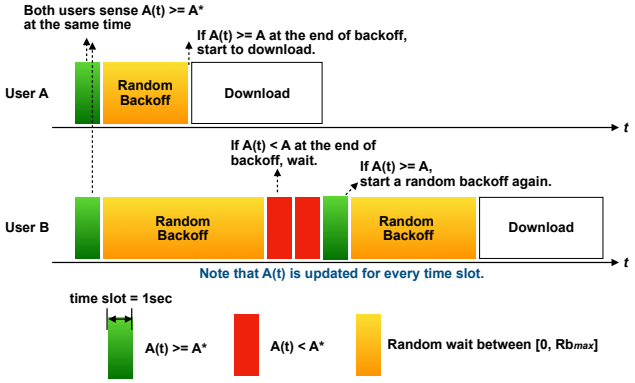


Figure 2: CASTLE's scheduling: Users A and B check the condition of $A(t) \ge A^*$ to start the download after a random backoff. User B cannot start the download after the backoff since User A's download has led User B to estimate that its $A(t)$ is less than A^*.

the load is the average number of allocated RBs per each 1 ms LTE subframe during the time slot t. Given the estimated $l_{\text{used}}(t)$ and knowing that other UEs may be making the same decision, the UE must now decide when to start downloading. While it would like to receive as much data as possible, which could be achieved by downloading in every time slot, transferring data at a congested time requires more power expenditure; thus, the UE would prefer to receive only at less congested times.

To trade off between these objectives, the UE should make use of its (known) received signal strength, which we represent as $s_n(t)$ for UE n at time t; this can be further interpreted as the number of bytes transmitted per RB allocated to the UE as specified in [2]. Let ϕ_n denote the total amount of data that UE n would like to transfer. Let DDL_n denote the number of time slots between the time UE n has the data to transfer and its deadline. Since ϕ_n and DDL_n are application-specific parameters, we can assume that $\frac{\phi_n}{DDL_n}$ is the same, for all UEs n using the same application.

Each UE minimizes its energy expenditure by solving:

$$\min \quad \sum_{t=1}^{T} e\left(x_n(t), g(s_n(t), x(t), l_{\text{used}}(t))\right), \tag{1}$$

$$\text{s.t.} \quad \sum_{t=1}^{T} x_n(t) \times g(s_n(t), x(t), l_{\text{used}}(t)) \ge \phi_n, \tag{2}$$

$$CT_n \le DDL_n, \forall n, \tag{3}$$

where $x_n(t)$ is an indicator variable on whether UE n downloads at time t ($x_n(t) = 1$) or not ($x_n(t) = 0$), $e(\cdot)$ denotes the energy expenditure of the UE in time slot t, $g(\cdot)$ denotes the number of bytes downloaded as a function of UE n's current signal strength and the cell load in time slot t, and CT_n denotes the completion time for downloading ϕ_n. Here we use $x(t)$ to denote an N-dimensional decision vector for all UEs in slot t.

To solve (1–3), we note that $\frac{\sum_{t=1}^{T} x_n(t) g(s_n(t), x(t), l_{\text{used}}(t))}{\sum_{t=1}^{T} x_n(t)}$ represents the average amount of data downloaded over all time slots in which the UE is active. Minimizing the total energy expenditure of transferring a fixed amount of data is equivalent to minimizing the average energy per bit. Since the energy per bit at any time decreases as the UE achieves higher instantaneous throughput, (1–3)

is equivalent to maximizing the average throughput over all time slots in which UE is active.

We can now parameterize our strategy for deciding $x_n(t)$, i.e., whether UE n should transmit or not at each time t. At each time t, UE n first estimates the total amount of data $A_n(t)$ it can download as

$$A_n(t) = s_n(t)(l_{\text{total}}(t) - l_{\text{used}}(t)). \tag{4}$$

The UE then compares $A_n(t)$ with an optimized threshold A_n^* (which is derived in §3.3). If $A_n(t) \geq A_n^*$, it randomly picks a backoff time $Rb_n(t)$, which is drawn from a uniform distribution over $[0, 1, ..., Rb_{\max}]$ and is *i.i.d.* for all UEs over time. Let $\overline{Rb} = Rb_{\max}/2$ denote the mean random backoff time. For long-term fairness, we require that each UE is active for at most MTT (Maximum Transfer Time) consecutive time slots. Otherwise (i.e., $A_n(t) < A_n^*$), the UE does not download data and continues to sense in the next time slot.

Since the $x_n(t)$ are determined by A_n^* and $Rb_{\max}$ (equivalently $\overline{Rb}$), we now optimize these parameters. Suppose $A_n(t)$ is a random variable uniformly drawn from the range $[a, b]$ and is *i.i.d.* over time. Thus, the expected amount of data downloaded in each slot t (conditioned on the fact $A_n(t) \geq A_n^*$ and that the UE is not in the middle its backoff time) is as follows:

$$\begin{aligned}\mathbb{E}\{g(\cdot)\} &= \frac{\mathbb{E}\{\text{total data to be downloaded in slot } t\}}{\mathbb{E}\{\text{total number of active UEs in slot } t\}} \\ &= \frac{(A_n^* + b)/2}{\left((N(t)-1) \times \frac{MTT}{\overline{Rb}+MTT} + 1\right)},\end{aligned} \tag{5}$$

where $N(t)$ denotes the expected total number of UEs at time t and we use the fact that each UE is active for an average $MTT/\left(\overline{Rb} + MTT\right)$ time slots. We assume that $N(t)$ is a constant in the rest of the formulation. This makes sense since the number of active UEs varies significantly in a longer timescale, not in a shorter timescale, as measured in [16]. Let $\mathbb{E}\{CT_n\}$ denote the expected completion time of UE n, which can be approximated as follows:

$$\begin{aligned}\mathbb{E}\{CT_n\} &= \frac{\phi_n}{\mathbb{E}\{g(\cdot)\}\mathcal{P}\{A_n(t) \geq A_n^*\}\mathcal{P}\{\text{UE } n \text{ is downloading}\}} \\ &= \frac{\phi_n}{\frac{A_n^*+b}{2\left((N(t)-1)\cdot\frac{MTT}{\overline{Rb}+MTT}+1\right)}} \times \frac{1}{\frac{b-A_n^*}{b-a}} \times \frac{1}{\frac{MTT}{\overline{Rb}+MTT}}.\end{aligned} \tag{6}$$

From the deadline requirement for data delivery, we should guarantee that $\mathbb{E}\{CT_n\} \leq DDL_n, \forall n$. Rearranging the above relationship, we get the constraints:

$$\frac{MTT}{\overline{Rb} + MTT} \geq \frac{1}{\frac{DDL_n(b^2 - A_n^{*2})}{2\phi_n(b-a)} - N(t) + 1}, \tag{7}$$

$$\frac{DDL_n(b^2 - A_n^{*2})}{2\phi_n(b-a)} > N(t) - 1. \tag{8}$$

We finally remark that since $N(t)$ is a constant the throughput achieved by the active UE (i.e., Eq. (5)) is a constant, as well. Thus, our original problem (1–3) is equivalent to solving the following:

$$\begin{aligned}\max_{A_n^*, \overline{Rb}} \quad & \frac{A_n^* + b}{2\left((N(t)-1)\cdot\frac{MTT}{\overline{Rb}+MTT}+1\right)}, \\ \text{s.t.} \quad & (7), (8), \forall t = 1, \cdots, T, \forall n.\end{aligned} \tag{9}$$

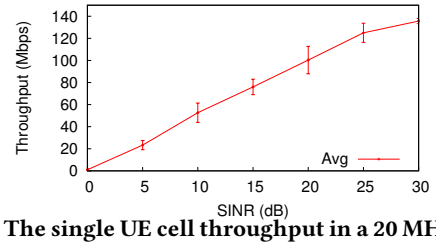


Figure 3: The single UE cell throughput in a 20 MHz channel. Measured with a Motorola G5 Plus, Samsung Galaxy S5, and Samsung Galaxy Note4.

3.3 Calculation of A_n^* and $\overline{Rb}$

Combining Eq. (6) with Eqs. (9) and (7), we know that for each UE n, the optimal $\overline{Rb}$ and A_n^* should satisfy $\mathbb{E}\{CT_n\} = DDL_n$. Hence, we replace the term $\frac{MTT}{\overline{Rb}+MTT}$ in (9) by $\left(\frac{DDL_n(b^2-A_n^{*2})}{2\phi_n(b-a)} - N(t) + 1\right)^{-1}$ and solve for A_n^* by maximizing (9) with the single variable A_n^*. If $\frac{\phi_n}{DDL_n}$ is the same for all n, then A_n^* will be the same for all UEs. Given the optimal A_n^*, the mean random backoff time $\overline{Rb}$ can be directly derived from $\frac{MTT}{\overline{Rb}+MTT} = \left(\frac{DDL_n(b^2-A_n^{*2})}{2\phi_n(b-a)} - N(t) + 1\right)^{-1}$. Thus, we obtain the following:

$$\overline{Rb} = \max\left\{MTT \times \left(\frac{DDL_n(b^2 - A_n^{*2})}{2\phi_n(b-a)} - N(t)\right), 0\right\}.$$

Each parameter in this equation is known to the UE, except for $N(t)$. In §5.4, we show that the UE can estimate $N(t)$ accurately enough to achieve good performance in practice.

3.4 Estimation of $A_n(t)$

We estimate the amount of data that can be downloaded, $A_n(t)$, with two parameters: (i) the transport block size per RB based on the received signal strength $s_n(t)$ and (ii) the number of available RBs $l_{\text{unused}}(t)$ from the relation $l_{\text{unused}}(t) = l_{\text{total}}(t) - l_{\text{used}}(t)$. In §4, we modify our estimate of $l_{\text{unused}}(t)$ to account for inter-cell interference.

Once an LTE UE measures its wireless channel, it obtains the RSSI (Receive Strength Signal Indicator), RSRP (Reference Signal Received Power), and SINR (Signal to Interference and Noise Ratio) [1]. The RSSI is the linear average of the total received power observed over the total number of resource blocks in a channel, by the UE from all sources, including co-channel serving and non-serving cells, adjacent channel interference, thermal noise etc. The RSRP is the average power of the resource elements that carry cell-specific reference signals. The SINR is defined as the power of a certain signal of interest divided by the sum of the power of interference and noise, which is a measure of channel quality. In LTE, the channel quality indicator (CQI) of a UE is determined by the SINR of the UE and is reported to the serving eNodeB. Then, the eNodeB decides a modulation and coding scheme (MCS) and transport block size (TBS) of the UE based on the CQI reported [2].

From Eq. (4), $s_n(t)$ represents the bitrate per RB at time slot t for UE n. Since $s_n(t)$ can be directly derived from the index of TBS, we assume that $s_n(t)$ can be represented as a simple linear function of $\text{SINR}_n(t)$. To verify this, we measured a single UE cell throughput with a range of SINR values in our LTE testbed. Because a link adaptation algorithm is implementation-specific, three different

Table 1: Values of $k_{j,l}$ for different number of antennas

k with number of antennas j	$l = idle$	$l = full$
$k_{1,l}$	2	12
$k_{2,l}$	4	20
$k_{4,l}$	4	36

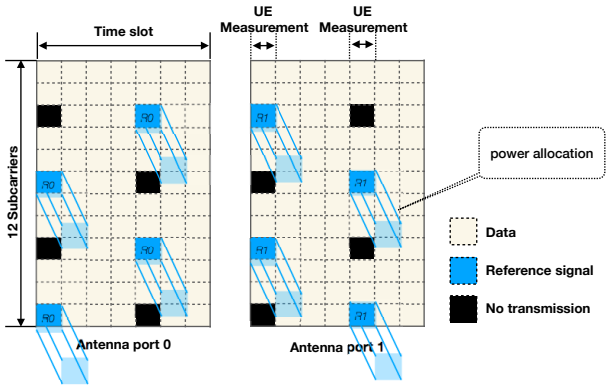


Figure 4: One RB consists of 12 subcarriers (12 × 15 KHz) in the frequency domain and one slot (0.5 ms) in the time domain. Power is allocated only to 2 REs of the reference signal per antenna in a UE measurement.

models of UEs are tested, and Figure 3 verifies that the throughput and SINR are proportional. Therefore, $s_n(t)$ can be rewritten as

$$s_n(t) \approx \frac{\eta}{N_{RB}} \times \frac{\text{SINR}_n(t)}{\text{SINR}_{\max}}, \tag{10}$$

for some constant η, where N_{RB} is the total number of RBs and $\text{SINR}_{\max}$ is the maximum SINR value[5].

We will now estimate the number of available (i.e., not used) resource blocks at time slot t, $l_{\text{unused}}(t)$, in terms of the Reference Signal Received Quality (RSRQ). Prior work in LTE cell load analysis revealed that the RSRQ can be used to quantify the LTE channel resource utilization [6, 28]. In fact, RSRQ is the power ratio between RSRP and RSSI as follows:

$$\text{RSRQ} = N_{\text{RB}} \times \frac{\text{RSRP}}{\text{RSSI}}. \tag{11}$$

Figure 4 depicts the power allocation of a single RB with a 2 antennas configuration. An LTE eNodeB allocates radio resources to UEs in the unit of RBs. A resource element (RE) is the smallest resource unit in the LTE physical layer (the smallest box in Figure 4), which spans one OFDM symbol in time and one OFDM subcarrier in frequency. In the LTE MAC layer, the RB is the smallest allocation resource unit. Typically, each RB contains 7×12 REs as it spans across 7 time-domain symbols (one slot) and 12 frequency-domain subcarriers [2]. In the time domain, a one 1ms LTE subframe consists of two slots. An LTE frame has 10 subframes, which is 10ms. In the frequency-domain, there are 25 RBs for each 5 MHz channel bandwidth. For a 20 MHz channel, there will be 100 RBs.

When the cell is idle, only the power of reference signals is measured as shown in Figure 4; RSSI will be $4 \times$ RSRP from the two antennas[6]. Let α and β be the RSRQ values measured by a UE for an idle cell and a fully loaded cell, respectively. Then we have

$$\alpha = \frac{N_{\text{RB}} \cdot \text{RSRP}}{k_{j,\text{idle}} \cdot \text{RSRP} \cdot N_{\text{RB}} + I}, \tag{12}$$

$$\alpha_{I=0} = (k_{j,\text{idle}})^{-1}, \tag{13}$$

where $k_{j,idle}$ is a constant as shown in Table 1 representing the number of reference signal REs in the duration of UE measurement for the number of antennas j, and I is the interference power from other signals. If the interference is ignored, it becomes much simpler so that α will be $1/k_{j,\text{idle}}$.

On the other hand, there are a total of 16 data REs and 4 REs of reference signals when the cell is fully loaded. Then β can be expressed as

$$\beta = \frac{N_{\text{RB}} \cdot \text{RSRP}}{k_{j,\text{full}} \cdot \text{RSRP} \cdot N_{\text{RB}} + I}, \tag{14}$$

$$\beta_{I=0} = (k_{j,\text{full}})^{-1}, \tag{15}$$

where $k_{j,\text{full}}$ is the number of all available REs except for some REs that are not allowed to be used in the 3GPP standard[7]. If we ignore the interference term (I), Eqs. (13) and (15) are inversely proportional to the RB usage. Therefore, $l_{\text{unused}}(t)$ can be written as

$$l_{\text{unused}}(t) \approx N_{\text{RB}} \times \frac{\text{RSRQ}(t) - \beta}{\alpha - \beta}. \tag{16}$$

From Eqs. (4), (10) and (16), we estimate the amount of data that can be downloaded for user n, which is expressed as

$$A_n(t) \approx \eta \times \frac{\text{SINR}_n(t)}{\text{SINR}_{\max}} \times \frac{\text{RSRQ}(t) - \beta}{\alpha - \beta}. \tag{17}$$

To account for interference, we can instead estimate $l_{\text{unused}}(t)$ in Eq. (16) with machine learning methods, as we do in the next section.

4 IMPLEMENTATION

LTE testbed: We built an end-to-end LTE testbed to evaluate CASTLE, as shown in Figure 5. The testbed consists of UEs, eNodeBs, EPC (Evolved Packet Core), and application servers. For the eNodeBs, we use two commercialized indoor LTE Band3 small cell products, Juni JL620 [15], which are connected to GPS to correct frequency offsets. We placed multiple UEs from various vendors and models inside a shield box, so that the UEs can communicate with the eNodeBs via a pair of antennas inside the box. The signal attenuator installed between the antenna located in the box and eNodeBs outside provides appropriate signal strengths to the UEs and further allows us to emulate a variety of RF situations, including interference from neighbor cells. For the EPC, we use the open source software from NextEPC [20]. The application servers gather cell-specific information from both the UEs and eNodeBs.

eNodeBs: In order to obtain the ground truth on the cell load, we modify the eNodeBs to report the actual cell load to the application servers at one second intervals. We also modify the eNodeB's scheduling algorithm to adjust a tradeoff between efficiency and fairness in the centralized scheduler. For this, we received technical support from the manufacturer.

[5]For most LTE UEs, 30 dB is typically assumed.

[6]The RSSI is measured only in the configured OFDM symbol that carries the reference signal.

[7]For the sake of simplicity, we assume that the power levels of all data REs are the same to the RSRP.

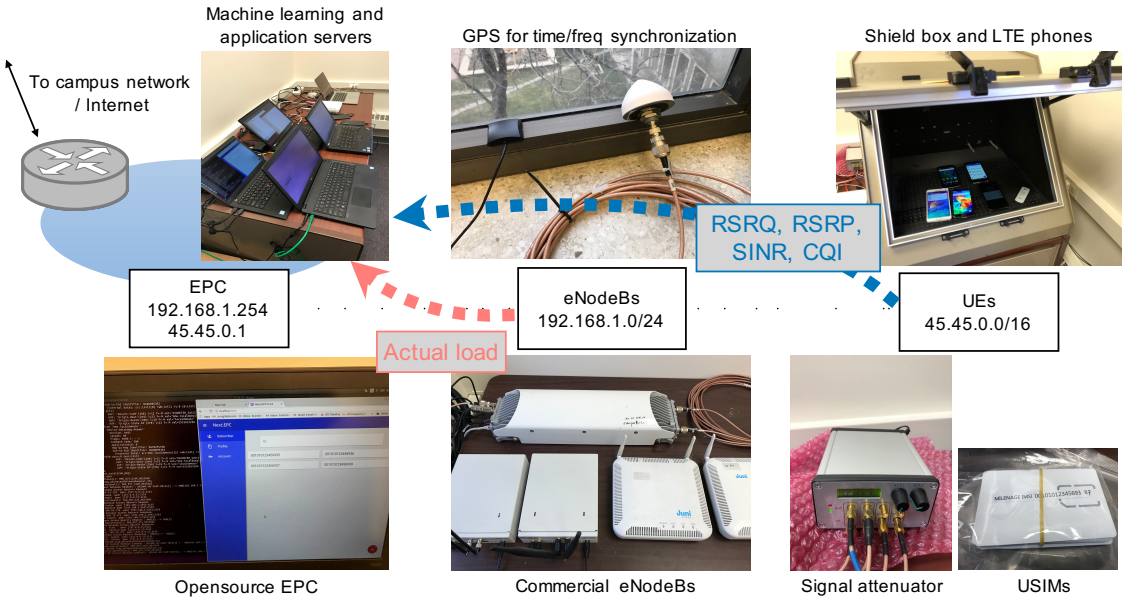


Figure 5: Our controllable end-to-end LTE network testbed composed of eNodeBs, UEs, an EPC, a signal attenuator, and a shield box. We used 2x LTE Band3 FDD eNodeBs and 10x LTE phones for the experiments.

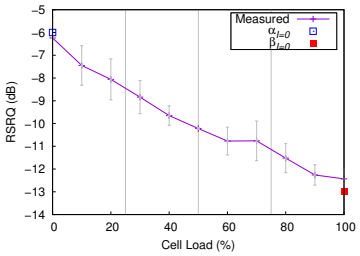


Figure 6: Measured average RSRQ in idle channel as a function of cell load.

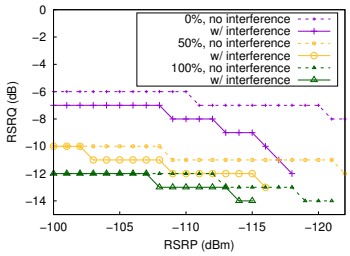


Figure 7: Measured RSRQ with co-channel interference.

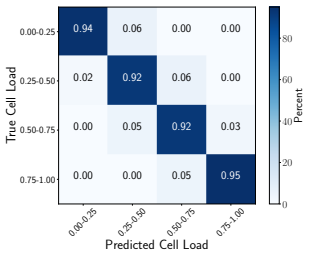


Figure 8: Normalized confusion matrix on cell load estimation.

Application servers: The application servers collect both actual cell load information from eNodeBs and measurement reports from each UE at the same time. The eNodeBs and UEs send report packets with GPS time stamps so that we can match them at the servers.

Cell load estimation with machine learning: As shown in Figure 6, we confirm that the measured RSRQ of a cell is a good indicator of the current load of the cell, regardless of the RSRP value [28], when there is no interference. With interference, however, RSRQ-based estimation is not accurate. Figure 7 shows the RSRQ measurement at the cell edge when there is co-channel interference from a neighboring cell. We set the neighbor cell to be fully loaded so that it can cause a significant level of interference at the edge of the serving cell. As defined in Eq. (11), we observe that the measured RSRQ is degraded as the RSSI increases due to the interference, which, in turn, creates a challenge of improving the accuracy of the cell load estimation in the presence of interference. This is quite common in real-world environments as measured in [16]. We further found that the SINR values are degraded compared to those in the idle channel. Therefore, in order to obtain the relationship between RSRP, RSRQ, SINR, and cell load, we trained a non-linear Support Vector Machine (SVM) classifier on our measurements. Our estimated idle portion of cell load corresponds to the ratio $\frac{\text{RSRQ}(t)-\beta}{\alpha-\beta}$ in Eq. (16).

The ML server runs the popular (Gaussian) Radial Basis Function (RBF) kernel of SVM. The range of the cell load ($0-100\%$) is divided into four smaller intervals: $0-25\%$, $25-50\%$, $50-75\%$, and $75-100\%$, each corresponding to a label in the classification algorithm. We use 60% of our measurements as training data and 20% for cross-validation. The accuracy of our trained SVM classifier on the remaining 20% of the data is about 93%, with the normalized confusion matrix shown in Figure 8. Note that, if we decrease the number of load classes to two (i.e., $0-50\%$ and $50-100\%$), as explored by LoadSense, the accuracy is increased to more than 97% with our test dataset. If we increase the number of load classes to more than 4, the accuracy will be degraded due to the coarse granularity of RSRQ values that we can get through the existing Android API.

CASTLE SDK: At each UE, the channel inference module of CASTLE maintains a mapping table (Cell Load Lookup Table) that inputs the RSRP, RSRQ, and SINR values and outputs the estimated load

```
Thread initialise=new Thread((Runnable) () → {
        cs.init(sharedPreferences,astar,b,ddl);//8000*1000
});
Thread download_decision=new Thread((Runnable) () → {
        while(true){
            int predict_class=cs.castle_predict_class_long();
            Log.d( tag: "application", msg: "\nPredicted class\t"+predict_class+"\n");
            decision=cs.castle_schedule();
            if(decision==1){
                Log.d( tag: "application", msg: "*****************Start downloading*" +
                        "**************");
            }
        }
```

Figure 9: An example code using CASTLE API: With only 3 lines, a basic CASTLE operation is supported.

class. The table consists of 45,198 entries where each entry is encoded with 2 bits to represent four classes of cell load. The total table size is about 89 KBytes which is reasonably small for today's smart devices. We also implement CASTLE's Scheduler module. It estimates the throughput using the RSRQ, RSRP, SINR, and mapping table at every time slot (i.e., 1 sec), and decides when the download starts. The CASTLE SDK is available in [17], which includes the following APIs:

- *castle_predict_class_short()* - predict the class with instantaneous RSRQ, RSRP, SINR measurements.
- *castle_predict_class_long()* - return the class mostly picked from the recent 10 consecutive measurements. In general, it is more accurate than *castle_predict_class_short()*, since it can ignore a UE measurement error.
- *castle_schedule()* - schedule a download. It compares the estimated throughput $A(t)$ and the threshold A^* to decide when the download starts. If $A(t) < A^*$, it stays silent for this time slot as long as its DDL has not yet come. If the DDL expires, it starts the download anyway.

Applications: Figure 9 shows an Android FTP client program we implemented that leverages CASTLE APIs.

5 EVALUATION

In this section, we first evaluate the performance of CASTLE's load estimation (§5.1). Next, we compare CASTLE with Peek-n-Sneak [6] and centralized scheduling at the eNodeB in terms of energy expenditure and spectral efficiency in stationary (§5.2) and mobile (§5.3) environments. Finally, we investigate the effect on the CASTLE performance of varying the number of participating users $N(t)$ in §5.4.

For the centralized scheduling algorithms at the eNodeB, two different proportional fair scheduling algorithms are tested: 'PFS-FAIR' and 'PFS-TPUT'. In general, PFS running in a commercial BS achieves a desirable tradeoff between average cell throughput and fairness; the Jain's fairness index [14] of the default eNodeB scheduler in our testbed is measured as 0.7. Compared to PFS, 'PFS-FAIR' improves fairness by strictly guaranteeing fair allocation of wireless resources to all users (Jain's index approaches 1), while 'PFS-TPUT' improves efficiency by assigning more wireless resources to the users who have the better channel quality (Jain's index is measured as 0.5). We modified the PFS to test PFS-FAIR and PFS-TPUT. For our experiments with mobility and co-channel interference, we construct a multi-cell environment.

Table 2 lists the experimental parameters used. As shown in Figure 3, a and b are set to 0 and 140 as the minimum and maximum achievable throughputs in Mbps, respectively. We consider the

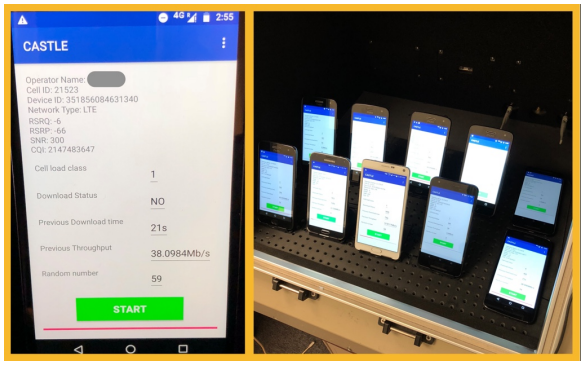
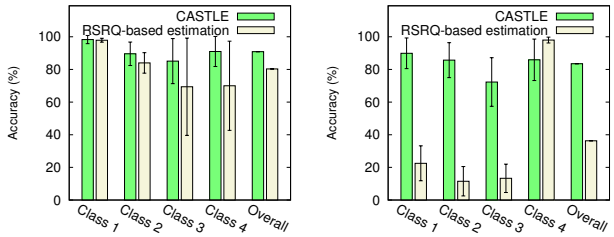


Figure 10: Evaluation of CASTLE, Peek-n-Sneak, and two PFS algorithms with 10 UEs (right) and a snapshot of one UE running CASTLE (left).

Table 2: Parameter settings for CASTLE evaluation

Parameter	Value	Parameter	Value
a	0	b	140
ϕ_n	800	DDL_n	800
MTT	1	$N(t)$	10



(a) Low interference (b) High interference

Figure 11: Evaluation of the load prediction: (a) CASTLE shows 91% accuracy. (b) CASTLE shows 83% accuracy.

download of a 100 Mbyte file (ϕ_n = 800 Mbits and DDL_n = 800 seconds), whereas a 1000 Mbyte file is used for the simulation (ϕ_n = 8000 and DDL_n = 32000). MTT controls the level of contentions like $\overline{Rb}$, which is set to 1. We set $N(t)$ to 10 unless stated otherwise. An error bar in the graphs represents a standard deviation.

5.1 Accuracy of Load Estimation

Test scenario: As explained in Section 4, we collected the ground truth of the cell load. We also used *castle_predict_long()* in each UE to report the cell load class to the application server so that we can compute the accuracy of CASTLE's load estimation. When the reported class exactly matches the actual cell load, it is counted as a success. We compare it with the RSRQ-based estimation [28]. We configure a full load at the neighbor cell so as to generate interference at cell edges. For each case, we took 48,000 measurements with multiple UEs locating at the cell center and edges.

Performance comparison: Figure 11a shows the accuracy of the estimated load class achieved by CASTLE at the cell center (i.e., low interference). CASTLE reveals 98% accuracy for Class 1, 90% for Class 2, 85% for Class 3, and 91% for Class 4. It outperforms the

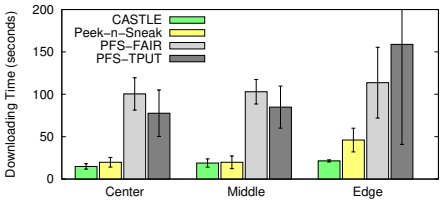


Figure 12: Comparison of average downloading time when UEs are stationary. CASTLE shows reduction of downloading time by 5–53% and 78–87%, compared to Peek-n-Sneak and two PFS algorithms, respectively.

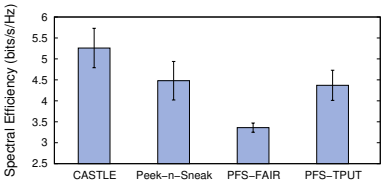


Figure 13: CASTLE shows better spectral efficiency when the UEs are stationary by 17%, 57%, and 20%, compared to Peek-n-Sneak, PFS-FAIR, and PFS-TPUT, respectively.

RSRQ-based estimation which shows 98%, 84%, 70%, and 70% for Classes 1, 2, 3, and 4, respectively. Since the RSRQ is described as a logarithmic function from Android API, more RSRQ values are mapped into the classes with a lower number. As a result, the accuracy of Class 1 is the best. The accuracy decreases as the class number increases, but Class 4 is improved because it can include lower RSRQ measurements than the theoretical bound (i.e., RSRQ(t) < β).

Figure 11b shows that the RSRQ-based estimation is inaccurate under high co-channel interference. It selects Class 4 for almost all cases, because of the lower RSRQ level due to interference. Therefore it shows 23%, 12%, and 13% accuracy for Classes 1, 2, and 3, respectively. CASTLE still shows good performance; 90%, 86%, 72%, and 86% for Classes 1, 2, 3, and 4, respectively.

In summary, as CASTLE is on average 91% accurate at the center and 83% accurate at edges, these results confirm that our ML-based load estimation technique accurately estimates the cell load for applications that wish to know the cellular load conditions.

5.2 Effect of Location

Test scenario: We chose three different positions for 10 UEs: Cell Center (-70 ≤ *RSRP* ≤ -60 dBm), Middle (-90 ≤ *RSRP* ≤ -80 dBm), Edge (-110 ≤ *RSRP* ≤ -100 dBm). We compare the performance of CASTLE, Peek-n-Sneak, and the two PFS algorithms by letting every UE repeatedly download 100 Mbytes files. Thus, during the whole experimental period, all UEs keep contending for cellular resources. Peek-n-Sneak originally assumes the use of LoadSense [6], which requires a prior measurement and training at the site. To isolate our results from the impact of the load estimation accuracy, we use the same ML-based load estimation as an input to both Peek-n-Sneak and CASTLE. In addition, Peek-n-Sneak assumed that the channel is idle when more than 7 Mbps throughput is expected. With our load estimation, Classes 1 to 3 are assumed idle for Peek-n-Sneak. For UEs, we used six Motorola G5 Plus, two Google Nexus 5X, one Samsung Galaxy S5, and one Samsung Galaxy Note4, as shown in Figure 10. The bandwidth of each cell is 20 MHz.

Energy expenditure: Figure 12 compares the average downloading time with each scheme. Note that the energy expenditure of the UE decreases as the total download time decreases, as modeled by [12]. First, the distributed methods, CASTLE and Peek-n-Sneak, significantly outperform the centralized methods (PFS-FAIR and PFS-TPUT); CASTLE especially reduces the downloading time by 78–87%. The downloading times in PFS-FAIR are similar for all locations, while the time in PFS-TPUT decreases as the UE approaches the cell center. In both cases, however, all UEs without load-awareness are competing for resources continuously, and the PFS enforces short-term fairness, which leads to low performance in terms of the energy minimization objective.

Second, we can see that CASTLE shows the best performance for all cases: 14.9 sec at Center, 18.9 sec at Middle, and 21.4 sec at Edge. Peek-n-Sneak shows better performance than PFS, but is still below CASTLE: 19.8 sec at Center and Middle, and 46.0 sec at Edge. CASTLE's improvement is due to the fact that CASTLE utilizes the optimal throughput threshold (i.e., A_n^*) derived from Eq. (9) to perform an optimally chosen random backoff, while Peek-n-Sneak simply depends on a binary decision (idle or busy) and performs a simple random backoff irrespective of the number of UEs. Thus, the performance of Peek-n-Sneak is not adaptive to the channel condition and available bandwidth; a UE in Peek-n-Sneak tends to initiate data transfers even when the cell is considerably loaded, incurring more competition. Compared with Peek-n-Sneak, the downloading time of CASTLE is reduced by 25% at Center, 5% at Middle, and 53% at Edge.

Spectral efficiency: Figure 13 shows the spectral efficiency measured during the experiments. For this spectral efficiency calculation, we first measured total number of bytes transferred by CASTLE, Peek-n-Sneak, PFS-FAIR, and PFS-TPUT at the application level. We recorded the total time spent for the data transfer (i.e., active time). In addition, we have averaged the RB usage over time reported by eNodeBs. The application-level spectral efficiency is then calculated as

$$SE = \frac{B_{\text{transfer}}}{T \times 20\ \text{MHz} \times \text{AvgRB}/100},$$

where B_{transfer}, T, and AvgRB are transferred data size (in bytes), transmission time, and averaged RB usage over T reported by the eNodeBs, respectively[8]. CASTLE shows the best spectral efficiency of 5.26 bits/s/Hz, while Peek-n-Sneak does not considerably outperform PFS-TPUT. Compared to Peek-n-Sneak, PFS-FAIR, and PFS-TPUT, CASTLE is an improvement by 17%, 57%, and 20%, respectively. In CASTLE, a data transfer can be deferred until the expected amount of data that can be downloaded is larger than the optimal threshold (i.e., $A(t) \geq A_n^*$), and therefore, the spectral efficiency can be increased.

5.3 Effect of Mobility

Test scenario: We next conduct experiments in a mobile environment, adding co-channel interference from the neighbor cell. Figure 14 illustrates our mobile testbed setup; two co-channel eNodeBs are installed, two UEs are stationary in Cell A, and the remaining

[8] An LTE channel with 20 MHz bandwidth has 100 RBs.

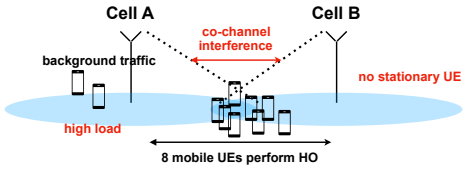
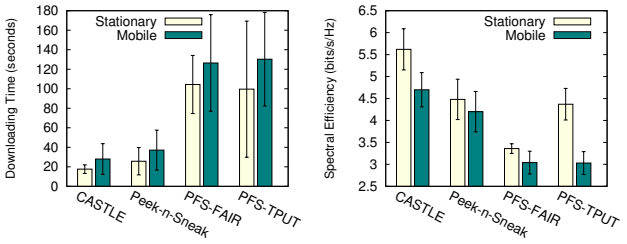


Figure 14: Mobile test scenario with handovers between co-channel neighbor cells.



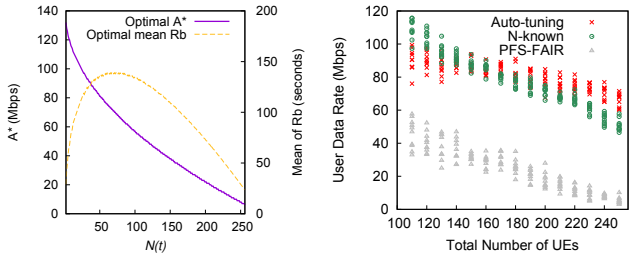
(a) Downloading time **(b) Spectral efficiency**

Figure 15: Stationary vs mobile: CASTLE performs better when UEs are stationary, but still outperforms Peek-n-Sneak, PFS-FAIR, and PFS-TPUT in mobile environments.

eight UEs continuously move and are handed off between the Cells A and B. The mobility is emulated by the signal attenuator, which is programmed to increase or decrease the signals of Cells A and B by 0.5 dBm for every second, in the range of [-50 dBm, -120 dBm]. Since the two stationary UEs are continuously downloading, they cause interference near the edge of Cell B.

Performance comparison: CASTLE heavily relies on UE measurements. Higher mobility makes CASTLE's estimation less accurate, degrading its performance. In Figure 15, we observe that CASTLE still significantly outperforms Peek-n-Sneak, PFS-FAIR, and PFS-TPUT in a mobile environment, although its overall performance is worse than in the stationary experiment. Due to mobility, the average download time of CASTLE increases from 17.6 sec to 28.0 sec, while that of Peek-n-Sneak increases from 25.7 sec to 37.1 sec in Figure 15a. Compared with Peek-n-Sneak, the download time of CASTLE is reduced by 25%. Compared with PFS-FAIR and PFS-TPUT, which now show similar performance, CASTLE's download time is much reduced by 78–79%. Spectral efficiency is also degraded as shown in Figure 15b. The average spectral efficiency of CASTLE decreases from 5.62 bits/s/Hz to 4.70 bits/s/Hz. Compared to Peek-n-Sneak and PFS, CASTLE's spectral efficiency is enhanced by 12% and 55%, respectively.

Implications: Although the performance of CASTLE is degraded in a mobile environment, we found that CASTLE still outperforms Peek-n-Sneak and PFS. Since we assume high mobility (8 UEs move together into another cell every 140 seconds), we can expect better performance in many practical scenarios and/or considerations. Some examples include 1) the UE does not continuously move; 2) DDL_n, the deadline of downloading, is long; 3) there is less interference; and 4) $N(t)$ is small, so the higher throughput threshold will be selected. In the next section, we describe the effect of $N(t)$, since it is a parameter that cannot be calculated or estimated by individual UEs.



(a) Optimal A^* and $\overline{Rb}$ **(b) Average user data rate**

Figure 16: Simulation results with $\phi_n/DDL_n = 0.25$. (a) The throughput threshold decreases as the number of CASTLE users increases. (b) CASTLE *Auto-tuning* outperforms PFS-FAIR, while it is comparable to CASTLE *N-known*.

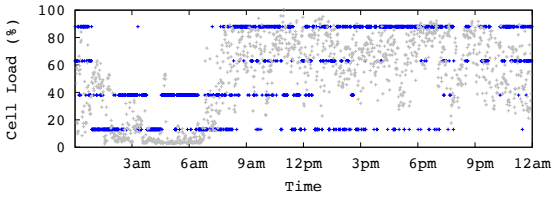
5.4 Auto-tuning of $N(t)$

As explained in §3, $N(t)$ is the number of concurrent UEs running CASTLE at time slot t; however, it cannot be exactly estimated at a single UE. In practice, we suggest an approach to tune $N(t)$ over time t for each download. When a UE starts using CASTLE, it assumes that there is no other CASTLE user, i.e., that $N(t) = 1$. Then $N(t)$ is increased with a rate of $N(t)_{\max}/DDL_n$ where $N(t)_{\max}$ is the maximum value of $N(t)$ that satisfies $A^* > 0$, which can be directly derived from $N(t)_{\max} = (DDL_n \cdot b^2)/(2\phi_n(b-a))$. For example, in Figure 16a, $N(t)_{\max}$ is calculated to be 280 for $\phi_n/DDL_n = 0.25$. Thus, $N(t)$ linearly increases over time, starting at 1, while the threshold A^* decreases. This approach assumes that having longer wait times means that there are likely more CASTLE users, which is similar to CSMA/CD where more collisions indicate more users in the system.

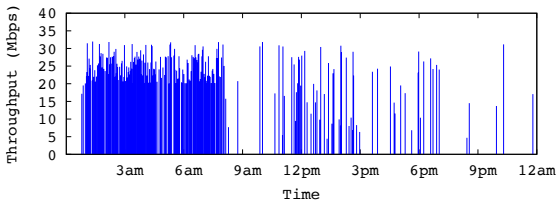
***Auto-tuning* evaluation:** In order to see the impact of this $N(t)$ auto-tuning on CASTLE, we conducted a large scale simulation. We assumed a cell of 1km radius with 110–250 randomly located UEs per cell. Each UE moves at random with speed between 0 and 120 km/h. 100 UEs generate random background traffic, while the remaining 10–150 UEs try to download a 1000 MB file using CASTLE or PFS-FAIR. For PFS-FAIR, we randomly distributed the starting times of the UEs to prevent all UEs from downloading at the same time. For CASTLE, *Auto-tuning* is compared to *N-known*, which assumes that $N(t)$ is known to each UE. Figure 16b shows the average data rate achieved by users through *Auto-tuning*, *N-known*, and PFS-FAIR over ten runs. Each dot represents the average value of a single simulation run. Overall, CASTLE *Auto-tuning* and *N-known* achieve similar average data rates for any number of CASTLE UEs, outperforming PFS-FAIR. Thus, we empirically demonstrate that *Auto-tuning* is a practical way to implement CASTLE.

5.5 Evaluation in a Real LTE Network

We finally evaluate CASTLE in a commercial LTE network. To analyze the scheduling information of a commercial LTE eNodeB, we first implement a LTE downlink control channel decoder by modifying the OWL software [4]. By leveraging this decoder, we report how many RBs are utilized in the commercial LTE cell while we run CASTLE. Figure 17a compares the measured cell load to the estimated load class of CASTLE over one day. There are some



(a) CASTLE's cell load estimation vs. measured cell load



(b) CASTLE file downloads and throughputs

Figure 17: Tested at the edge of a real AT&T cell operating on 739 MHz where the average RSRP was -110 dBm. (a) A blue dot depicts one of four cell load classes estimated by CASTLE, while gray dots are the measured loads. (b) A single vertical line depicts a file transfer done by CASTLE. Our CASTLE application effectively leverages the time when there was not much load in the early morning.

mismatches due to the different measurement points, but CASTLE's estimation is overall accurate. We finally installed a CASTLE application on an AT&T phone that continuously downloaded a file from the Internet on the same day of the week exactly one week later (which had a similar traffic pattern to the cell load training data). Figure 17b shows that the CASTLE application efficiently utilized wireless resources by downloading the file when there was not much load (1AM - 8AM), validating our approach.

6 DISCUSSION

Distributed vs. Centralized scheduling: Our objective is to minimize energy expenditure for each UE, which is equivalent to maximizing average per-UE throughput for active users subject to application-specific constraints as discussed in §3.2. To achieve this objective with a centralized algorithm, the BS should schedule the UEs with maximum achievable throughput in each time slot subject to the same constraints. This, however, requires an extensive amount of control messages to be exchanged between UEs and the BS, which needs further standardization efforts. Our distributed scheduling lets users with high expected throughput participate in channel access with a random backoff, approximately achieving not only maximal per-UE throughput but also fairness by virtue of backoff. The only common information required is an estimate of the number of UEs, which we show the UEs can make in practice.
Accuracy of the ML algorithm: To mitigate computational overhead, CASTLE allows each device to easily estimate the cellular load from a lookup table in the SDK, where the ML algorithm has been trained from multiple device types. However, to achieve higher accuracy (albeit with little marginal improvement over our current results), we can further divide the data for different ML models based on different device types (i.e., vendors and models).

Coexistence with existing mobile devices: A mobile device running CASTLE attempts the backoff operation only when it expects that the estimated throughput is higher than the optimal threshold. This behavior would protect the traffic of existing mobile devices that do not adopt CASTLE by utilizing only unused resources. We can further introduce the concept of maximum transfer time to ensure fairness and avoid starvation of existing devices as in [6]. On the other hand, even in highly loaded situations, the CASTLE devices with a small expected throughput $A(t)$ can get allocated resources from the eNodeB since a shorter deadline DDL leads to a smaller threshold A^* over time.
Use Cases: CASTLE can be easily integrated into existing applications as a separate SDK, though ideally it would be integrated at the platform level to improve both spectral efficiency and battery consumption on smart devices. Below we list some potential use cases of CASTLE.

- **Energy- and spectrum- efficient transmission of delay tolerant traffic.** Frequent firmware and ML model updates on smart devices have a relatively large delay tolerance. Exploiting this tolerance by utilizing unused spectrum can measurably help to support these devices' growing bandwidth needs. Moreover, mobile users' background traffic such as software updates, app downloads, emails, and social media updates are also delay-tolerant, though the degrees of tolerance vary across users, applications, and device types [10]. CASTLE can consider this individual tolerance when running a fully distributed scheduling algorithm on each device.
- **Scheduling for D2D communication.** Device-to-device (D2D) communication is emerging as a key enabler to facilitate the realization of machine-to-machine communications. Even in D2D, both energy efficiency and quality of service (QoS) are severely degraded by the strong intra-cell and inter-cell interference caused by dense deployment and spectrum reuse. CASTLE's inference of the cell load and interference can improve on a centralized interference mitigation method in D2D settings [33].
- **Transport protocols for exploiting extra bandwidth without causing network congestion.** LEDBAT [24] is a transport protocol that uses the extra bandwidth of the network without causing network congestion. LEDBAT is used by delay-tolerant applications and estimated to transport 13-20% of Internet traffic. CASTLE's theoretical framework can be used to implement LEDBAT-like transport protocol for cellular networks by providing an accurate estimation of extra wireless resources.
- **UE-triggered interference cancellation and handover decisions.** Existing approaches to interference cancellation in cellular networks are mostly network-triggered, and the existing handover decisions are mostly taken based on RSRP. More accurate cell load information can improve these decisions. CASTLE's load and interference models in §3 can be used to facilitate UE-triggered interference cancellation and handover algorithms.
- **New types of affordable mobile pricing.** The rapid growth in demand for mobile data forces Internet Service Providers (ISPs) to over-provision network capacity, incurring costs

Category	CASTLE	LoadSense [6]	CQIC [19]	piStream [27]	CLAW [28]
Modification	UE (COTS)	UE (w/ QxDM)	UE & Server	UE (w/ QxDM)	UE (COTS) & Server
Load level	Quaternary	Binary	Capacity estimation	RB utilization	RB utilization
Interference	Inter-cell	×	×	×	×
Scheduling	Distributed (theory-driven)	Distributed (CSMA-like)	×	×	×
API provision	○	×	×	×	×
Measurement period	1 sec	3 sec	RTT	200 ms	RTT
Application	Delay-tolerant	Background	Bulk download	Video streaming	Web

Table 3: Comparison between CASTLE and other UE-side solutions to infer cellular network load.

to extend their infrastructure [10]. ISPs can offer incentives (e.g., discounts or rewards) to users who run CASTLE on their smart devices, as the use of CASTLE on many devices can improve network utilization without hurting existing customers, reducing the need for network over-provisioning.

7 RELATED WORK

Passive load estimation. UE-based load estimation for commercial cellular networks has been proposed by works that collect PHY-layer information from the cellular modem [6, 19, 27, 28]. As a pioneer, LoadSense [6] showed that RSRQ can be used for estimating the cellular load. However, LoadSense was only evaluated on 3G networks, where it had only 75% accuracy on average due to making binary inferences based on a fixed throughput threshold (e.g., 1.5 Mbps in 3G). In CQIC [19], the authors built a UDP-based transport protocol in which the client estimates available capacity based on a CQI-to-rate mapping. Similarly, piStream [27] proposed to monitor more detailed PHY-layer information (i.e., per-subcarrier energy level) to estimate available bandwidth over LTE for a new rate adaptation scheme for mobile video streaming. Recently, Xie et al. [28] proposed a new congestion control algorithm called CLAW that harnesses limited PHY-layer statistics available from LTE phones with an analytical model. These works, however, have not considered inter-cell interference in their estimation models.

Active load estimation. There have been several research works on maximizing cellular link utilization based on observed packet transmissions, e.g., [9, 21, 25, 30–32]. PROTEUS [30] forecasts achievable network performance in real time for an interactive mobile application. Sprout [25] uses packet inter-arrival times to infer cellular link bandwidth and further determines the number of packets that can be transmitted as an end-to-end transport protocol. Verus [32] adapts the sending rate following the delay-to-bandwidth mapping learned during a training phase to react quickly to cellular capacity. LinkForecast [31] takes a ML-based hybrid approach that leverages both upper-layer (e.g., throughput) and lower-layer information (e.g., RSRP/RSRQ) to predict link bandwidth in real time. To mitigate excessive queueing in cellular networks, ExLL [21] adjusts the congestion window of the transport protocol for downlink traffic, while QCUT [9] controls LTE firmware buffer occupancy for uplink traffic.

Coordinated transmission. To improve channel utilization and energy consumption, several works have studied a scheduling-based approach that tries to distribute cellular traffic from multiple clients or applications over time based on channel and load states. Bartendr [23] performs an energy-aware scheduling based on prediction of signal quality, but does not consider the load level. In LoadSense [6], Peek-n-Sneak was presented as a distributed scheduling protocol for coordinated transmission from multiple UEs, which is similar to CSMA for Wi-Fi networks. However, since Peek-n-Sneak uses a simple random backoff based on a relatively long slot duration of 3 seconds, it is likely to have sub-optimal performance compared to CASTLE's optimized backoff strategy and further may deliver unfair service to UEs located in the cell edge, which may not have high throughputs. CoSchd [26] is a mathematical framework that considers both channel and load fluctuations for cellular congestion alleviation. Even though its performance is close to optimality, it requires each BS to update its congestion signal from aggregated load, which is impractical.

Cellular network analysis. Huang et al. [13] discovered that actual bandwidth usage is less than 50% of the available bandwidth due to both application behaviors and TCP parameter settings. LTEye [16] is an open platform that can run on off-the-shelf software radios. It analyzes the LTE radio performance by monitoring the LTE PHY-layer and provides deep insights on these networks. MobileInsight [18] analyzes operational cellular networks on smartphones. Despite its wide applicability, it does not estimate cellular load in real time.

In summary, Table 3 compares UE-side solutions related to cellular load estimation.

8 CONCLUSION

In this paper, we have presented CASTLE, a fully distributed scheduling framework that can jointly optimize the spectral efficiency of cellular networks and the battery consumption of smart devices. For CASTLE's validation, we built a prototype LTE system that provides a full connectivity between application servers and mobile devices and implemented CASTLE as a mobile SDK. Through extensive experiments on our testbed and in AT&T's LTE network, we confirmed that CASTLE outperforms the existing protocol Peek-n-Sneak and centralized PF scheduling, in terms of both spectral efficiency and battery consumption. We expect that both cellular operators and mobile users can benefit from CASTLE by easily deploying its SDK.

ACKNOWLEDGMENTS

We would like to thank the MobiSys reviewers and our shepherd Giovanni Pau for their feedback on earlier versions of this paper. This work was partially supported by the NIST under Grant No. 70NANB17H186, the DARPA under contract No. HR001117C0048, and the NSF under Grants CNS-1525435 and CNS-1738097.

REFERENCES

[1] 3GPP. 2016. LTE: Evolved Universal Terrestrial Radio Access (E-UTRA); Physical layer; Measurements (Release 13). http://www.3gpp.org/dynareport/36214.htm. (2016).

[2] 3GPP. 2016. LTE: Evolved Universal Terrestrial Radio Access (E-UTRA); Physical layer procedures (Release 13). http://www.3gpp.org/dynareport/36213.htm. (2016).

[3] 3GPP RAN WG 3. 2016. 3GPP Standards for the Internet-of-Things. https://goo.gl/DAzMLT. (2016).

[4] Nicola Bui and Joerg Widmer. 2016. OWL: a Reliable Online Watcher for LTE Control Channel Measurements. In *ACM All Things Cellular (MobiCom Workshop)*. ACM, 25–30.

[5] F. Capozzi, G. Piro, L. A. Grieco, G. Boggia, and P. Camarda. 2013. Downlink Packet Scheduling in LTE Cellular Networks: Key Design Issues and a Survey. *IEEE Communications Surveys Tutorials* 15, 2 (2013), 678–700.

[6] Abhijnan Chakraborty, Vishnu Navda, Venkata N. Padmanabhan, and Ramachandran Ramjee. 2013. Coordinating Cellular Background Transfers Using Loadsense. In *Proceedings of ACM MobiCom*. ACM, 63–74.

[7] Cisco. 2017. Visual Networking Index: Global Mobile Data Traffic Forecast Update, 2016-2021 White Paper. https://goo.gl/iUQZmQ. (2017).

[8] Forbes. 2017. 2017 Roundup Of Internet Of Things Forecasts. https://goo.gl/bKLCJB. (2017).

[9] Yihua Guo, Feng Qian, Qi Alfred Chen, Zhuoqing Morley Mao, and Subhabrata Sen. 2016. Understanding On-device Bufferbloat for Cellular Upload. In *Proceedings of ACM IMC*.

[10] Sangtae Ha, Soumya Sen, Carlee Joe-Wong, Youngbin Im, and Mung Chiang. 2012. Tube: time-dependent pricing for mobile data. *ACM SIGCOMM Computer Communication Review* 42, 4 (2012), 247–258.

[11] W. Hu and G. Cao. 2015. Energy-aware video streaming on smartphones. In *2015 IEEE Conference on Computer Communications (INFOCOM)*.

[12] Junxian Huang, Feng Qian, Alexandre Gerber, Z. Morley Mao, Subhabrata Sen, and Oliver Spatscheck. 2012. A Close Examination of Performance and Power Characteristics of 4G LTE Networks. In *Proceedings of ACM MobiSys*.

[13] Junxian Huang, Feng Qian, Yihua Guo, Yuanyuan Zhou, Qiang Xu, Z. Morley Mao, Subhabrata Sen, and Oliver Spatscheck. 2013. An In-depth Study of LTE: Effect of Network Protocol and Application Behavior on Performance. In *Proceedings of ACM SIGCOMM*.

[14] R. Jain, D. Chiu, and W. Hawe. 1984. *A quantitative measure of fairness and discrimination for resource allocation in shared systems.* Tech. Rep. DEC-TR-301. http://www1.cse.wustl.edu/âĹijjain/papers/ftp/fairness.pdf

[15] Juni. 2017. Enterprise Small Cell JL620. http://www.juniglobal.com/product/jl-620fdd-jlt-621tdd/. (2017).

[16] Swarun Kumar, Ezzeldin Hamed, Dina Katabi, and Li Erran Li. 2014. LTE Radio Analytics Made Easy and Accessible. In *Proceedings of ACM SIGCOMM*.

[17] Lee, Jihoon and Lee, Jinsung and Im, Youngbin and Dhawaskar Sathyanarayana, Sandesh and Rahimzadeh, Parisa and Zhang, Xiaoxi and Hollingsworth, Max and Joe-Wong, Carlee and Grunwald, Dirk and Ha, Sangtae. 2019. CASTLE SDK. https://github.com/cu-pscr/CASTLE_LIBRARY.git/. (2019).

[18] Yuanjie Li, Chunyi Peng, Zengwen Yuan, Jiayao Li, Haotian Deng, and Tao Wang. 2016. Mobileinsight: Extracting and Analyzing Cellular Network Information on Smartphones. In *Proceedings of ACM MobiCom*.

[19] Feng Lu, Hao Du, Ankur Jain, Geoffrey M. Voelker, Alex C. Snoeren, and Andreas Terzis. 2015. CQIC: Revisiting Cross-Layer Congestion Control for Cellular Networks. In *Proceedings of ACM HotMobile*. ACM, 45–50.

[20] NextEPC Inc. 2019. Open source implementation of LTE EPC. https://www.nextepc.com/. (2019).

[21] Shinik Park, Jinsung Lee, Junseon Kim, Jihoon Lee, Sangtae Ha, and Kyunghan Lee. 2018. ExLL: An Extremely Low-latency Congestion Control for Mobile Cellular Networks. In *Proceedings of ACM CoNEXT*.

[22] Klaus Ingemann Pedersen, Troels Emil Kolding, Frank Frederiksen, István Zsolt Kovács, Daniela Laselva, and Preben Elgaard Mogensen. 2009. An Overview of Downlink Radio Resource Management for UTRAN Long-term Evolution. *IEEE Comm. Mag.* 47, 7 (July 2009), 86–93.

[23] Aaron Schulman, Vishnu Navda, Ramachandran Ramjee, Neil Spring, Pralhad Deshpande, Calvin Grunewald, Kamal Jain, and Venkata N. Padmanabhan. 2010. Bartendr: A Practical Approach to Energy-aware Cellular Data Scheduling. In *Proceedings of ACM MobiCom*.

[24] S. Shalunov, G. Hazel, J. Iyengar, and M. Kuehlewind. 2012. *Low Extra Delay Background Transport (LEDBAT)*. RFC 6817. RFC Editor. http://www.rfc-editor.org/rfc/rfc6817.txt http://www.rfc-editor.org/rfc/rfc6817.txt.

[25] Keith Winstein, Anirudh Sivaraman, and Hari Balakrishnan. 2013. Stochastic Forecasts Achieve High Throughput and Low Delay over Cellular Networks. In *Proceedings of USENIX NSDI*.

[26] H. Wu, X. Lin, X. Liu, K. Tan, and Y. Zhang. 2016. CoSchd: Coordinated Scheduling With Channel and Load Awareness for Alleviating Cellular Congestion. *IEEE/ACM Transactions on Networking* 24, 5 (October 2016), 2579–2592.

[27] Xiufeng Xie, Xinyu Zhang, Swarun Kumar, and Li Erran Li. 2015. piStream: Physical Layer Informed Adaptive Video Streaming over LTE. In *Proceedings of ACM MobiCom*.

[28] Xiufeng Xie, Xinyu Zhang, and Shilin Zhu. 2017. Accelerating Mobile Web Loading Using Cellular Link Information. In *Proceedings of ACM MobiSys*.

[29] F. Xu, Y. Li, H. Wang, P. Zhang, and D. Jin. 2017. Understanding Mobile Traffic Patterns of Large Scale Cellular Towers in Urban Environment. *IEEE/ACM Transactions on Networking* 25, 2 (April 2017), 1147–1161.

[30] Qiang Xu, Sanjeev Mehrotra, Zhuoqing Mao, and Jin Li. 2013. PROTEUS: Network Performance Forecast for Real-time, Interactive Mobile Applications. In *Proceedings of ACM MobiSys*.

[31] C. Yue, R. Jin, K. Suh, Y. Qin, B. Wang, and W. Wei. 2017. LinkForecast: Cellular Link Bandwidth Prediction in LTE Networks. *IEEE Transactions on Mobile Computing* Preprint (2017).

[32] Yasir Zaki, Thomas Pötsch, Jay Chen, Lakshminarayanan Subramanian, and Carmelita Görg. 2015. Adaptive Congestion Control for Unpredictable Cellular Networks. In *Proceedings of ACM SIGCOMM*.

[33] Z. Zhou, M. Dong, K. Ota, G. Wang, and L. T. Yang. 2016. Energy-Efficient Resource Allocation for D2D Communications Underlaying Cloud-RAN-Based LTE-A Networks. *IEEE Internet of Things* 3, 3 (June 2016), 428–438.