# CoAST: Collaborative Application-Aware Scheduling of Last-Mile Cellular Traffic

Cong Shi*, Kaustubh Joshi†, Rajesh K. Panta†, Mostafa H. Ammar*, Ellen W. Zegura*
*School of Computer Science
Georgia Institute of Technology, Atlanta, GA
{cshi7, ammar, ewz}@cc.gatech.edu
†AT&T Labs - Research, Bedminster, NJ
{kaustubh, rpanta}@research.att.com

## ABSTRACT

The explosive growth of mobile data traffic poses severe pressure on cellular providers to better manage their finite spectrum. Proposed solutions such as congestion-pricing exist, but they degrade users' ability to use the network when they want. In this paper, we propose a fundamentally different approach - rather than reducing the aggregate busy hour traffic, we seek to smooth the peaks that cause congestion. Our approach is based on two key insights obtained from traffic traces of a large cellular provider. First, mobile traffic demonstrates high short-term variation so that delaying traffic for very short periods of time can significantly reduce peaks. Second, by making collaborative decisions on which traffic gets delayed and by how much across all users of a cell, the delays need not result in any degradation of user experience. We design a system, CoAST, to implement this approach using three key mechanisms: a protocol to allow mobile applications and providers to exchange traffic information, an incentive mechanism to incentivize mobile applications to collaboratively delay traffic at the right time, and mechanisms to delay application traffic. We provide extensive evaluations that show that CoAST reduces traffic peaks by up to 50% even for applications that are not thought to be delay-tolerant, e.g., streaming and web browsing, but which together account for 70% of all cellular traffic.

## Categories and Subject Descriptors

C.2.3 [**Computer-Communication Networks** ]: Network Operations —*Network Management*

## Keywords

Cellular Networks; Peak Throughput; Delay Tolerance; Scheduling

## 1. INTRODUCTION

The rapid proliferation of smartphones, tablets, and mobile applications has led to a tremendous increase in mobile data traffic in the last few years. For example, the mobile data traffic on major US based mobile carriers has increased by more than 20,000% in five years [7]. Furthermore, according to forecasts by major equipment manufacturers, this trend is likely to continue in future with 78% compound annual growth rate [1]. In contrast, the capacity of cellular networks, especially the wireless spectrum, has not increased proportionally. The efficient management of mobile traffic is, therefore, critical for cellular network operators.

Various solutions have been proposed to manage the mismatch between the ever-increasing traffic demand and finite wireless spectrum. These solutions can be broadly classified into two categories—adding more network resources to increase the overall capacity (i.e. increasing supply), or managing user demands and behavior to reduce the load on the network (i.e. controlling demand). Examples of the first category include the use of small cells for augmenting the capacity of traditional macro cells, adding WiFi hotspots to offload cellular traffic to WiFi, using portable base stations (e.g. Cells On Wheels or COWs) to meet high traffic demands in event venues where large numbers of users gather for some time periods, etc. Examples of the second category include congestion pricing, off-peak delivery, network-aware throttling, etc. These approaches reduce the aggregate traffic in busy periods by either shifting the parts of traffic that can tolerate some delay to off-peak hours (e.g. backup, synchronization, cloud offload, etc.) [12], or causing the user to use the network less frequently.

These existing approaches have some fundamental limitations. Shifting traffic to off-peak hours can cause degradation of quality of service experienced by the end users. The vast majority of mobile data traffic, including video streaming and mobile web browsing, cannot be shifted to off-peak hours because of latency requirements. Although additional network resources increase the overall capacity, they also incur significant costs. Furthermore, solutions like small cells can be deployed only gradually because of the detail radio engineering trials required for the correct positioning and deployment of such infrastructure.

In this paper, we take a fundamentally different approach to tackle this problem—delaying mobile traffic like video streaming and mobile web browsing that are not traditionally thought to be delay tolerant. This is based on two key insights derived from mobile traffic traces of a large US cellular provider. First, we observe that the mobile data traffic exhibits high burstiness over small time scales (tens of seconds). Thus, to ensure adequate quality of service at all times, it is important to reduce the instantaneous peak traffic, not just the aggregate traffic. Second, even applications like video streaming and mobile web browsing can, in fact, tolerate small delays. For example, a video streaming client can tolerate delays of tens of seconds as long as its playback buffer is not empty. Mobile web browsers can delay downloading the contents that are not currently displayed on the screen. These two insights suggest that if the right user traffic (from the set of all current user traffic in the

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.
*MobiSys'14,* June 16–19, 2014, Bretton Woods, New Hampshire, USA.
Copyright 2014 ACM 978-1-4503-2793-0/14/06 ...$15.00.
http://dx.doi.org/10.1145/2594368.2594385 .

cell) is delayed at the right time for the right time duration, it is possible to reduce the peak traffic in a cell without affecting the user experience on any mobile device. This requires both device-level information (e.g. tolerable delay values at the given time instant) and cell-level information (e.g. the total traffic demand in the cell at the given time instant). Thus, an efficient interaction mechanism between mobile devices and cellular infrastructure is necessary to enable collaboration between them to make proper decisions about delaying the user traffic.

Using these insights, we present the design, implementation, and extensive evaluation of a novel system called CoAST (**Co**llaborative **A**pplication-Aware **S**cheduling of Last Mile Cellular **T**raffic). CoAST provides an interface to enable an efficient collaboration between mobile devices and the network element to which they are connected (e.g. a base station). The interface is simple and flexible, allowing dynamic policies and protocols to be built on top of it, according to the requirements and capabilities of individual mobile applications. CoAST also provides an incentive mechanism for mobile applications to delay their traffic and the actual mechanisms to delay the application traffic.

In addition to benefits to the mobile network operator, CoAST also improves the application performance for the end user. By delaying traffic of users with enough playback buffer contents, CoAST can aggressively fill the starving buffers of other users, thereby reducing their buffering delays.

In summary, CoAST makes following novel contributions:

1) Rather than reducing the aggregate busy hour traffic, CoAST reduces the instantaneous peak load in a cell, without compromising the quality of service experienced by the end users.

2) CoAST can handle traffic which is not traditionally thought to be delay tolerant.

3) CoAST provides a simple and flexible interface for mobile devices and the cellular network to exchange various information to enable them to make proper decisions about delaying user traffic.

We implement a prototype of CoAST on the Android platform for two sample application categories, streaming (e.g., YouTube) and web browsing, which are the top two generators of cellular network traffic, accounting for nearly 70% of global cellular traffic [25]. We evaluate our implementation on a per-cell basis using emulation based on real YouTube and web browsing traces obtained from a major US cellular provider. Our results show that CoAST reduces traffic peaks by an average of 30-50%, or conversely, increases the capacity of a cell by 20% without compromising the quality of the end user experience. In fact, we show that CoAST achieves a better quality of service in terms of reduced buffering delay compared to the cell that does not use CoAST. Our experiments also show that the control plane overhead introduced by CoAST is negligible.

The rest of the paper is organized as follows. In Section 2, we describe the background of this work and present an overview of CoAST design. We motivate the CoAST design with an analysis on cellular traffic in Section 3. The design details of CoAST is described in Section 4. The deployment and implementation of CoAST system is discussed in Section 5. The system evaluation of CoAST prototype and the trace-driven evaluation of CoAST are presented in Section 6 and 7, respectively. We discuss the related issues in Section 8. Section 9 summarizes the related work. We conclude this paper and discuss future work in Section 10.

# 2. BACKGROUND AND DESIGN OVERVIEW

In this section, we describe the background of this work and present a high-level overview of CoAST design.

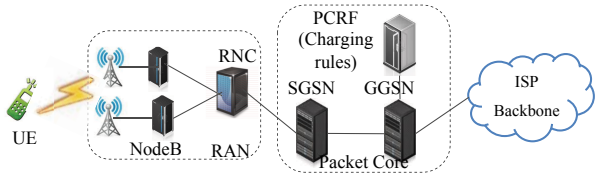



**Figure 1: Architecture of UMTS data network.**

## 2.1 Background of Cellular Networks

In this subsection, we first describe the basics of cellular architecture and then provide information about the data set used in our evaluations.

Figure 1 shows the key components of a typical UMTS data network. It consists of 2 major components: the Radio Access Network (RAN) and the Core Network (CN) (or Packet Core). The mobile device, called User Equipment (UE) in UMTS terminology, is connected to one or several cell sectors in RAN. A physical base station (called NodeB in 3G and eNodeB in LTE) can have multiple cell sectors, which provide radio resources to UEs for wireless communications. Cellular data traffic from several NodeBs are then passed to the Radio Network Controller (RNC), which manages handovers, and scheduling of wireless resources among the NodeBs under its control. The RNCs connect to Serving GPRS Support Nodes (SGSNs) at the core network. The SGSNs are connected to the external networks, such as the Internet, via Gateway GPRS Support Nodes (GGSNs). When a UE connects to the network, it establishes a Packet Data Protocol (PDP) context which facilitates tunneling of IP traffic from the UE to the peering GGSN using GPRS Tunneling Protocol (GTP) (see [14] for details of the UMTS network).

We evaluate CoAST using real-world cellular traffic data from a tier-1 cellular network carrier. All device and user identifiers are anonymized for our analysis. The first data set is collected from the link between SGSN and GGSN in the core network. It contains information about IP flows carried in PDP contexts for a 3% random sample of devices collected every minute, e.g. start and end time stamps, per flow traffic volume, application identifier, etc. This data set is used to collect information about the characteristic of video streaming traffic. To gain more fine grained information about the actual traffic volume in a given cell sector, we use another dataset collected every 2 seconds at RNCs in the RAN network.

## 2.2 Design Overview

CoAST aims to reduce the peak-to-average ratio of the cellular last-hop traffic, and consequently improve application performance for the end user. It is not designed for persistently congested networks whose average traffic is close to the capacity. Such networks need to be upgraded to increase the capacity. We focus primarily on *downstream* as the upstream traffic intensity is not as significant. (Note that CoAST can apply equally to upstream traffic as well. But we do not claim it is the contribution of this paper.) We accomplish this by delaying downstream traffic at times when the link is experiencing heavy load. In order to avoid degrading the quality of service experienced by the end user, the acceptable delays may range from a few to tens of seconds depending on the applications. As will be shown in the next section such relatively short traffic delays are enough to produce the desired effect of reducing traffic peaks.

One key insight of CoAST is that only mobile applications know the delay constraints of their traffic while the eNodeB has the aggregated traffic information. Thus, solutions that rely solely on the eNodeB or on the mobile devices do not have enough information

to simultaneously achieve both high utilization and good quality of service. Collaboration between the eNodeB and mobile devices is required to determine if and by how much the downstream data should be delayed. The scheme, described in detail in Section 4, enables the users to optimize their download rates while minimizing the cost of data download and maximizing the quality of service.

It is important to note that the pricing incentive used in our scheme is, strictly speaking, internal to the system and is used to facilitate decision making regarding whether to delay a certain chunk of the download traffic. It is not meant to be exposed directly and as-is to the user, nor is it meant to translate directly into billing. It will be important, however, to incentivize users to deploy this system in their devices. So we expect that there will be some correlation between billing and pricing practices of operators to be influenced by the usage of this system. However, the exact approach here is beyond the scope of our technical discussion.

Although a wide user participation increases the effectiveness of CoAST, it is not necessary that all users in a cell deploy CoAST—only enough number of UEs that can make a difference in traffic peaks is needed. Also note that users who do not deploy the system will not necessarily have any advantage over those who use the system since the user experience is preserved for those using it. So there is no individual incentive to "cheat" the system from this perspective. On the contrary, users will be interested to participate if operator pricing does somehow reflect the usage of the system. CoAST also monitors the behaviors of participating devices to identify potential cheating, which will be discussed in Section 8.

# 3. FEASIBILITY AND BENEFITS OF DELAYING MOBILE TRAFFIC

In this section we present an analysis of the real-world traffic traces of a large US cellular carrier to provide motivation about the feasibility and potential benefits of delaying mobile traffic for short time durations.

## 3.1 Short-Term Burstiness of Mobile Traffic

First, we analyze the traffic distribution of a large number of cells (13522 NodeBs) using a large dataset of cellular traffic collected at several RNCs to demonstrate that the mobile data traffic of a typical cell demonstrates high variations over short time scales (e.g. 30 seconds). This dataset provides per cell as well as per UE cellular data records. We focus on the downlink traffic because it is significantly larger than the uplink traffic in cellular networks [9, 18].

Figure 2 plots the downlink throughput along with its 20 second moving average in a typical cell in one day. The traffic is clearly very bursty with large short-term variations. To ensure adequate quality of service at all times, the cellular network provider needs to over-provision the network resources based on the peak traffic demand. Therefore, the burstiness of mobile traffic makes the resources underutilized during most periods of time.

A heuristic idea to better utilize the cellular resources is to delay a portion of mobile traffic for a short period of time to reduce the peak throughput over time. As a simple approximation, we use the moving average in a short period of time to demonstrate the potential benefits of delaying mobile traffic. Figure 2 shows that the peak value of the 20 second moving average is only about 60% of the original peak throughput. This implies that delaying a portion of mobile traffic by 20 seconds or less can result in a significant reduction in the peak throughput demand in the cell. It will lead to two major benefits. First, the reduction in the peak throughput allows the network to support more users and mobile traffic without upgrading the infrastructure. In some cases, it also means that the cellular network may be able to support better quality services. For example, it may be feasible to support a better quality video streaming (say HD video with better resolution) if there is sufficient reduction in the overall load on the cell. Second, this also reduces congestion and helps improve the performance of mobile applications that are sensitive to delays (e.g., VoIP).

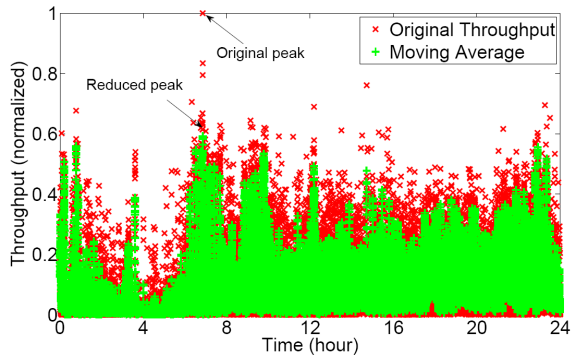



**Figure 2: The downlink traffic and its 20 second moving average in a cell in one typical day. The throughput is normalized with the maximal capacity used. The difference between the original traffic and its moving average demonstrates its high short-term variations.**

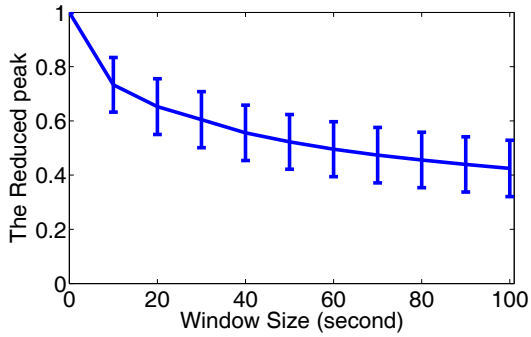



**Figure 3: The reduction of the peak downlink throughput for moving averages computed over different time periods. The average and standard deviation are plotted.**

To further quantify the relationship between the extra delay and the reduction on the peak throughput, we compare the reduced peak throughput achieved by moving averages computed over different time intervals for the top 200 heavy-loaded cells. The results are plotted in Figure 3. When the window size increases from 1s to 30s, the reduced peak throughput quickly decreases from 100% to 60.5% of the original peak on average. As the window size further increases to 100s, the peak is gradually reduced to 42.5% of the original peak. This figure implies that most benefits in terms of reduction of peak throughput can be obtained by delaying traffic for short time durations (e.g., 30 seconds), with diminishing returns for larger delay intervals.

## 3.2 Delay Tolerance of Mobile Applications

The real-world cell traffic traces indicate that delaying mobile traffic for a few seconds can reduce the peak load in the cell significantly. The next natural question is: Can real-world mobile applications tolerate such delays without affecting the quality of service

experienced by the end-users? To investigate this, we consider following major traffic classes:

### 3.2.1 Streaming

Streaming applications (e.g., video streaming, audio streaming) account for around 34% of the total mobile traffic [25]. As they usually buffer some data, they can tolerate small delays which are equivalent to the current buffer occupancy of their playback buffer.

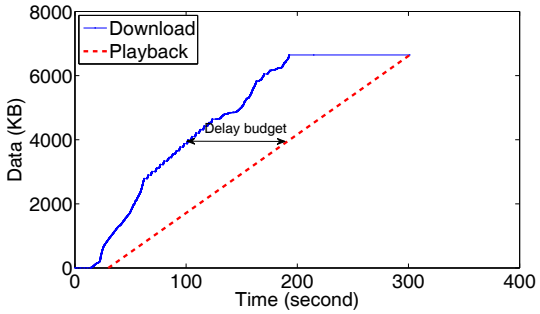



**Figure 4: The download and playback progress of a Youtube video on an Android smartphone using a cellular network. The difference between download and playback represents the delay it can tolerate.**

To investigate delay tolerance of streaming applications over cellular networks, we play a Youtube video on an Android smartphone and record the cellular traffic using Wireshark. Figure 4 compares the number of downloaded bytes and the actual playback progress during the experiment. Initially, the Youtube client aggressively buffers the video contents. But once the client has sufficient contents to play for some time, it slows the download to avoid downloading unnecessary contents in case the user does not watch the complete video. The difference between the actual download and the playback progress at any given time is the amount of delay the Youtube client can tolerate at that time without affecting the user experience (i.e., pausing the video).

We also see from Figure 4 that depending on the size of the buffered data, the tolerable delay varies from a few seconds to more than one hundred seconds in this example. In addition, user operations like "back" and "forward" will also impact the delay that the video client can tolerate at the specific time.

To accurately predict the future traffic and delay that the video client can tolerate, video size, bitrate, buffered data, and playback progress are required. Specifically, we can predict the amount of data to download based on the video size and buffered data. Meanwhile, the bitrate, buffered data and playback progress can be used to estimate the delay that it can tolerate. Fortunately, all of them are available to the video client. Video size and bitrate can be obtained from the video metadata, while buffered data and playback progress are internal states of the video client. In contrast, delaying the streaming traffic arbitrarily, say from the network side without taking real time input from the application, may affect the user experience.

### 3.2.2 Web Browsing

Web browsing applications, which are also one of the top generators of cellular mobile traffic [25], are generally not regarded as being delay-tolerant. Due to their small size, mobile devices like smartphones and tablets can display only a small portion of a webpage (e.g., texts, images, and other multimedia contents) at any given time. Thus when a user browses a web page, only contents that are shown on the screen need to be downloaded immediately, while off-screen contents can be downloaded a little bit later without impacting the user experience. In other words, off-screen contents can be treated as being delay-tolerant. In fact, some websites (e.g., Huffington Post [2]) already support progressive download and display of the web pages. When web pages contain many multimedia contents (e.g., images), the amount of traffic that can be delayed will be significant.

To identify the delay tolerance of web browsing, we analyze the on-screen and off-screen contents of the top 500 Alexa websites [6] in various categories. We only treat off-screen images as off-screen contents and the rest components as on-screen contents. We use PhantomJS [4] to download and render the web pages. For every website, we only download and analyze its home page. We set the user agent of all HTTP requests to that of the Android web browser and let those websites decide whether to return the mobile version or the full version. The screen size is set to 480×800, a typical setting for smartphones. For each web page, we identify all the off-screen images and treat them as off-screen content, while all other contents are treated as on-screen content.

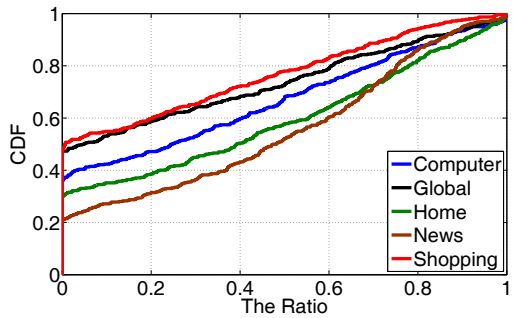



**Figure 5: The ratio of off-screen content to the total content on the homepages of the top 500 websites. We also select 4 categories and plot the top 500 websites in these categories.**

Figure 5 shows the ratio of the size of the off-screen contents to that of the total contents on the homepage of the top 500 Alexa websites [6]. The category "Global" represents the overall top 500 websites. We also select and plot 4 categories among 16 special categories listed by Alexa. The distributions of other categories are between that of "News" and that of "Shopping". Generally a significant portion of the web content is off-screen and can tolerate short extra delays. For categories like "News", more than 50% of those websites have more than 50% of the contents that can tolerate extra delay. In addition, since the homepages of the websites that we analyzed usually contain less content than other pages, our estimation of the potential benefits as shown in Figure 5 is likely very conservative.

Like streaming applications, we also notice that only the web browser knows which images are off-screen and can tolerate short delay, especially when the user scrolls the web page. Specifically, by parsing the HTML file, the web browser can identify the images that are not shown on the screen. In addition, using the height and width attributes of the corresponding "img" tags, it can estimate their sizes.

## 4. COAST DESIGN

CoAST enables collaboration among mobile devices for scheduling their mobile traffic, when necessary and feasible, to reduce the peak traffic load on a cell. The basic idea is to use dynamic pricing to motivate the mobile applications to proactively shift their traffic in small time scales (up to 30 seconds) while still satisfying

their delay constraints. To realize this idea, CoAST uses three major mechanisms: a protocol to allow mobile applications and their associated cell to exchange traffic information, an incentive mechanism to incentivize mobile applications to collaboratively delay traffic at the right time for the right time duration, and a mechanism to enable applications to delay their traffic. The first two mechanisms are incorporated into the *control plane* of CoAST, while the last one is realized in its *data plane*. However, it should be noted that all CoAST mechanisms are data plane functions from 3GPP protocol perspective, i.e. CoAST does not change the 3GPP protocol itself.

## 4.1 Design Principles

**Minimal modification to mobile applications:** For ease of deployment, CoAST requires no modification on the server side, but only small changes on the client side because only mobile applications know the delay constraints of their traffic. CoAST modifies the underlying socket API implementation to allow client applications to specify the tolerable delay information via these socket calls in a transparent manner. The actual value of the tolerable delay at a given time instant depends on the application itself. In this paper, we describe how playback buffer size can be used to figure out the value of tolerable delay for video streaming applications. For other applications, mobile developers may use existing instrumentation systems [13] to determine the tolerable delay, thereby reducing the development efforts.

**Privacy preservation:** To reduce the peak load on the cell, CoAST relies on collaboration and sharing of traffic information among all mobile devices in that cell. A malicious mobile device may participate CoAST and collect the traffic information of other mobile devices to infer their application usage. To prevent privacy leak under such attack, the control plane is divided into UE proxies on mobile devices and a market proxy on the eNodeB. Each mobile device shares its aggregated traffic demand only with the market proxy and obtains the prices for downloading data only from the market proxy, avoiding the direct sharing of traffic demands among participating devices. With this design, it will be impossible for malicious mobile device to obtain accurate traffic information of mobile applications running on other mobile devices.

**Control of demand through pricing:** To motivate mobile applications to delay their traffic when necessary, CoAST uses dynamic "prices" to charge mobile traffic at different time instants. The prices used by CoAST can be $ per bit as used in [12]. More generally, it can also be treated as the discount ratio on the accounted traffic. For example, when the price is 0.8, 1 Mb mobile traffic can be accounted as 0.8 Mb. The latter case is compatible with the usage-based pricing model used by most cellular providers nowadays. In this paper, we don't specify the pricing model used by the cellular providers. We treat the price as the ratio of accounted traffic to the transferred traffic.

**Tackling system abuse:** CoAST requires UEs to report traffic demands to the market proxy, which sets the prices based on demand information from all UEs in the cell. Malicious UEs may attempt to report fake information to abuse the system. To prevent such cheating behavior, CoAST records and compares the reported traffic demands and the real traffic to identify suspicious behavior (please see Section 8 for a detail discussion).

## 4.2 Overview of CoAST Operation

Figure 6 provides a high-level overview of the CoAST architecture. It consists of two major components: a *market proxy* that resides on a cell-level network element like eNodeB and a *UE proxy* on each mobile device. The market proxy collects the traffic demands for some future time window from all mobile devices in a cell. These are used by the market proxy as parameters of an optimization problem (eOpt) which determines new future prices for each UE (for the next time window) with the goal of minimizing the traffic peak in the cell. The traffic demands and prices may also be reported to the LTE network for accounting and billing purpose. On the user side, applications determine their traffic demands in a manner that satisfies their delay constraints and sends this information to its UE proxy through the user library. The UE proxy uses these demand information as inputs to an optimization problem (uOpt) that attempts to minimize the cost of satisfying these demands based on prices obtained from the market proxy. CoAST also includes a mechanism by which applications can control their traffic according to the outputs of uOpt.

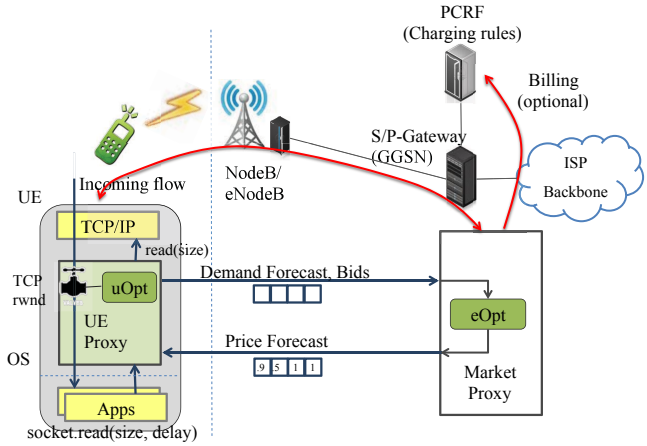



**Figure 6: CoAST Architecture.**

## 4.3 Control Plane

CoAST ensures that the traffic demand does not exceed the network capacity at any time without affecting the user experience of mobile applications. Let's denote the capacity of a cell sector by $c$, the throughput at time $t$ of the $i^{\text{th}}$ flow under the control of CoAST by $th_i(t)$, and the total throughput of all other flows by $th_b(t)$. CoAST's goal can be expressed as optimizing an objective function of the difference between the throughput and capacity over time, as follows:

$$\min \ \Phi(\{\sum_{i=1}^{n} th_i(t) + th_b(t) - c\}) \tag{1}$$

where $\Phi$ can be any meaningful utility function. For simplicity of description, we omit the constraints in Eqn 1 and will present them in Eqn 4.

One design challenge is the fact that CoAST has neither control nor accurate information about the background traffic in the cell. By background traffic, we mean all mobile traffic generated by applications that do not use CoAST. Given the burstiness of mobile traffic as shown in Figure 2, it's also hard to accurately predict the background traffic using historical information. To solve this problem, CoAST optimizes the maximal throughput of the flows under its control over time, as follows:

$$\min \ \max_t \sum_{i=1}^{n} th_i(t) \tag{2}$$

Therefore, no information of the background traffic is required. Let $th_b^* = \max_t th_b(t)$. A pleasant property of this objective function is that it is equivalent to minimizing $\sum_{i=1}^{n} th_i(t) + th_b^*$, which

will be proved in Section 4.5. When enough numbers of UEs use CoAST, the peak throughput of total traffic will be minimized.

A centralized solution to Formula 2 would require the market proxy and the UE proxies to share information of all flows, resulting in a lot of control overhead. Moreover, it violates our privacy preservation design goal. Thus, a distributed control protocol that exchanges limited information is required. To solve this problem, we use the dual decomposition method [20] to decompose the original problem into a master problem and multiple independent sub-problems. The master problem (eOpt) is solved by the market proxy, while the independent sub-problems (uOpt) are solved by the UE proxies. We describe this control protocol next.

At a high level, the control plane functions as follows: The market proxy periodically computes the projected prices for mobile traffic for some time window in the future. These prices are calculated based on the traffic demand collected from all connected UEs. The prices are broadcasted to all connected UEs. Each UE proxy uses the price information from the market proxy and the demand information from its mobile applications to schedule the mobile traffic such that the demand of each application is satisfied and the overall cost is minimized. The UE proxy then sends back the traffic demands that it calculates for some future time window to the market proxy, and the process is repeated.

### 4.3.1 Market Proxy Operations

The market proxy operates in time slots with length $\tau$ (e.g., 1 second). The time slot that time $t$ belongs to is denoted as $\lfloor t \rfloor_\tau = \lfloor \frac{t-t_0}{\tau} \rfloor$, where $t_0$ is the start time. Let $p(\lfloor t \rfloor_\tau)$ represent the price for the downlink traffic through a cell sector during time slot $\lfloor t \rfloor_\tau$. Let $th_e(\lfloor t \rfloor_\tau)$ represent the downlink traffic demand of UE $e$ during time slot $\lfloor t \rfloor_\tau$.

In every $\delta$ seconds (e.g., 0.1), the market proxy receives a vector of traffic demand for the next $\kappa$ (e.g., 30) time slots, i.e., $\overline{th_e} = \{th_e(\lfloor t \rfloor_\tau), th_e(\lfloor t \rfloor_\tau + 1), \ldots, th_e(\lfloor t \rfloor_\tau + \kappa)\}$, where $t$ is the current time. Then it updates the price vector , i.e., $\overline{p} = \{p(\lfloor t \rfloor_\tau), \ldots, p(\lfloor t \rfloor_\tau + \kappa)\}$, as follows:

$$\overline{p'} = [\overline{p} + \beta(\sum_e \overline{th_e} - \alpha)]_H \tag{3}$$

where $\alpha = \max_t \sum_e th_e(t)$, $[\overline{p}]_H$ represents the projection of $\overline{p}$ onto the hyperplane $H = \{p(t) | \sum_t p(t) = 1, p(t) \geq 0\}$, and $\beta$ is the step length. The value of $\beta$ is set to ensure that $\forall t, |p'(t) - p(t)| < p(t)/10$ in our implementation.

The market proxy then broadcasts the prices to all UEs in the cell.

### 4.3.2 UE Proxy Operations

The UE proxy collects the information about traffic demand per slot ($\tau$) for $\kappa$ slots into the future from all mobile applications and periodically receives the future price information from the market proxy. It generates the future traffic demand, $\overline{th_e}$, and sends it to the market proxy.

Each mobile application reports its delay constraints for the next $\kappa$ time slots to its UE proxy. Let's use $\langle D, t \rangle$, i.e., a tuple of the amount of data $D$ and its deadline $t$, to express the delay constraint that data $D$ should be downloaded before deadline $t$. Therefore, for a specific flow $i$, its delay constraints can be expressed as a set of tuples, i.e., $\{\langle D_{i,0}, t_{i,0} \rangle, \langle D_{i,1}, t_{i,1} \rangle, \ldots, \langle D_{i,n}, t_{i,n} \rangle\}$, where $t_{i,0} < t_{i,1} < \cdots < t_{i,n}$. Let's define function $d_i(t) = \sum_{j=0}^{T} D_{i,j}$, where $t_{i,T} \leq t \leq t_{i,T+1}$. It represents the amount of data required to be transferred before time $t$.

When a UE proxy receives the price information from the market proxy, it tries to minimize the cost of transferring data under the constraint that the delay constraints of all flows are satisfied. Specifically, at time $t^*$ the UE proxy solves the following optimization function:

$$\begin{aligned} \min \quad & \sum_i \sum_{t \in T} th_i(t) \times p(t) \\ \text{s.t.} \quad & \sum_i th_i(t) \leq b_d, \forall t \in T \\ & \sum_i \sum_{t \leq t'} th_i(t) \times \tau \geq d_i(t), \forall t' \in T \end{aligned} \tag{4}$$

where $th_i(t)$ is the throughput of flow $i$ at time $t$, $b_d$ is the bandwidth, and $T = \{\lfloor t^* \rfloor_\tau, \ldots, \lfloor t^* \rfloor_\tau + \kappa\}$.

By solving the above optimization function we obtain the desired throughput of all downlink flows over time. $th_i(\lfloor t^* \rfloor_\tau)$ corresponds to the bandwidth allocated to flow $i$ in the current time slot. The UE proxy will send this value back to the mobile applications to control their traffic in the data plane accordingly as described in Section 4.4. It should be noted that those constraints may not be satisfied, i.e., the available bandwidth may be smaller than the traffic demand. Under such scenario, CoAST will allow mobile applications to transfer data as fast as possible and let users decide if they want to stop some applications.

The UE proxy will send $\{th_e(t) | th_e(t) = \sum_i th_i(t), t \in T\}$ to the market proxy. The market proxy collects such traffic demands from all UEs in the cell and updates the prices in the next round.

## 4.4 Data Plane

The primary functionality of the CoAST data plane is to control the downlink traffic based on the throughput cap assigned by the control plane. As most of the mobile applications we are considering use TCP and we don't want to modify the server, we focus on controlling the TCP traffic from the receiver side. It is also important to note that while the control plane operates with a window of projected demands and prices, the data plane only controls the traffic for the current slot.

In TCP the amount of data that the sender can send within an RTT is limited by $\min\{cwnd, rwnd\}$ where $cwnd$ is the congestion window size, and $rwnd$ is the receiver's window size advertised in the acknowledgement packets. When cwnd is larger than rwnd, rwnd will determine the throughput of a TCP flow. To control the downlink traffic from the receiver side, we set an upper-bound on rwnd as:

$$\text{DL_CAP} = \max\{\text{throughput} \times \text{RTT}, \text{MSS}\} \tag{5}$$

where $throughput$ is the target throughput assigned by the control plane. When throughput is very small, DL_CA$P$ is set to MSS to avoid totally blocking the flow. In our implementation, we add a new socket option, DL_CAP, to allow applications to dynamically specify the upper-bound on the advertised receiver's window size at runtime.

It's noteworthy that the receiver will obtain the required downlink throughput after one RTT since the sender receives the new advertised rwnd after RTT/2 and then spends RTT/2 to deliver the new packets to the receiver. When RTT is large (e.g., 1 second), the receiver should use the estimated future throughput to set DL_CAP.

## 4.5 CoAST Performance Guarantee

The primary goal of CoAST is to schedule the last-mile traffic to ensure that the total traffic demand does not exceed the network capacity. For scalability, CoAST is designed as a distributed system that only schedules the traffic under its control. A natural question

is whether the CoAST design meets its goal. Here we present a theoretical analysis to answer this question.

CoAST uses an iterative control protocol between a market proxy and a set of UE proxies to schedule the traffic. This control protocol allows CoAST to minimize the maximal throughput of flows under its control over time. Formally, we have the following theorem:

THEOREM 1. *With the interaction interval, $\delta$, approaching 0, CoAST approaches the optimization goal defined in Formula 2.*

Its proof can be found in Proof 1 of the Appendix. As CoAST minimizes the maximal throughput of flows under its control, it also reduces the maximal throughput of all flows including those background traffic. Formally, we have the following theorem:

THEOREM 2. *Assume $\sum_{i=1}^{n} th_i(t)$ and $th_b(t)$ are independent random variables. With $t \to +\infty$, the expected value of the peak throughput obtained by using CoAST approaches $\max_t \sum_{i=1}^{n} th_i(t) + \max_t th_b(t)$.*

The proof can be found in Proof 2 of the Appendix. Thus, when the number of UEs that uses CoAST is large enough, CoAST can significantly reduce the overall peak traffic.

# 5. DEPLOYMENT AND IMPLEMENTATION

In this section, we discuss various ways in which CoAST can be deployed on a real network and describe our prototype implementation.

## 5.1 Deployment

CoAST proposes two new functions to be added to the cellular network: a UE proxy for each mobile device and a market proxy for each cell sector. The UE proxy interacts with mobile applications running on the UE to collect their delay constraints and allocates bandwidth to them. The market proxy aggregates demand information from all the UEs in a cell and sets the prices. UE and market proxies communicate with each other to exchange demand and pricing information. The network elements on which these functions are deployed determines both the information available to CoAST and the changes needed to the network. We consider three deployment options and their merits.

**Clean Slate:** The market proxy for a cell and the UE proxies for all UEs in the cell are hosted in the cell's eNodeB. Mobile applications directly communicate their requirements to their UE proxy through extensions to the 3GPP RAN control plane (via mobile OS APIs to expose these extensions). The UE proxy directly controls bandwidth allocations through the eNodeB scheduler. Because the market proxy also resides on the eNodeB, it has full access to cell utilization information when setting prices and no latency between the UE and market proxy. The market proxy can also compare reported traffic and real traffic to detect price manipulation. While this represents the cleanest and most functional design, it requires changes to the 3GPP protocol to exchange demand and price information, to eNodeBs, and to the mobile OS, and thus may be difficult to deploy.

**Incremental:** The UE proxy is deployed on the mobile device as a daemon process, while the market proxy is deployed by the network provider as a new network element in the packet core. The UE proxy interacts with mobile applications through user library calls for collecting delay constraints and allocating bandwidth (via a controlled socket abstraction). Interaction between the UE and market proxy for exchanging demand and price information occurs using the normal cellular network data plane, through a well known UDP port and a special destination IP address that points to the market proxy for the current cell. This configuration imposes higher latency between the UE and market proxies, but because the market proxy is deployed by the network provider, it can be made as low as possible. Also, the market proxy may be given access to real-time traffic information through a private interface to the eNodeB as well as the provider's charging and data metering systems [21]. Thus, it provides most of the benefits of the clean slate design while being easier to deploy - 3GPP protocols or eNodeBs do not have to be extensively modified on the provider side, while the UE proxy can be implemented as a user space library without modifying the mobile OS on the UE side.

**Over-the-top:** The UE proxy is deployed as a library on the UE just as in the incremental design, but the market proxy is deployed as a third-party service on an external cloud server, possibly even on an application-by-application basis. E.g., a large video streaming provider may have its own market proxy that serves to smooth only its own traffic within a cell and improve the performance for its own users. In this design, when a mobile device connects a cell, its UE proxy uses the cell ID to find the IP address of the corresponding market proxy through standard DNS mechanisms. UE and market proxies exchange information through UDP as in the incremental design, but the third party nature of the market proxy precludes the market proxy from basing its pricing decisions based on real-time traffic information (e.g., it cannot lower prices during periods of low utilization to incentivize more aggressive transfers) or from actually providing real monetary incentives to users. However, the design requires no modification to any cellular network elements, or to the mobile OS, and thus is the easiest to deploy.

Due to its ease of deployment, we chose the over-the-top design for our prototype. However, the goal of this paper is to evaluate the feasibility of CoAST and quantify its benefits and costs instead of advocating a particular deployment choice. Our evaluation will hold irrespective of the design chosen.

## 5.2 Prototype Implementation

We implemented market proxy on a Linux system with a dual-core 2.53 GHz CPU and 4GB of RAM. The market proxy divides time into 1-second slots and dynamically sets the prices for future 30 time slots. Every 100ms the market proxy collects the traffic demands for the future 30 time slots from all mobile devices that connect to it. Then it updates the prices based on the aggregated traffic demand and sends the new prices to all the mobile devices.

Ideally the market proxy should reside on a cell-level network element like the eNodeB in a 4G LTE network or the RNC in a 3G network where it can also monitor the traffic over the cell. However, because we have no access to such network elements, we have to run the market proxy on a remote server whose RTT to the mobile devices through a 4G LTE network is 57ms on average. Since the RTT between the market proxy and UE proxies is much smaller than their interaction interval (i.e., 100ms), the functionality of the control plane will not be affected.

We implemented the UE proxy on Android 4.1. It is implemented as an Android application collecting delay information from mobile applications and the prices from the market proxy and assigning throughput caps to the mobile traffic. To dynamically control the downlink traffic from the receiver side, we also patched the Android kernel to add a new socket option, DL_CAP, to limit the maximal throughput of the TCP flows at runtime.

### *5.2.1 Communication and Computation Overhead*

The traffic demands and prices are the only data exchanged between the UE proxy and the market proxy. In the current implementation, we use 2 bytes for the traffic demand and 1 byte for the

price in each time slot. Therefore, each UE proxy uploads 60 bytes and downloads 30 bytes from the market proxy in every round. Since they exchange information every 100ms, the control overhead in this case is 600 Bytes/s in the upload direction traffic and 300 Bytes/s in the download direction in the worst case. The overhead is much lower in reality because of two reasons. First, the UE proxy will exchange information with the market proxy only if some communication-intensive applications (e.g., video streaming) are running. Second, compared with the high traffic volume of those target applications, the extra communication overhead of CoAST is negligible. As the experiment in Section 6.2 will show, CoAST only leads to 0.07% extra traffic.

The computation overhead is also very low. The market proxy updates the prices by solving a projection problem. Its computational complexity is $O(\kappa)$ where $\kappa$ is the number of time slots (30 in our implementation). The UE proxy needs to solve an optimization problem that minimizes the cost under the delay constraints. As we discretize time into time slots, the maximal number of delay constraints for a flow is $O(\kappa)$. The optimization problem can be solved by gradually finding the minimal cost for each constraint. Therefore, its computational complexity is $O(f \times \kappa^2 \times log(\kappa))$, where $f$ is the number of concurrent flows. On an old Motorola ATRIX smartphone with Android 2.3, it takes less than 1ms for the UE proxy to solve the problem for 100 concurrent flows.

# 6. SYSTEM EVALUATION

In this section, we evaluate the CoAST prototype as described in Section 5.2. We will demonstrate how CoAST improves the performance of mobile applications under various LTE network congestion states.

## 6.1 Experimental Setup

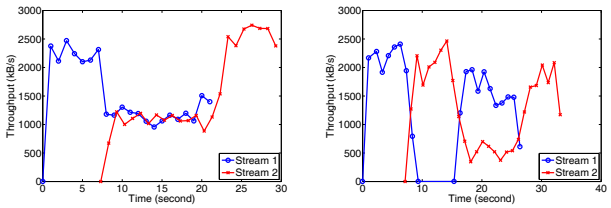
Our testbed is composed of 5 Linux workstations and 4 Android smartphones with 4G LTE capability. The first workstation acts as the market proxy. Its average RTT to these smartphones through the LTE network is 57ms. The other 4 workstations act as the remote mobile application servers, each of which serves one smartphone. The Linux traffic control tool *tc* is used on application servers to control their available bandwidth. We make sure that all these smartphones connect to the same cell during the experiments by checking the cell id of their connected cell. All experiments are conducted in a residential area around midnight to reduce the impact of other cellular users.

CoAST is designed to improve the performance of mobile applications when the LTE network is congested. We use following mobile streaming strategies to create different levels of network congestion and streaming behavior. The evaluation for other types of mobile applications (e.g. web browsing) will be presented in Section 7. We find that the streaming strategies impact the distribution of mobile traffic and how they interact with each other in CoAST.

- **All-at-once:** strategy aggressively transfers data from the server to the client as fast as possible. This strategy is usually used by some audio streaming [22] and video streaming applications on some specific phones [28].

- **Pacing:** strategy controls the download throughput at the "steady state" after initial buffering. This strategy is widely used in video streaming services such as Youtube, Hulu, and Netflix [10].

- **Bundling:** strategy is proposed to reduce the energy consumption of video streaming applications [28]. It divides the entire stream into several large chunks and aggressively transfers each chunk periodically.




(a) Native LTE network (b) CoAST

**Figure 7: The interaction between two video streams**

We use two metrics to evaluate the impact of CoAST on mobile users:

- **Buffering time:** It is a direct measure of the user experience for streaming applications. We consider both initial buffering period and rebuffering period in the steady state.

- **Energy consumption:** The energy consumption of video streaming is primarily caused by the device screen in the "on" state and the LTE network interface. For fair comparison, we use the LTE energy model that calculates energy consumption based on traffic traces [15].

## 6.2 The Reduction of Buffering Time in Video Streaming

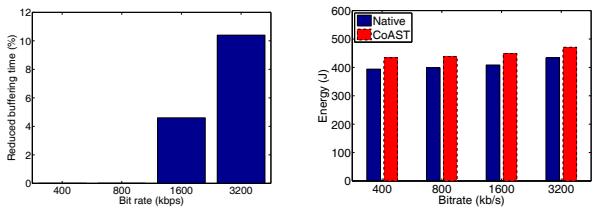
We first demonstrate the basic mechanism through which CoAST enables collaborative traffic scheduling via a simple two-device video streaming experiment. Let the two devices start streaming two HD videos at time 0s and 7s, respectively. Both of them use the all-at-once strategy. The LTE cell is the only bottleneck of both streams.

Figure 7 plots the throughputs of two streams for the native LTE network and CoAST cases. In the native LTE case, stream1 is unaware of the traffic demand of stream2 and, thus, continues downloading data from its server even after stream2 starts. Therefore, the initial buffering period for stream2 is twice as long as that of stream1. In CoAST, when stream2 starts at 7sec with an empty buffer, it tries to download aggressively, thus causing the market proxy to increase prices due to the increased demand. Because stream1 has a relatively full buffer, it is less willing to pay the increased price than stream2, and thus delays its traffic. After some time, when stream2 has buffered enough data for playback, it reduces its willingness to pay higher prices, and thus begins delaying its data more. In the meantime, stream1 has already played back a portion of data in its buffer and becomes more aggressive to meet its delay constraints. Due to this cooperative inter-play between the UEs — the UE with full buffer deferring its download in favor of the UE with empty buffer— the buffering time of stream2 is reduced by 50% while that of stream1 is not affected.

By exchanging traffic demands and prices between the UE proxies and the market proxy, CoAST incurs a total of 47.5 kB extra traffic in the experiment. Compared with the total download traffic (i.e., 64 MB), CoAST only incurs a 0.07% communication overhead.

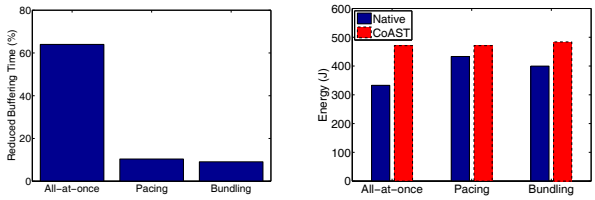
## 6.3 The Impact of Network Congestion

Next we evaluate the impact of traffic demand on CoAST performance by varying the stream bitrate from 400 kb/s (i.e., average Youtube bitrate [11]) to 3200 kb/s while keeping the number of




(a) The reduced buffering time (b) Energy consumption

**Figure 8: The impact of taffic demand on CoAST performance**




(a) The reduced buffering time (b) Energy consumption

**Figure 9: The impact of streaming strategies on CoAST performance**

streams unchanged. We randomly start 4 streams within 30 seconds. The length of each stream is 3.33 minutes, i.e., the median Youtube video length [11]. The upload bandwidth of servers are unlimited, i.e., the LTE cell is the only bottleneck. Each experiment is repeated 3 times with different random seeds. The average values are reported.

Figure 8 plots the reduced buffering time and average energy consumption of the smartphones. It is clear that the benefits of CoAST start increasing with the increase in traffic demand (i.e. bitrate in the experiments). When the bitrate is 3200 kb/s, CoAST is able to reduce buffering time by more than 10% on average. It is exactly the design goal of CoAST, i.e., to reduce the impact of increased mobile traffic on user experience.

Figure 8(b) also shows that CoAST increases the energy consumption by less than 8% in all the experiments. More importantly, with the increase in traffic demand, the extra energy consumption is even smaller (e.g. 7% for 3200 kb/s case). The increase in energy consumption is caused by longer streaming time in CoAST. Note that energy consumption is based on the assumption that there is no background traffic. Otherwise, CoAST will incur even smaller energy consumption overhead.

## 6.4 The Impact of Streaming Strategies

Finally we evaluate the impact of streaming strategy on the performance of CoAST. The stream bitrate is 3200 kb/s. Other parameters are the same as previous experiments. The experiment results are shown in Figure 9. When the all-at-once strategy is used, CoAST reduces buffering time by 66%. However, it also increases the energy consumption significantly if the entire video is downloaded. Compared to the pacing strategy, CoAST incurs similar buffering time and slightly more energy consumption in the bundling strategy.

# 7. TRACE-DRIVEN EVALUATION

In this section, we demonstrate the potential benefits of CoAST by using trace-driven evaluations of real-world cellular traffic traces from a large US mobile network operator.

## 7.1 Experimental Setup

We implement CoAST on a packet-level simulator, ns-3 [3], which supports the simulation of LTE networks. The network is composed of an eNodeB, a remote server and multiple mobile devices. The market proxy is installed on the eNodeB, while the application servers are installed on the remote server. All mobile devices connect to the same eNodeB during the experiments. We use the default parameters for all the experiments.

**Traffic traces:** We identify the top 100 heavily-loaded cell sectors in our cellular traffic dataset and evaluate how CoAST reduces their peak throughputs within 1 day. For each cell, we generate the flows for Youtube video streaming and web browsing as follows.

- **YouTube video streaming:** We identify all flows from Youtube servers to the mobile devices in the cellular traffic dataset. But only flows whose size and duration are large enough (i.e., size > 100 kB and duration > 10 s) are treated as streaming flows. Since we don't have video information (e.g., bitrate, video length) in our dataset, we use the empirical models from [11] to generate the video profile for each flow. The pacing strategy is used to control the stream.
- **Web browsing:** We first identify all web traffic using port number 80. If two flows between the same source and destination start within 5 seconds, they are considered as belonging to the same browsing event. Using this method, we obtain a set of browsing events with their start time instants. Since the dataset doesn't contain the identity of the web page, we randomly pick one of the top 500 Alexa websites for each browsing event.

**Metrics:** We compare CoAST against the native network using the following metrics:

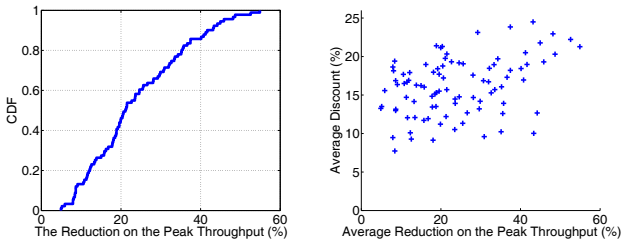
- **Peak reduction**: This is the most important metric because the goal of CoAST is to reduce the peak cell throughput. It is defined as $\frac{Peak_{native}-Peak_{CoAST}}{Peak_{native}}$.
- **Discount**: A key promise of CoAST is that mobile users will also benefit from CoAST and, thus, be willing to use it. We use $\frac{D_t-D_a}{D_t}$ to denote the user benefit, where $D_t$ is the amount of transferred data, $D_a$ is the amount of accounted data.
- **Overhead**: We also analyze if CoAST causes any overhead, including energy consumption and RRC signaling overhead [14].

We acknowledge that our trace-based evaluation has some limitations. First, there is no system feedback. By scheduling the traffic better, CoAST may cause mobile users to use more data and, thus, increase the peak traffic. However, our experiments cannot capture this phenomenon. Second, the user behavior in using applications is not available. For video streaming, the user may skip ahead or pause the video. For web browsing, the user may quickly scroll down in the webpage. These user behaviors impact the delay constraints of the corresponding traffic and, thus, affect the performance of CoAST.

## 7.2 Experimental Results

### *7.2.1 Video Streaming Experiments*

We first perform simulation-based video streaming experiments using 100 heavy-loaded cells for 1 day. In all experiments we only consider Youtube video streams. As shown in Figure 10, CoAST




(a) The Peak Reduction (b) Discount

**Figure 10: The performance of video streaming supported by CoAST on various sectors.**

successfully reduces the peak throughput for all cells. We also observe the following phenomena. First, the peak reduction ranges from 5% to 55% for different cells, as shown in Figure 10(a). More than 30% cells reduce their peak by 30%. The high variation of the peak reduction among different cells is probably caused by diverse distribution of Youtube video streams among various cells. Cells with more video streams are able to achieve better performance. Second, the average discount obtained by the mobile users correlates with the peak reduction on the cells, as shown in Figure 10(b). Finally, none of the videos is paused during the experiments.

Next we explore how the reduced peaks actually help increase the capacity of cellular networks by examining the impact of increasing user demand on application level performance with and without CoAST. Specifically, we increase the users/spectrum ratio and measure how long the video streams pause waiting for more data. However, rather than increasing the number of users in a cell artificially, we increase the ratio by reducing the effective bandwidth (spectrum) available to a cell by a fraction $\alpha \in [0, 1)$, i.e., Capacity = $(1 - \alpha)\times$ MaxCapacity. The average values of video pause time over 100 cells are reported for different values of $\alpha$ in Figure 11. We see that CoAST results in very little application performance degradation while supporting a capacity increase of up to 20% ($\alpha = 0.2$). Also, for the same value of application performance degradation, CoAST can support more users per unit of available spectrum.

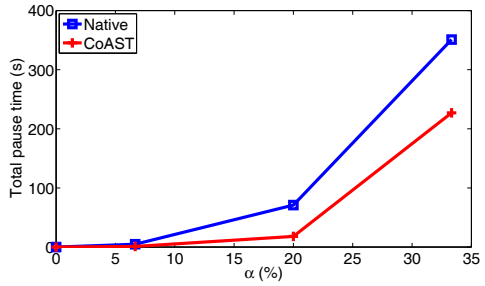
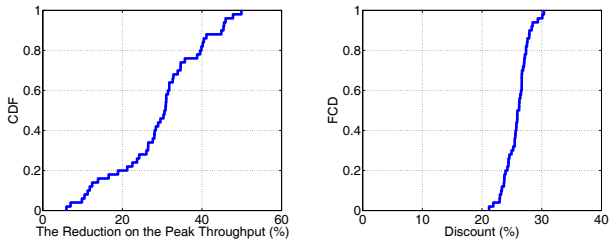



**Figure 11: Comparison of video pause time for different number of users with and without CoAST.**

### 7.2.2 *Experiments on Web Browsing*

Web browsing is another important application that can benefit from CoAST because of its delay tolerance. Unlike video streaming, web browsing flows are relatively small. In this subsection, we analyze the performance of CoAST-based web browser.

First, we consider the simple scenario that all cellular traffic are web traffic. For each browsing event, we randomly choose one of

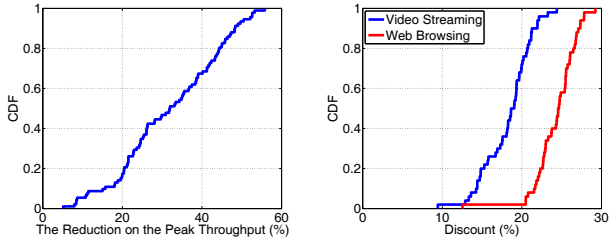



(a) The Peak Reduction (b) Discount

**Figure 12: The performance of web browsing supported by CoAST on various sectors.**

the top 500 Alexa websites in the "news" category. We report the results for the 100 heavy-load cells in Figure 12.

As expected, CoAST-based web browser reduces the peak throughput for all cells. Like in video streaming experiments, the peak reduction varies significantly among cells, ranging from 6% to 50%. However, different from video streaming, CoAST-based web browser helps more than 55% cells achieve more than 30% peak reduction. In addition, as shown in Figure 12(b), the average discount obtained by the mobile users ranges from 22% to 31%, which is more than video streaming case. This is because web browsing flows are usually very small and are able to take advantage of the variation in price.




(a) The Peak Reduction (b) Discount

**Figure 13: The performance of both video streaming and web browsing supported by CoAST.**

Next, we evaluate the scenario in which both video streaming and web browsing use CoAST to schedule their traffic. Video streaming and web browsing are the most important mobile applications, accounting for more than 70% mobile traffic. The experimental results are plotted in Figure 13. In this more realistic scenario, CoAST helps all cells to reduce their peak throughputs. We also plot the discount obtained by video streaming and web browsing in Figure 13(b). Web browsing is still able to obtain higher discount than video streaming.

In Section 6, we noticed that CoAST slightly increases the energy consumption in some scenarios. This is because by delaying mobile traffic, mobile devices need to keep the network interface active for longer duration, resulting in extra energy consumption. To analyze the overhead of CoAST, we assume that the mobile device is initially in the RRC_IDLE state, and there is no other traffic on the mobile device. We use the RRC state model [15] to analyze the overhead. For both video streaming and web browsing, the mobile devices always stay at the RRC_CONNECTED state when using these applications. Thus, CoAST does not introduce extra RRC state transitions. However, CoAST does keep cellular interface alive for slightly longer duration and, thus, consumes a little more energy. Figure 14 shows that in CoAST, the mobile de-

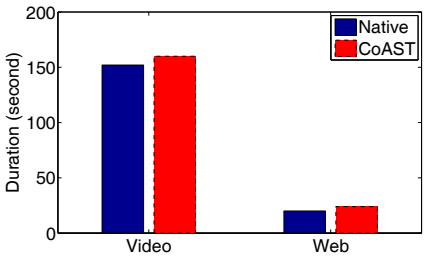



**Figure 14: The time duration at RRC_CONNECTED state for video streaming and web browsing.**

vices stay slightly longer in the RRC_CONNECTED state than the native case.

### 7.2.3 *The Impact of Partial Deployment*

CoAST reduces the peak throughput by rescheduling the traffic of mobile devices under its control. The number of participating mobile devices will impact the CoAST performance. In this subsection, we evaluate the performance of CoAST when only a portion of mobile devices support it.

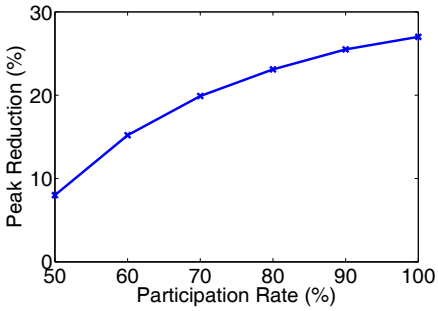



**Figure 15: The impact of partial deployment on CoAST performance.**

In this set of experiments, we randomly select $r\%$ mobile devices to support CoAST, where $r$ varies from 50 to 100. The video streaming application is used in the evaluation. Each experiment is repeated 10 times with different random seeds. We report the average value of all the 100 cells.

The experimental results are shown in Figure 15. It's clear that the participation rate has significant impact on the CoAST performance. When $r$ reduces from 100 to 50, the average peak reduction decrease from 27% to 8%. We also observe that the slope of the curve decreases with the increase of the $r$ value. This is because the peak value of the background traffic will be higher when the participation rate is low. Thus, according to Theorem 2, the peak reduction achieved by CoAST will be much lower. This set of experiments indicate the importance of increasing the participation rate in CoAST.

To analyze whether partial deployment will impact the user experience of early adopters or non-adopters, we compare the buffering time of streaming applications in the above experiments and that in LTE networks. The buffering time of each stream in both scenarios are the same in all experiments. This is because CoAST tries to satisfy the delay constraints of all adopters. When the network is not very congested, their delay constraints can be easily satisfied. In contrast, when the network is persistently congested which is uncommon in our dataset, adopters behave the same as non-adopters, i.e., downloading as fast as possible without rescheduling their traffic.

Therefore, the partial deployment will primarily impact the peak reduction with little impact on the user experience of mobile applications.

### 7.2.4 *The Impact of Design Choices*

In this subsection, we evaluate the impact of CoAST parameters and network factors on the performance of CoAST.

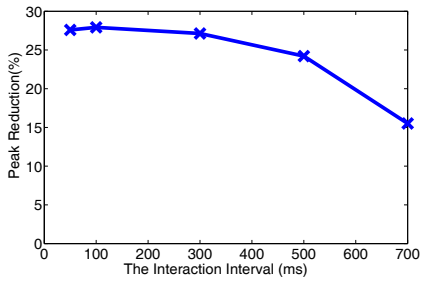



**Figure 16: The impact of interaction frequency between UEs and the market proxy.**

In the first set of experiments, we analyze the impact of the interaction frequency between the UE and Market proxies. We change the interaction interval (i.e., $\delta$) from 50 ms to 700 ms, while other parameters are kept unchanged. The video streaming application is used in the evaluation. We report the average value of all the cells.

The experimental results are shown in Figure 16. When the interaction interval is less than 100 ms, CoAST achieves similar performance in terms of peak reduction. As the interval increases from 100 ms to 500 ms, the average peak reduction slightly drops from 27% to 24%. When it further increases to 700 ms, the obtained peak reduction is quickly reduced to 15%. This is because the control plane of CoAST utilizes an iterative protocol that gradually optimizes its performance in each iteration. When interaction interval is too large, it is hard for the system to converge to the optimal solution. On the other hand, increase in interaction frequency results in more communication overhead. From our experiments, we find that 100 ms is a good choice as it achieves good tradeoff between performance and overhead.

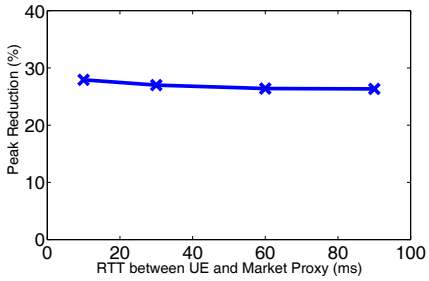



**Figure 17: The impact of the delay between UEs and the market proxy. The interaction interval is 100 ms.**

In the second set of experiments, we analyze the impact of the distance between the UE and market proxies. We vary the RTT between them from 10 ms to 90 ms. The results are reported in Figure 17. We observe that the RTT between UE proxy and market proxy has very small impact on peak reduction. This indicates that CoAST can still achieve good performance even if it is not possible

to deploy the market proxy in cellular network elements close to the end user devices, like eNodeB in LTE or NodeB in 3G networks.

# 8. DISCUSSION

In this section, we discuss how the mechanisms proposed by CoAST interact with other existing and proposed mechanisms in the RAN.

**3GPP quality of service**: The LTE specification provides a QoS model based on Quality Class Indicators (QCI) [5]. A UE may create bearers with one of up to 9 QCI classes, each with a different priority (diffserv), packet delay budget, loss rate, and bitrate guarantee. Many cellular providers today reserve QCI only for managed services (e.g., IMS), and do not expose QCI classes to third party applications. Because CoAST is an over-the-top protocol that can operate without any support from 3GPP infrastructure, it is applicable even on networks which do not expose QCI. Furthermore, QCI provides a static and inflexible partitioning of applications into a small number of priorities. It is not sufficient for scenarios presented by both our examples — streaming and web browsing — in which the *same* application requires different QoS and delay tolerance at different times, depending on the context. Furthermore, any non-collaborative mechanism cannot coordinate behavior across multiple UEs the way that CoAST does.

**Congestion aware pricing/control**: Access to real-time congestion information at the eNodeBs is not exposed through 3GPP standardized interfaces. Therefore, to remain independent of vendors-specific implementations, CoAST does not assume any access to the eNodeB, including information about whether there is cell congestion or not. Instead, it tries to continuously minimize the peaks across the applications whose traffic is managed through its APIs, independently of other background traffic. If real-time congestion information could be made available, it could easily be used to trigger when CoAST optimization mechanisms kick-in and provide improved fidelity and price control to the market proxy's demand estimation step. We leave this extension to future work.

**RNC/eNodeB schedulers**: The UMTS RNC or LTE eNodeB have a scheduler that allocates scrambling codes (variable sized slices of the spectrum) to UEs every 2ms based on their demands, QCI, channel noise, and overall cell congestion. Beyond differentiation using QCI, this scheduler is *application context agnostic*. CoAST does not interfere with this scheduler because it operates at a much coarser granularity (hundreds of msec). CoAST adds an additional application aware layer of control on the top, and helps the RAN scheduler by reducing demand peaks themselves, thus reducing the need for the RAN scheduler to allocate less than what UEs demand. However, the RAN scheduler can have an impact on applications that use CoAST because it may restrict the bandwidth available to a UE (because of noise or congestion), and thus decrease the amount of time an application's data can be delayed. To solve this problem, CoAST should take the radio link condition into consideration. We leave it for future work since it requires deeper integration with the eNodeB scheduler.

**Energy vs. congestion tradeoffs**: Several proposals have been made in the literature to help mobile devices save energy by batching mobile traffic into short concentrated bursts and reducing the time UE radios spend in the active state, e.g., [24, 15]. It would seem that CoAST proposes the opposite philosophy—spread out traffic to minimize congestion. However, in reality, these two mechanisms are relatively complementary. CoAST can just as easily work with applications where data transfers occur in bursts—e.g., web browsing. All CoAST advocates is that when there is contention, priority be given to the traffic on which users are waiting as opposed to applications that are just filling up their buffers. While this can increase the amount of time needed to transfer an application's background data, as Figure 14 shows, the increase is not substantial. An interesting future use of CoAST is for applications to increase their price based on how much available battery they have, thus prioritizing their transfers over everyone else.

**Potential for price manipulation**: Because CoAST's mechanisms do not force users to transfer any data after they have indicated their demand forecast, it is possible that malicious users may increase the price others have to pay by falsely forecasting high demands. While it is not easy to detect one-off instances of such behavior (a user's demand may legitimately have changed), it is easy to detect systematic abuse over a period of time by statistically comparing forecasts to actual data transfers using cellular providers existing data tracking mechanisms. Abusers may then have their bids ignored in the future, thus effectively removing them from the set of CoAST managed devices. With incremental deployment, it is possible that some legitimate applications don't support CoAST, resulting in the discrepancy between reported traffic and the real traffic. To solve this problem, the UE proxy needs to report the flow information of those adopters, while the CoAST will only monitor those flows.

**Impact of handover**: Finally, we briefly describe CoAST's interactions with mobility mechanisms, i.e., handover. UEs detect when a handover takes place by querying their baseband chip for the current cell id. They inform the market proxy for every handover, which then simply assigns the UE's projected demand to the new cell and recomputes the demand for both the old and new cells. Because we expect each market proxy to cover a relatively large area (we expect one market proxy per P-GW), the number of inter-proxy handoffs are few and are handled by the UE simply reconnecting to the new market proxy.

# 9. RELATED WORK

The idea of time shifting traffic to reduce the peak throughput over a link is not brand new. Many ISPs use traffic sharping against bulk flows to reduce the peak of inter-domain traffic [19]. Laoutaris et al. [17, 16] proposed intentionally using diurnal variations in Internet traffic to transfer delay tolerant bulk data over the Internet at off-peak times. With peak pricing, these shifts can also reduce cost. Recently, Ha et al. [12] applied the idea of medium-to-long time scale shifting to cellular networks and proposed a time-dependent pricing mechanism, TUBE, to motivate mobile users to shift some cellular traffic sessions from peak time to off-peak times in exchange for lower prices. CoAST is fundamentally different from TUBE in many aspects. First, TUBE utilizes medium-to-long time-scale variation of the background traffic and thus are only suitable for applications that can tolerate substantial delays. Second, CoAST requires no involvement of mobile users, while TUBE requires user change their behavior. Third, CoAST also has a data plane to schedule traffic, while TUBE only focuses on the control plane. In addition, our approach is compatible with medium and long time scale traffic shifting: our work can reduce the short time scale peaks that will remain after other data is time shifted by hours or 10s of minutes.

In addition to reducing traffic peaks, time shifting mobile traffic can be beneficial in saving device energy. For example, one set of approaches [8, 27] utilize the observation that aggregation of traffic can reduce energy by avoiding the excessive energy-consuming RRC transitions and tail energy that occur in 3G and LTE networks [24, 15] when traffic is transfered in disjoint time periods. Studies show that mobile applications have sufficient periods of sparse traffic transfer to make these approaches useful for energy saving with relatively little traffic delay [23]. Another approach

saves device energy by scheduling transfers when signal strength is strong [26], leveraging the observation that energy consumption per bit increases when signal strength degrades.

Our approach is not directly compatible with approaches that use scheduling to reduce energy because both approaches operate on a similar time scale but with different objective functions. To use an energy saving scheduler with a peak reduction scheduler would require an integrated objective function that minimizes a combination of device energy and cost to use the shared link, while meeting deadlines. An adaptive approach may be most appropriate, where reducing cost is favored when device energy is plentiful, and reducing energy is favored when device energy is low. In its most simplistic form, a controller could simply switch from one scheduler to the other based on a device energy threshold. More complex approaches are obviously possible and are left to future work.

# 10. CONCLUSIONS AND FUTURE WORK

In this paper, we present a new approach to improving the capacity of cellular network cells through application-aware collaborative microscheduling and delaying of traffic. Our implementation, CoAST, supports applications such as streaming and web-browsing that are not normally considered to be delay tolerant, but yet account for over 70% of cellular network traffic today. Our extensive evaluation demonstrates shows the approach's potential by showing that it can reduce traffic peaks by up to 50%, and increase the capacity of cells to serve such workloads by up to 20% without any degradation to user experience. There are a number of future issues to consider. These include implementing a fully functioned web browser that supports CoAST and field test of the system with mobile users, and incorporating additional factors such as energy use and congestion awareness into CoAST's scheduling algorithms.

## Acknowledgments

We would like to thank our shepherd, Chunyi Peng, and the anonymous reviewers for their insightful feedback. Cong Shi, Mostafa Ammar, and Ellen Zegura are supported in part by the US National Science Foundation through grant CNS 1161879.

# 11. REFERENCES

[1] Cisco systems. cisco visual networking index: Forecast and methodology, 2011-2016, may 30 2012. `http://tinyurl.com/VNI2012`.

[2] Huffington post. `http://www.huffingtonpost.com`.

[3] ns-3. http://www.nsnam.org/.

[4] PhantomJS: Headless WebKit with JavaScript API. http://phantomjs.org/.

[5] 3GPP TS 25.214: 3rd Generation Partnership Project; Technical Specification Group Radio Access Network; Physical layer procedures (FDD) (Release 8), 2010.

[6] Alexa. Top 500 website. http://www.alexa.com/.

[7] AT&T. 2011 annual report, 2011. http://bit.ly/Lf0goQ.

[8] N. Balasubramanian, A. Balasubramanian, and A. Venkataramani. Energy consumption in mobile phones: a measurement study and implications for network applications. In *ACM IMC*, pages 280–293, 2009.

[9] H. Falaki, D. Lymberopoulos, R. Mahajan, S. Kandula, and D. Estrin. A first look at traffic on smartphones. In *Proceedings of the 10th annual conference on Internet measurement*, pages 281–287. ACM, 2010.

[10] M. Ghobadi, Y. Cheng, A. Jain, and M. Mathis. Trickle: rate limiting youtube video streaming. In *USENIX ATC*, pages 17–17, 2012.

[11] P. Gill, M. Arlitt, Z. Li, and A. Mahanti. Youtube traffic characterization: a view from the edge. In *ACM IMC*, 2007.

[12] S. Ha, S. Sen, C. Joe-Wong, Y. Im, and M. Chiang. Tube: time-dependent pricing for mobile data. In *ACM SIGCOMM*, pages 247–258, 2012.

[13] S. Hao, D. Li, W. G. Halfond, and R. Govindan. Sif: A selective instrumentation framework for mobile applications. In *ACM MobiSys*, 2013.

[14] H. Holma and A. Toskala. *Hsdpa/Hsupa For Umts*. Wiley Online Library, 2006.

[15] 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 *ACM MobiSys*, pages 225–238, 2012.

[16] N. Laoutaris, M. Sirivianos, X. Yang, and P. Rodriguez. Inter-datacenter bulk transfers with netstitcher. *SIGCOMM Comput. Commun. Rev.*, 41(4):74–85, Aug. 2011.

[17] N. Laoutaris, G. Smaragdakis, P. Rodriguez, and R. Sundaram. Delay tolerant bulk data transfers on the internet. In *ACM SIGMETRICS*, pages 229–238, 2009.

[18] G. Maier, F. Schneider, and A. Feldmann. A first look at mobile hand-held device traffic. In *Passive and Active Measurement*, pages 161–170. Springer, 2010.

[19] M. Marcon, M. Dischinger, K. P. Gummadi, and A. Vahdat. The local and global effects of traffic shaping in the internet. In *IEEE COMSNETS*, pages 1–10, 2011.

[20] D. P. Palomar and M. Chiang. A tutorial on decomposition methods for network utility maximization. *IEEE Journal on Selected Areas in Communications*, 24(8):1439–1451, 2006.

[21] C. Peng, G.-h. Tu, C.-y. Li, and S. Lu. Can we pay for what we get in 3g data access? In *ACM MobiCom*, pages 113–124, 2012.

[22] F. Qian, Z. Wang, A. Gerber, Z. Mao, S. Sen, and O. Spatscheck. Profiling resource usage for mobile applications: A cross-layer approach. In *ACM MobiSys*, 2011.

[23] F. Qian, Z. Wang, A. Gerber, Z. Mao, S. Sen, and O. Spatscheck. Profiling resource usage for mobile applications: a cross-layer approach. In *ACM MobiSys*, pages 321–334, 2011.

[24] F. Qian, Z. Wang, A. Gerber, Z. M. Mao, S. Sen, and O. Spatscheck. Characterizing radio resource allocation for 3g networks. In *ACM IMC*, pages 137–150, 2010.

[25] Sandvine. Global internet phenomena report. `http://www.sandvine.com/news/global_broadband_trends.asp`, 2012.

[26] A. Schulman, V. Navda, R. Ramjee, N. Spring, P. Deshpande, C. Grunewald, K. Jain, and V. N. Padmanabhan. Bartendr: a practical approach to energy-aware cellular data scheduling. In *ACM MobiCom*, pages 85–96, 2010.

[27] A. Sharma, V. Navda, R. Ramjee, V. N. Padmanabhan, and E. M. Belding. Cool-tether: energy efficient on-the-fly wifi hot-spots using mobile phones. In *ACM CoNEXT*, pages 109–120, 2009.

[28] M. Siekkinen, M. A. Hoque, J. K. Nurminen, and M. Aalto. Streaming over 3g and lte: How to save smartphone energy in radio access network-friendly way. In *Proceedings of the 5th Workshop on Mobile Video*, 2013.

## Appendix

PROOF 1. The goal of the market proxy is to minimize the maximal throughput on a cell sector over time, as shown in Formula 2. To decompose this problem and derive a distributed protocol, we rewrite the objective of the market proxy as follows:

$$\min \quad \alpha \tag{6}$$

$$\text{s.t.} \quad \forall t, \sum_i th_i(t) \leq \alpha \tag{7}$$

Introducing a dual variable $p(t) > 0$ (i.e., price for downlink traffic through the cell sector at time slot $t$) for each constraint of (7), we define the Lagrange dual function

$$L(\{p(t)\}) = \min \alpha + \sum_t p(t)(\sum_i th_i(t) - \alpha) \tag{8}$$

To make $L(\{p(t)\})$ finite, the coefficient of $\alpha$ should be 0:

$$\sum_t p(t) = 1 \tag{9}$$

Then, we simplify $L(\{p(t)\})$ to

$$L(\{p(t)\}) = \min \sum_t p(t) \sum_i th_i(t) \tag{10}$$

$$= \min \sum_i \sum_t p(t) th_i(t) \tag{11}$$

The original problem can be decomposed into independent problems for each mobile applications. If we aggregate the throughput by UEs, we obtain

$$L(\{p(t)\}) = \min \sum_e \sum_{i \in e} \sum_t p(t) th_i(t) \tag{12}$$

$$= \sum_e \min \sum_{i \in e} \sum_t p(t) th_i(t) \tag{13}$$

Therefore, the original problem is decomposed into independent sub-problems for each UE. The objective of each UE is to select $th_e(t)$ among all feasible values so that $\sum_{i \in e} \sum_t p(t) th_i(t)$ is minimized. The derived objective of each UE is exactly the same as (4).

The market proxy, running the master program of the decomposition method, gets feedbacks (i.e., $th_e(t)$) from UE proxies and updates $\{p(t)\}$ using a projected sub-gradient method as described in (3). Therefore, the objective of Formula (2) is equivalent to the control protocol in Section 4.3. □

PROOF 2. Let $x_t = \sum_{i=1}^{n} th_i(t)$ have a probability density function $f(x_t)$ and a cumulative probability distribution function of $F(x_t)$. Let $y_t = th_b(t_t)$ have a probability density function $g(y_t)$ and a cumulative probability distribution function of $G(y_t)$. Let $x_{max}$ and $y_{max}$ be the lowest value of $x_t$ and $y_t$ such that $F(x_t) = 1$ and $G(y_t) = 1$, respectively. Since the number of flows associated with a cell is limited, $x_{max}$ and $y_{max}$ are finite. Let $p$ be the convolution of $f$ and $g$, and $P$ be its cumulative probability distribution function. Then the sum $z_t = x_t + y_t$ is a random variable with the density function $p(z_t)$ and the cumulative probability distribution function of $P(z_t)$. Since $x_t$ and $y_t$ are independent, $P(z_t \leq x_{max} + y_{max}) = 1$.

Let $z^* = \max_t z_t$ where $t \in \{1, 2, \ldots, T\}$. The probability density function of $z^*$, $h(z^*)$ is

$$h(z^*) = T[P(z^*)]^{T-1} p(z^*) \tag{14}$$

The expected value of $z^*$ is

$$E(z^*) = \int_{-\infty}^{+\infty} z^* T[P(z^*)]^{T-1} p(z^*) dz^* \tag{15}$$

$$= \int_0^1 P^{-1}(z^{*\frac{1}{T}}) dz^* \tag{16}$$

Therefore, when $T \to +\infty$, we have

$$\lim_{T \to +\infty} E(z^*) = \int_0^1 P^{-1}(1) dz^* = x_{max} + y_{max} \tag{17}$$

Since CoAST minimizes $x_{max}$ with $y_{max}$ unchanged, it is equivalent to minimizing $\max_t \sum_i th_i(t) + y_{max}$ when $T \to +\infty$. □