Minimizing Capacity Requirements of Cellular Networks via Delayed Scheduling

Navid Hamed Azimi†, Himanshu Gupta†, Utpal Paul†, Milind Madhav Buddhikot††, Samir Das†

***Abstract*—The volume of data in broadband cellular network is growing exponentially. However, studies have indicated the traffic load on the cellular base stations varies significantly over time. This gives an opportunity to accommodate additional traffic with the same network capacity if some of the traffic (e.g., p2p, cloud sync) can be amenable to 'delayed scheduling' without hurting the user experience any significantly. In this paper, we study various algorithmic problems that can arise in this context. Using a model where all flows can have certain flexibility in scheduling (via use of a 'deadline'), we develop optimal or near-optimal algorithms to determine the minimum network capacity for two different models. We also develop various semi-online and online algorithms for online scheduling of flows, and analyze their performance. In particular, even though the online scheduling problem is shown to be intractable, our proposed semi-online algorithm can schedule flows optimally if aided by historical data and slightly additional network capacity over the optimal. Finally, using flow level traffic traces collected at the core of a commercially operated cellular network, we evaluate the effectiveness of our techniques. Evaluations show that delayed scheduling, when done efficiently (using an offline optimal algorithm), can accommodate the same traffic with much lower network capacity (up to 50% less) with only modest delays. While such an optimal solution needs an offline approach, we demonstrate that online scheduling can be almost equally effective when historical traffic data can be exploited for estimation purposes.**

I. Introduction

Broadband cellular networks are emerging to be the most common means for mobile data access worldwide. Predictions from industry analysts indicate that the volume of data through cellular data networks will increase exponentially in near future [1]. The impact of this data volume on the operators' networks has been carefully analyzed [3]. It is widely anticipated that severe congestion in the cellular network infrastructure is in the offing if not already happening.

The research community has been responding to this challenge using various means. Moving from macro-cells to femto-cells [9] or automatic offloading traffic to WiFi networks [19] have been widely considered. Operators are adding capacity by employing more spectrally-efficient technologies such as WiMax or LTE, adding more spectrum and macro-cells. But these are very capital intensive processes. Spectrum deregulation is also being considered by policy makers [7]. From a more immediate and practical standpoint, cellular operators have started adding pressure on consumers to reduce traffic load by moving away from flat-rate to usage-based pricing model [2], and more recently, throttling data speeds of high volume users [30]. While such strategies can encourage the consumer to optimize usage, they are ultimately detrimental to widespread adoption of cellular networks by discouraging use of bandwidth hungry applications on mobile devices.

Delayed Scheduling. We consider an alternative approach that can be deployed without any additional capital cost while only minimally hurting – if at all – user experience. Several recent studies have reported that the aggregate cellular network traffic load in a region exhibits a diurnal behavior with peak traffic appearing during mid-day and very low volumes during the night [12], [24]. Individual base station traffic also fluctuates widely during the day [24]. Thus, if certain lower-priority traffic can be deferred from peak times to the off-peak times, the congestion issue can be easily alleviated. Many traffic types are amenable to such deferred scheduling. Examples include large downloads such as apps, e-books, videos/pictures, or sync services such as email or cloud-based data. Often, the originating application type (e.g., p2p) is a sufficient hint that a flow can be deferred. At other times, a hint from the programmer or directly from the user may be needed to decide on such flows. Regardless of such mechanics, it is conceivable that a significant reduction of congestion is possible by such deferral. The operator no longer has to design the network for the peak-demand and/or can accommodate much more traffic than is currently possible, without hurting user experience any significantly.

Model. In this paper, we address various algorithmic problems that arise in the context of the above philosophy of allowing the flows to be deferred. The basic idea is to have a 'deadline' associated with every flow indicating by when the flow should be completed. The deadline provides a way to specify priority or scheduling laxity. It can be specified manually or via a profiling mechanism at the application layer depending on the type and size of the flow and probably other contextual information.

Each flow has an associated location, which determines the set of base stations (BS) that can serve the flow. This exploits the fact that the BSes may have overlapped coverage areas. This is a very reasonable assumption in dense deployments. Recent papers on energy savings (e.g., [12], [13]) exploit this fact to turn down BSes to save energy while providing adequate coverage.

We allow a scheduled flow to be preempted to accommodate other (perhaps, more urgent) flows, and rescheduled as many times as necessary, perhaps at neighboring BSes that also covers the location of the flow. Any form of 'rescheduling,' however, only happens at flow arrival or completion (scheduling epoch), allowing such scheduling to work at a higher layer and *at a much longer time scale than and independent* of the link layer scheduling at the air interface.[1] The actual instantaneous transmission bit rate for the flow could be variable and dependent on the actual radio resource (e.g., bandwidth) allocated at the air interface and the SNR at the mobile client.

Problems Addressed. With the above modeling approach, we address the following problems:

† Stony Brook University, Stony Brook NY
†† Alcatel-Lucent Bell Labs, NJ

[1] The median flow inter-arrival time per BS in the data set we are using (described later) is roughly around 100ms to give the reader an idea about the time-scale.

- Determine the minimum capacity needed for the BSes to schedule all the given flows successfully within their respective deadlines. This problem is addressed in two different contexts: all BSes have the same (uniform) capacity or non-uniform capacities. For the former, we design a polynomial-time optimal algorithm, while for the latter, we show the problem to be NP-hard and design a near-optimal algorithm.
- Given the capacity of BSes, schedule the flows in an *online* manner, so as to maximize the number of flows finished before their deadline. We show the problem to be NP-hard (even, in its *offline* form). Thus, we consider a special (and more pertinent) case of the problem, and design online and semi-online algorithms with provable performance guarantees.

The focus of our paper is to provide efficient solutions for the above problems, under above reasonable modeling assumptions, and to demonstrate potential performance improvements via use of real data (cellular network traces). Encouraged by the results here, our future work will focus on investigating the engineering issues of deploying such mechanisms in a real network.

II. Model, Problem Formulation, and Related Work

In this section, we describe our model of the cellular network and flow arrivals, give a formal description of the problems addressed, and discuss related work.

A. Model

Informally, we address the problem of optimizing peak capacity of cellular base stations (BS), when we have the flexibility of delaying (within certain constraints) the incoming flows. Below, we explain our model of the cellular network, cellular BS, and flows.

Cellular Network and Base Station. A cellular network infrastructure consists of a number of cellular *base stations* (BS) distributed in a two-dimensional geographic region. Each BS is associated with a *coverage region* of arbitrary shape and size.

Base Station Capacity. Each BS is associated with a *capacity*. The capacity is best looked upon as the number of channels available to serve the flows. But, in general, the notion of capacity is some measure of the BS's resources to handle the data demands, e.g., amount of bandwidth or number of channels.[2] Capacities may or may not be uniform across BSes; in this paper, we consider both settings. The optimization objectives considered in our paper are to minimize (i) the uniform capacity, or (ii) the sum of non-uniform capacities.

Flows. Each flow is a sequential stream of data packets, typically semantically related, similar to a TCP or UDP socket connection. For simplicity and clarity, we assume flows and tower capacities to be downlink only; incorporating uplink flows and capacities into your model is straightforward.[3] Put it at the end of "socket connection. Each flow i arrives in the system at a particular time a_i and geographic location l_i, is of a certain size number of bits/packets) s_i, and has a deadline d_i associated with it. The deadline is the time by which the entire flow must be served (as defined below). Note that the deadline value can be used to make a flow "non-deferrable."

Mobility. For simplicity of presentation, we assume that the location l_i remains static (i.e., does not change during its lifetime, even if it is delayed). Our developed techniques easily generalize to the case when the location l_i may change over time, which corresponds to the setting wherein the originating user of the flow is mobile. We discuss generalization of our techniques to mobile users in the end of section III.

Base Station Serving a Flow; Transmission Rates α_{ijt}. A flow i arriving a location l_i can be served by any BS j whose coverage region contains l_i. A BS serves a flow in its coverage region using exactly one unit of its capacity (e.g., one of its channels). We will relax this assumption, i.e., allow a flow to be served using an arbitrary fraction of a BS capacity, towards the end of Section III and IV. Also, the rate at which a flow i is served by BS j at time t is given by the bit-rate parameter α_{ijt}; this parameter essentially captures the variable link quality dependent bit rate of the downlink.[4]

Preemption and Parallelism. To reflect a practical setting, in our model, we allow a flow being served to be *preempted* by another flow. The preempted flow can be resumed later, perhaps, at another BS. Thus, essentially, a flow can be broken into parts and each part served at different BSes[5] at different times. However, we do not permit "parallelism," i.e., different parts of the same flow must be served sequentially.

Completely Served. A flow is considered ***completely served*** if all the parts of it are finished before the deadline.

Model Assumptions and Justifications. We have used some simplifying assumptions to make the problem tractable and to facilitate evaluation over the available network traces which have limited amount of information. We assume that each BS's capacity is constant and independent of the neighboring BSes' capacities. Interference management across BSes is assumed to be perfect (e.g., via prior frequency planning). We assume that the flow size is either known or can be estimated at the flow arrival, and that a flow a continuous stream of packets rather than discontinuous bursts. We do not account for any overhead cost for network controlled hand-offs to move around loads onto different neighboring BSes. But such costs are not hard to account for in the optimization problem. Note that in our schemes such hand-offs only happen at flow arrival or completion times which serve as scheduling epochs.

B. Problem Motivation, Formulations, and Contributions

In the context of the above described model, we consider the following offline and online problems in this paper. The motivation behind the offline problems is to determine optimal

[2]We implicitly assume that a BS's capacity is independent of the load and capacity of the neighboring BSes.

[3]To incorporate uplink flows and capacities: (i) our LP formulation of Section III and IV can be easily changed, and (ii) in Section V, we just need to define and use an additional concept similar to the g-capacity. See Appendix for details.

[4]In particular, the parameter α allows us to model the fact that neighboring towers may take longer to serve a flow than the original tower where the flow arrives.

[5]We assume that a network controlled hand-off (NCHO) [21], [33] mechanism can be used to achieve this.

capacity *needs* based on *historical* traffic information. They can also be used to future traffic growth that can be sustained with the existing network capacities. More importantly, our offline algorithms are also used to estimate "traffic indicators" which are useful in driving the online algorithms.

1) **Minimize Uniform Capacity (MUC).** Consider a cellular network consisting of BSes with given coverage regions, wherein each BS has a uniform capacity. Given the historical data on a set of flows with associated parameters, the MUC problem is to compute the minimum uniform capacity such that all the given flows can be served within their deadlines.
   In Section III, we design a polynomial-time optimal algorithm for the MUC problem.
2) **Minimize Total Capacity (MTC).** Consider a cellular network consisting of BSes with given coverage regions, wherein different BSes may have different capacities. Given the historical data on a set of flows with associated parameters, the MTC problem is to assign capacities to the BSes such that all the given flows can be served within their deadlines, while minimizing the total sum of capacities.
   In Section IV, we show that MTC is NP-hard, and design a polynomial-time near-optimal algorithm for the problem.
3) **Online Scheduling of Flows (OSF).** Consider a cellular network consisting of BSes with given coverage regions and capacities (possibly, non-uniform). At any time instant, a flow may arrive with the associated parameters. The OSF problem is to schedule the flows to BSes in an online manner (i.e., as the flows arrive), while maximizing the number of flows that are completely-served.
   In Section V, we show the above problem to be intractable, and design online and semi-online algorithms for a certain special case of the problem which is more relevant in our context.

C. Related Work

Theoretical Studies. The offline scheduling problems (MUC/MTC) discussed in this paper are similar to the preemptive scheduling problems on identical machines with arrival times and deadlines with the objective of minimizing the number of machines. Although there is a considerable literature on this subject, our model has a key difference: ours is the first preemptive scheduling problem that uses a *constraint* on the set of machines that can schedule a job. The constraint makes a significant difference as many preemptive scheduling problems with various objective functions [28], [31] (including our MTC problem) are polynomial without such a constraint, but can become NP-hard with the constraint.

The other key difference of our addressed problems with the prior literature is our unique objective of minimizing the number of machines that yield a valid schedule. There has been a considerable amount of work on preemptive scheduling with various objectives such as finding a valid schedule [34], and on minimizing makespan [29] [26], number of late jobs [27], lateness [16], job-completion costs [4], etc. However, to the best of our knowledge, there is no work on the objective of minimizing the number of machines for scheduling jobs with arrival times, lengths, and deadlines.

Our online scheduling problem OSF is again a preemptive scheduling problem of jobs with arrival times and deadlines on machines, with the objective of maximizing the number of finished jobs. This problem without the job-machine pairing constraint of our model, has been studied before [5], [14] for a number of different objective functions such as minimizing makespan [11], guaranteed performance [10], etc. However, the constraint on the set of machines a job can use makes our problem much different than the prior-addressed problems.

Load Balancing in Cellular Networks Part of our work is related to the broad topic of load balancing, as we consider "spatial shift" of traffic flows to neighboring BSes that also cover a given flow. This general concept has been widely used at the link layer. For example, see papers on channel assignment, where wireless resources are redistributed rather than traffic [17], [22], [25]. In the same note, myriads of scheduling-based approaches are possible at the link layer [18], [20], [23], [32]. In contrast, our work reflects scheduling at a higher layer, scheduling at the flow level rather than at the packet/frame level. We thus ignore physical/link layer issues such as power, channels, interference and packet scheduling, instead focus on longer term scheduling of flows – either in whole or in part – assuming the capacity at the BS is largely independent of the traffic in the neighboring BSes. Similar load shifting has been used in cellular networks in the context of energy saving [12].

III. Minimizing Uniform Capacity (MUC) Problem

In this section, we address the MUC problem. As mentioned before, the offline MUC problem serves the purpose of determining optimal capacity needs of a network using *historical* traffic information. In our context, we also use our offline algorithms to estimate "traffic indicators" to drive our online algorithm (as described in Section V). We design an algorithm based on a Linear Programing (LP) formulation, and show that it returns an optimal solution.

LP Formulation, and Challenges. To define a linear program for the MUC problem, we need to divide the time into *intervals* (not necessarily of same size), and then, for each interval, determine the mapping that defines which flows are served by each BS. Normally, representation of such a mapping will require use of binary/integer variables, which are not allowed in a *linear* program. However, our model allows preemption of flows at any time instant – which facilitates representation of the mapping, since preemption allows a BS to serve an arbitrary fraction of a flow, within any time interval. In particular, we represent the mapping in terms of the time used by a flow at a BS for each interval. However, even with the above mapping, we still need to represent, for each interval, an actual "schedule" of flows onto BSes that satisfies the constraints of "non-parallelism." We will show (through Lemma 4) that if intervals and linear constraints are chosen appropriately, then the existence of such a schedule can be guaranteed. In particular, we define the *intervals* as the time intervals (of

possibly different sizes) between time instants of interest (i.e., arrival time or deadline of a flow), as formally defined below; the number of such intervals is polynomial in size of a given MUC problem. The set of equations in our LP formulation are defined as follows.

Variables. Based on the above observations, we define the following notations and variables, for our LP formulation.

- $T = \{T_1, T_2, \ldots, \}$ is the finite set of time "instants", where T_t is either an arrival time or a deadline of one of the given flows. We assume T_t's to be in increasing order; thus, $T_t < T_{t+1}$ for all t.
- Variables i, j, t to denote a flow, a BS, and a time instant, respectively.
- Variable k to denote the uniform capacity of BSes.
- Variable x_{ijt} to denote the amount of time the flow i is served by BS j, during the time interval T_t to T_{t+1}.

Equations. On the above variables, we define the following equations:

1) $x_{ijt} = 0$ for all t, and for all i, j, where the location l_i of flow i is *not* in the coverage region of cell j.
2) $x_{ijt} = 0$ for all j, and for all i, t where $T_t < a_i$ (the arrival time) or $T_t \geq d_i$ (the deadline).
3) $\sum_{t,j} \alpha_{ijt} x_{ijt} = s_i$, for all i, where α_{ijt} is the given bit-rate (a constant in the LP) and s_i is the size of the i flow. This represents the constraint that all the flows must be completely served.
4) $\sum_i x_{ijt} \leq k(T_{t+1} - T_t)$, for all j, t. This represents the constraint that the capacity of each BS j is at most k. Note that the values T_{t+1} and T_t are constants.
5) $\sum_j x_{ijt} \leq (T_{t+1} - T_t)$, for all i, t. This equation *attempts* to "represent" the non-parallelism constraint, i.e., (a) no flow is served by two different BSes at the same time, and that (b) a flow is served using exactly a unit capacity of a BS. We will show that this equation is sufficient to "represent" the above two constraints, as shown in Lemma 4.
6) **Objective.** Minimize k.

Integral BS Capacities. Note that the above LP returns a real number for the uniform capacity variable k. For example, for the simple MUC instance where there is only one BS and a single flow of size 1 with a deadline of 2, the above LP would return the value of k as 1/2. However, when a flow is served by a unit-capacity only, a BS capacity of 1/2 is not sufficient. In particular, we need to return an integral value for the BS capacity (e.g., when the capacity signifies number of channels and each flow is served by a channel). Thus, our overall algorithm for the MUC problem, called the *LP-based algorithm*, is to return $\lceil \mathrm{K} \rceil$ as the final value, where K is the objective value returned by the above LP. We now prove the correctness and optimality of this algorithm.

Proof of Correctness and Optimality.

Lemma 1: The solution ($\lceil K \rceil$) returned by the above LP-based algorithm is a "valid" MUC solution, i.e., using a uniform BS capacity of $\lceil K \rceil$, it is possible to completely-serve all flows.

PROOF: To prove the lemma, we need to show that an assignment of values to the LP variables that satisfies all the LP equations has a corresponding "schedule" of flows onto BSes (i.e., a function that maps BSes to a set of flows being served *for each time instant*) satisfying all the constraints of our model of serving a flow at a BS. In essence, we need to prove the following claim:

For any given t, the x_{ijt} values of an LP solution can be converted to a schedule of flows onto BSes such that at any time instant*: (a) each BS is using exactly a unit of capacity to serve a flow, and (b) a flow is not being served by multiple BSes.*

Once we prove the above claim, it is easy to see that the proof of the lemma follows. The proof of the claim is tedious and rather non-trivial, and hence, deferred to the Appendix (see Lemma 4). ■

Lemma 2: Given a cellular network with BSes with varying capacities and flows. Let the capacity of BS j be k_j. Consider a time interval $[0, T]$. We are given real values $\{x_{ij}\}$ for each flow i and BS j, signifying that the flow i must be served by BS j for x_{ij} time. The $\{x_{ij}\}$ values are such that (a) for each BS j, $\sum_i x_{ij} \leq k_j T$, and (b) for each flow i, $\sum_j x_{ij} \leq T$. We claim that there is a schedule of flows onto the BSes (i.e., mapping of BSes to flows, for each time instant) such that at any time instant: (i) A BS uses exactly a unit capacity to serve a flow, and (ii) Each flow is served by at most one BS. ■

Theorem 1: The solution ($\lceil \mathrm{K} \rceil$) returned by the above LP-based algorithm is an optimal solution of the given MUC problem.

PROOF: It is easy to see that an optimal solution to the MUC problem satisfies all the equations of our LP formulation. Now, since the LP formulation returns a solution with a minimum value of k, and hence, with a minimum value of $\lceil k \rceil$, the theorem follows. ■

Using Non-Unit Capacity to Serve a Flow. We can easily generalize our model and techniques to the case wherein a flow can be served using an arbitrary fraction of a BS's capacity. Such a model depicts the situation wherein the BS's capacity signifies the size of the available downlink bandwidth, and a flow can be served using any fraction of this bandwidth. To generalize our algorithm to allow use of an arbitrary fraction of BS's capacity to serve a flow, we make the following changes to our LP formulation:

- We let the variable x_{ijt} signify the total amount of BS i's resources (fractional-capacity times allocated-amount-of-time) used to serve flow j in the t^{th} interval.
- We change the 5^{th} equation to:

$$\text{for all } i, t, \ \sum_j (x_{ijt}/k) \leq (T_{t+1} - T_t).$$

Note that in the above model, the BS capacity can be an arbitrary real number. We can easily generalize Lemma 4 for the above model, which proves the optimality of the solution delivered by the above modified LP, when we allow an arbitrary fraction of a BS's capacity to serve a flow.

Bounding the Capacity. In the above model, we can also bound the amount of capacity that can be used to serve a flow, e.g., a bound on the number of channels that can be used to serve

a flow. If c is such an upper bound on the amount of capacity, then we change the 5^{th} equation to:

$$\text{for all } i,t,\ \sum_j (x_{ijt}/c) \leq (T_{t+1} - T_t).$$

However, we need to change the LP solution K to the next multiple of c, i.e., return $c\lceil K/c \rceil$ as the final solution, for the proof of Lemma 4 to work. This introduces an *additive* approximation factor of c to the solution delivered by the LP-based algorithm, i.e., if $|\text{OPT}|$ is the optimal BS capacity then the solution returned by the above LP-based algorithm is at most $|\text{OPT}| + c$.

Handling Mobility. Note that mobility of users can be modeled by defining the location attribute l_i associated with the flows to be varying over time. Thus, the location attribute is better represented as l_{it} for each time instant t. The above can be incorporated in our LP formulation as follows:

- Firstly, the time instants of interest will now also include time instants when a location of a flow crosses the boundary of a BS's coverage region.
- Secondly, due to the mobility, the set of BSes, whose coverage region contains the flow's location, changes over time. This can be incorporated by defining the first set of equations of LP appropriately, i.e., constraining x_{ijt} to be zero for every i, j, t where the location of i at time t is not contained in the coverage region of j.

With the above two changes, it is easy to see that the optimality claim of LP-based algorithm can be extended to the MUC problem with mobility.

IV. Minimizing Total Capacity (MTC) Problem

In this section, we address the MTC problem. As mentioned for the MUC problem, the purpose of an offline algorithm is to optimal the capacity needs of a cellular networks, based on historical traffic information. We start with showing that the MTC problem is NP-hard, and then modify the LP from the previous section to design a near-optimal algorithm for the MTC problem.

Theorem 2: MTC Problem is NP-Hard. ■

PROOF: Consider an instance of the disk-set-cover problem, where we are given points and fixed unit-disks in a Euclidean plane, and the problem is to select a minimum number of disks that cover all the given points [8].

Given an instance of a disk-set-cover, we construct an instance of our MTC problem as follows. For each disk, we construct a cell with its coverage region as the disk. For each point, we construct a flow at the point's location, with arrival time as 0, size as 1, and deadline as n, where n is the total number of points in the discrete unit disk cover instance (and hence, the number of flows in the constructed instance of MTC). Thus, all the flows in the system have the same arrival time, deadline, and size. Now, it is easy to see that any BS can serve all the flows in its coverage region using only a unit-capacity. Thus, the optimal solution of MTC assigns a capacity of either 0 or 1 to the BSes. It is now easy to verify that the optimal solution of MTC yields an optimal solution of the original discrete unit disk cover instance. Thus, the MTC problem is NP-hard. ■

Near-Optimal Algorithm For MTC Problem. Our approximation algorithm is based on the LP formulation from the previous section. We use the same variables and notations from the LP of the previous section, except that we use $\{k_1, k_2, \ldots, \}$ to denote the capacities of the various BSes, i.e., k_j is the capacity of the j^{th} BS. In our below LP formulation for MTC, the non-optimality of the LP solution comes from the fact that the capacity variables are treated as real numbers by the LP (when they are in fact positive integers).

The LP formulation for the MTC problem consists of the same equations as the LP in the previous section for the MUC problem, except that the fourth and sixth set of equations are changed to the following respectively.

4) $\sum_i x_{ijt} = k_j(T_{t+1} - T_t)$, for all j, t.
6) **Objective.** Minimize $\sum_j k_j$.

The above LP returns a solution with real values for the $\{k_j\}$ variables. We take a ceiling of these values to yield integral values for k_j, and return that as the solution for MTC. Thus, if $\{K_j\}$ are the values returned by the LP, we return $\{\lceil K_j \rceil\}$ as the final MTC solution. Below, we show that the solution $\{\lceil K_j \rceil\}$ is such that $\sum_j \lceil K_j \rceil \leq (|\text{OPT}| + J)$, where $|\text{OPT}|$ is the optimal total capacity and J is the total number of BSes in the given problem instance.

Proof of Correctness and Near-Optimality. The following lemma, whose proof is similar to the proof of Lemma 1, states that the above LP-based algorithm delivers a "valid" solution to a given MTC problem.

Lemma 3: The solution, $\{\lceil K_j \rceil\}$, returned by the above LP-based algorithm is a "valid" MTC solution, i.e., using the BS capacities $\{\lceil K_j \rceil\}$, it is possible to serve all flows within their deadlines. ■

Theorem 3: The solution, $\{\lceil K_j \rceil\}$, returned by the above LP-based algorithm is such that the sum of capacities $\sum_j \lceil K_j \rceil$ is at most $|\text{OPT}| + J$, where $|\text{OPT}|$ is the optimal sum of capacities and J is the number of BSes in the input.

PROOF: It is easy to see that any valid solution of the MTC problem satisfies the equations of the LP formulation. Thus, $|\text{OPT}|$, the value of the optimal solution to the MTC problem is more than $\sum_j K_j$, the value of the optimal LP solution. Since, $\sum_j \lceil K_j \rceil \leq (\sum_j K_j) + J$, we have $\sum_j \lceil K_j \rceil \leq |\text{OPT}| + J$. ■

Corollary 1: If $|\text{OPT}|$ is at least J, i.e., if each BS uses at least a unit capacity (in other words, no BS is turned off), then the above LP-based algorithm for the MTC problem is 2-approximate.

Using Non-Unit Capacity to Serve a Flow; Mobility. As in the previous section, we can generalize our techniques to allow use of fractional capacity of a BS to serve a flow. However, in the case of the MTC problem, we *need* to use a bound c on the units of capacity that can be used to serve a flow in order to maintain the linearity of our LP program. Thus, as before, we let x_{ijt} signify the total amount of resources used in t^{th} interval, and change the 5^{th} equation to:

$$\text{for all } i,t,\ \sum_j (x_{ijt}/c) \leq (T_{t+1} - T_t).$$

Then, if $\{K_j\}$ is the solution of the LP program, then we return $\{c\lceil K_j/c\rceil\}$ as the solution of the MTC problem. With the above change, we can show that the total capacity of the modified-LP is at most $|\mathrm{OPT}| + cJ$, where $|\mathrm{OPT}|$ is the optimal total capacity and J is the number of BSes in the input.

Finally, mobility can be handled for the case of MTC problem in the similar way as was done for the case of MUC problem.

V. Online Scheduling of Flows

In this section, we consider the online version of our problem, i.e., given a cellular network, we want to schedule the arriving flows onto BSes, so as to maximize the number of flows that are completely served. We prove that this problem is NP-hard, and consider the special case of the problem in which the input is such that all the flows *can* be completely served. For this special case of the problem, we design various semi-online and online algorithms, and prove appropriate performance guarantees. *For sake of clarity,* we assume a BS uses a unit-capacity to serve a flow. We discuss relaxation of this assumption towards the end of the section.

Online Scheduling of Flows (OSF). Consider a cellular network consisting of BSes with given capacities and coverage-regions. At any instant, a new flow with an associated size and deadline may arrive at a location. The OSF problem is to schedule the arriving flows in an online manner, so that the number of completely-served flows is maximized. Recall that a flow is considered completely-served if it finishes completely before its deadline, and note that the schedule delivered by an online algorithm may not completely-serve all the flows.

We claim that the OSF problem is intractable. In fact, even the offline version of the problem can be shown to be NP-hard, as the below theorem states. Proof is differed to Appendix.

Theorem 4: Given a cellular network with possibly non-uniform BS capacities, and flows, the problem of determining (even offline) a schedule of flows onto BSes so as to maximize the number of completely-served flows is NP-hard. ■

Online Scheduling of Completely-Servable Flows (OSCF). Since the above OSF problem is NP-hard, we consider the OSF problem wherein we restrict ourselves to "completely-servable" instances. A ***completely-servable*** instance of an OSF problem is an instance for which there *exists* a schedule of given flows onto BSes such that all the flows are completely-served. We refer to this restricted version as the *OSCF problem*. We can easily modify our LP formulation to deliver a schedule that completely-serves all flows of a completely-servable instance. However, we have shown through a counter example that there is no optimal online-algorithm possible for the OSCF problem. See below theorem.

Proposition 1: The offline version of the OSCF problem can be solved optimally in polynomial time.

Theorem 5: There is no *online* algorithm for the OSCF problem that, for every input instance, generates a schedule that completely-serves all the flows.

PROOF: We prove the theorem using a counter example. Consider a network with two BSes each of unit capacity. We assume the bit-rates to be uniformly unit. We represent a flow i as (a_i, s_i, d_i, R_i), where a_i is the arrival time, s_i is the size, d_i is the deadline, and R_i is the set of BSes where the flow can be scheduled (R_i essentially represents the location of the flow with respect to the coverage-regions of the BSes).

At $t = 0$, three flows arrive in the system $(0, 1, 3, \{1\}), (0, 1, 3, \{2\})$, and $(0, 2, 2, \{1, 2\})$. For this instance, the third flow *must* be scheduled immediately at $t = 0$. Since the problem (till now) is symmetric with respect to the BSes, we can assume without loss of generality that the online algorithm schedules the third flow on the first BS. Since the second BS is free, we schedule the second flow on the second BS at $t = 0$; this can not hurt the algorithm.

Now, at $t = 1$, if the fourth flow $(1, 2, 3, \{1\})$ arrives, then either this or the third flow will never be completely served. But, this instance is certainly a completely-servable instance, since the first and fourth flows could have been scheduled on the first BS, and the second and third flows on the second BS. ■

In the following subsection, we design a semi-online algorithm that solves the OSCF optimally when aided with appropriate statistics on historical traffic pattern and slightly additional capacity. We also present a purely-online heuristic that is optimal for non-overlapping coverage regions.

A. Semi-Online and Online Algorithms

In this section, we start with designing a *semi-online* algorithm for the OSCF problem, which is aided by appropriate statistics on historical traffic patterns. We will show that our semi-online algorithm solves the OSCF problem optimally if it is allowed to use certain additional capacity (depending on the variations in traffic patterns) than that used by the optimal offline algorithm. We also design a purely online heuristic. We start with a few definitions.

Covering BS; Remaining Size s_{it}. For a flow i, a BS j is said to be its *covering BS* if j's coverage region contains l_i, the location of the flow i.

For a flow i, the *remaining size* at a time instant is denoted by s_{it} and is defined as the size of the *remaining* (i.e., not yet served) part of the flow i. More formally, in terms of notations of previous two sections, $s_{it} = s_i - \sum_j \sum_{t'<t} x_{ijt'}$.

Allowable Delay w_{it}. At a time instant t, the *allowable delay* w_{ij} of a flow i is the maximum amount of time by which the remaining part of i can be delayed, while being completely-served. More formally, $w_{it} = (d_i - t) - s_{it}/\alpha_{it}$, where d_i is the deadline, s_{it} is the remaining size at t, and $\alpha_{it} = \min_j \alpha_{ijt}$ (the minimum bit-rate across BSes).

Subregions r_m. Given a 2D network region, we define a *subregion* r_m as the set of points in the 2D plane that lie within the same set of coverage regions. Note that the set of subregions are disjoint. See Figure 1. For circular coverage-regions, it is easy to show that the total number of subregions is $O(n^2)$ where n is the number of BSes [15].

Traffic Indicators $f_p(r_m, t)$**.** Consider a cellular network and a set of completely-servable historical instances of flows $\{M_1, M_2, \ldots, \}$. Each M_p is essentially a set of flows with associated parameters, such that the flows can be completely-served by some schedule S_p. For a subregion r_m, time instant

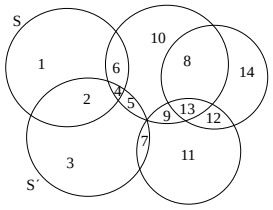


Fig. 1. Subregions. Here, five BSes with circular coverage-regions give rise to 13 subregions.

t, and an instance M_p, we define the *traffic indicator* $f_p(r_m, t)$ as the number of flows located in r_m that are being served by a BS at time t, in a schedule S_p of M_p that completely-serves all flows. Note that S_p can be computed in polynomial-time by Proposition 1.

Semi-Online Global (SOG) Algorithm. Given cellular network and a set of completely-servable historical instances $\{M_1, M_2, \ldots, \}$, let $f_p(r_m, t)$ be the traffic indicators as defined above. Let

$$g(r_m, t) = max_p \quad f_p(r_m, t).$$

The *Semi-Online Global (SOG)* algorithm uses the above g values to schedule a given online instance of flows as follows.

At each time instant t, for each subregion r_m, pick $g(r_m, t)$ (or less, if not available) flows (with non-zero remaining size) located in r_m with least allowable delays. Let this set of flows for the entire network be S. Find the largest subset S' of S that can be scheduled onto BSes at time t; this can be done by finding the maximum-matching problem between S and "servers," where a BS of capacity k is represented by k servers. Finally, we can also add more flows (that are not in S) to schedule, if possible.

Performance of SOG Algorithm. We now show that the SOG algorithm solves the OSCF problem optimally when aided with slightly additional capacity, which depends upon the variation of f values across historical instances and the deviation of the input instance from the historical instances. We start with a definition.

Definition 1: (g-Capacities.) For a given cellular network and set of historical instances of flows $\{M_1, M_2, \ldots, \}$, we define the g-capacity for each BS as follows. On the given cellular network, consider the following instance of flows: For each subregion r_m and time instant t, we create $g(r_m, t)$ flows of size 1, arrival time t, and deadline $t+1$. We solve this MTC problem using the LP-based algorithm, and call the resulting BS capacities as the *g-capacities* of the BSes. □

Theorem 6: Given a cellular network and set of historical instances of flows $\{M_p\}$. The SOG algorithm would completely-serve all the flows of any instance M_p, if it uses the g-capacities for the BSes.

PROOF: Consider an instance M_p. Now, by definition of f values, there exists a schedule S_p that schedules $f_p(r_m, t)$ flows (onto some BSes) from each subregion r_m at each instant t. Using an exchange argument, we can assume that the $f_p(r_m, t)$ flows being scheduled are the ones with the least allowable delay, for each r_m and t. Since SOG uses the g-capacities, it has sufficient capacity to schedule $g(r_m, t)$ flows for each r_m and t. Since $g(r_m, t) \geq f(r_m, t)$, the theorem follows. ■

By the above theorem, note that if we had only one historical instance M_1, then the SOG algorithm can completely-serve all the flows in M_1 using at most $n + O$ total network capacity, where O is the optimal capacity needed and n is the number of BSes in the network. This is because $g(r_m, t) = f_1(r_m, t)$ for all r_m, t, and hence, the sum of g-capacities is at most $n + O$ by Theorem 3. *In general, if (i) there is minimal deviation in $f_p(r_m, t)$ values across the given historical instances, and (ii) the given input to SOG is "similar-enough" to the historical instances, then SOG will completely-serve all the flows of the given input with minimal additional capacity compared to the optimal required.*

Semi-Online Localized (SOL) Algorithm. The above SOG algorithm is not localized, since it needs to solve the matching problem at each time instant. We can also consider the simpler *Semi-Online Localized (SOL)* algorithm that instead computes a *maximal* matching greedily, which can be done in a localized manner. Note that any maximal matching is of size at least half of the maximum matching.

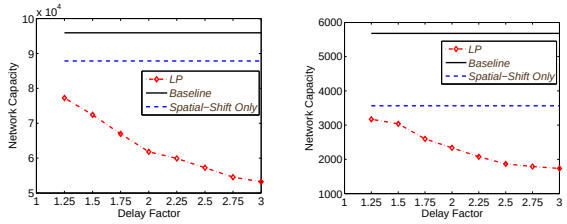
Purely-Online (PO) Heuristic. The above semi-online algorithms are aided by the g values computed from historical data, and hence, is not purely online algorithm. Below, we present a purely-online (PO) heuristic, which is motivated by the fact that it is optimal for networks with non-overlapping coverage-regions.

Heuristic Description. At a high-level, the PO heuristic does the following at each time instants of "interest." It orders the flows (with non-zero remaining size) in increasing order of their allowable delays at that instant. Then, it tries to "match" these flows with BSes greedily, as described in detail below. The time instants of interest (at which the above is done) are: (i) arrival of a new flow, (ii) completion of a flow, and (iii) passing of the deadline of a non-scheduled flow.

To match flows onto BSes, we consider the flows in the increasing order of their allowable delays, and try to schedule each flow i as follows. **If** there is a covering BS j of i that is free, then we schedule i on j. **Else**, we try free up some serving BS of i, by finding an "alternating path" and "shuffle" some flows along this path as follows. We find an alternating sequence of flows and BSes $(x_1, y_1, x_2, y_2, \ldots, x_m, y_m)$ such that (i) each x_q is a flow and each y_q is a BS, (ii) $x_1 = i$ and y_1 is a covering BS of x_1, (iii) for all $q \geq 1$, y_q is currently serving x_{q+1}, (iv) y_m is a covering BS of x_m, and (v) y_m is currently free (i.e., not serving any data request). If such an alternating path exists, then we stop serving all the flows $\{x_2, x_3, \ldots, x_m\}$, and reschedule each flow x_q to y_q. The above is thus able to schedule $x_1 = i$ on y_1, if an alternating path as described above exists. If there is no alternating path, then we do *not* schedule i at this point and consider the next flow in order.

As mentioned before, the motivation and intuition behind the above PO heuristic is that it can be shown to be optimal for the special case of the OSCF problem when the coverage regions of the BSes are disjoint. However, the non-optimality of the heuristic for the general case of arbitrarily overlapping coverage regions cannot be bounded.

Theorem 7: The above PO heuristic solves the OSCF problem optimally, if the coverage regions of the BSes are disjoint.



(a) Uniform Tower Capacities. (b) Non-uniform Tower Capacities.

Fig. 2. Total capacity requirements for varying delay factors, for various offline algorithms.

PROOF: Note that due to disjoint coverage regions, it is sufficient to prove the theorem for the case of a single BS. For the single BS, the PO heuristic essentially schedules flows in the order of allowable delays. Now, consider the schedule of an offline algorithm that schedules all the flows within their deadlines; such a schedule exists by definition of the OSCF problem. In such a schedule, we can show that if there is a time instant wherein a flow with higher allowable-delay is scheduled before a flow with a lower allowable-delay, then "exchanging" them in the offline schedule still ensures that all flows are scheduled within their deadlines. ■

Using Non-Unit Capacity to Serve a Flow. We now discuss how our techniques of this section can be extended to allow non-unit capacities to serve a flow. If c is the bound on the number of units of capacities of a BS that can be used to serve a flow, then we make the following changes: (i) Extend the definition of allowable delay as follows: $w_{it} = (d_i - t) - s_{it}/(c\alpha_{it})$. (ii) Make $\lceil c \rceil$ copies of each flow in S, when solving the matching problem for SOG and SOL. (iii) Allow multiple units of flow to be scheduled simultaneously (up to the bound of using c units of BS capacity at a time) at each BS, in the PO Heuristic.

VI. Simulations

In this section, we analyze the performance of our various algorithms on real cellular network data set collected at the core of a commercially operated 2G/3G network. The data used in the evaluation consists of flow level (UDP or TCP flows) statistics collected for 101 base stations for a one week period in 2007. The region covered is about 80 square km spanning both dense urban and suburban areas. There are about 1 million flows in the data set considered.[6] In the data set, each flow has an arrival time, BS-id where it is served, and a flow size. We assume that the flows to be served at a constant rate equal to its size divided by its duration.[7]

Since obviously there is no delayed scheduling in this network, the flows are immediately served with no delay. Flows that are served are recorded in the data set. Flows that could not be served due to unavailability of network resources do not have any record.

Using a simulation model with this data set, we will show that (i) there is a considerable reduction in capacity requirements, when flows are allowed to be delayed even by a small factor, and (ii) with slightly additional capacity than the near-optimal network capacity (computed by the offline LP-based algorithms), our online semi-online algorithms are able to completely-serve all the flows.

Flow Locations, Coverage Regions, Delay-Factors. Since our data set neither includes the exact flow locations nor the BS coverage regions, we determine the set of covering BSes for a flow as follows: (i) We construct the Voronoi diagram over the BS locations; (ii) If a flow i arrives at a BS j, then we assume that it can be served by any BS j' whose Voronoi region is adjacent to that to j.[8] Essentially, a flow i arriving at BS j can be served by j and any of its "neighboring" BSes. Defining the serving BSes of a flow in the above manner precludes the need to artificially generate flow-locations (which are absent from the network trace).

Also, the flows in our data do not have deadlines associated with them. So, we run simulations for various "delay-factors" and define deadlines based on the delay-factor; in particular, for a *delay-factor* of c, the deadline of a flow i is computed to be $a_i + c\alpha_{ijt}s_i = a_i + cs_i$ where a_i and s_i are the given arrival time and size respectively and α_{ijt} is assumed to be 1 (for lack of record of low-level parameter values in our data).

LP-Based Offline Algorithms: Delay Factor vs. Capacity Requirements. We start with analyzing the effect of deferring flows on the capacity needs of the network. Thus, in Figure 2, for varying delay-factor, we plot the total network capacity required to completely-serve all the flows as computed by our LP-based algorithms. We consider both cases, viz., the uniform as well as non-uniform BS capacities, and use all 7 days of data.[9] In the graph, for comparison purposes, we also plot two values corresponding to the case of delay-factor of one (i.e, no-delays): (i) capacity requirements when each flow can be served only by the BS it arrived at as recoded in the data set, and (ii) capacity requirements when a flow can be served by multiple BSes (as determined by Voronoi tessellation described above). We refer to these values as `Baseline` and `Spatial-Shift Only` respectively.

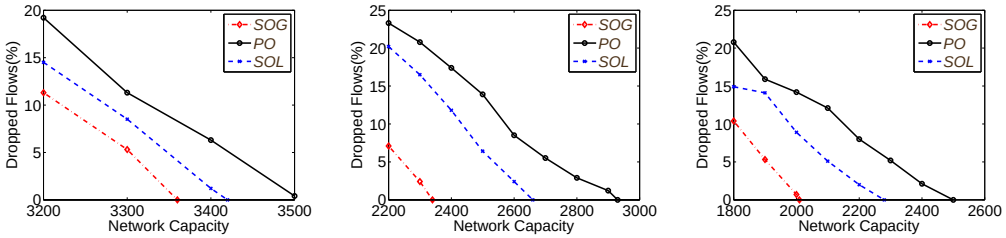
As expected, the capacity requirements decrease with the increase in the delay-factor. The decrease is substantial even with minimal delays. For example, even with a delay factor of 1.25 (flows can be delayed only up to one-fourth of their size), the capacity requirements reduce by about 50% for the non-uniform case and about 20% for the uniform case. This

[6]For proprietary reason we are unable to provide further details about the network. We believe that the missing details will not hurt the readers' understanding of our work.

[7]Technically, we are given flow duration, how long within this duration the flow was inactive (i,e., no resource scheduled) and number of bytes served for this flow. However, it is impossible to decipher from this aggregated information: (i) when the flow was descheduled either for resource limitations or plain inactivity (in 3GPP standard the resources can be scheduled even when a flow is dormant, until an inactivity timer fires), (ii) what bit rates the flow was served with when it was indeed scheduled.

[8]In our data set, the distribution of BSes is roughly uniform: more than 70% of the geographically closest neighbors are within 1 to 2 miles. Thus, coverage assumption based on Voronoi is not too different from a coverage assumption based on the geographic distance.

[9]To solve the LP program over 1 million flows and about 100 BSes in a reasonable amount of time, we employed the following divide-and-conquer strategy: we divided the data into appropriate 3-4 hour durations, computed the LP solution for each input independently, and then "combined" the solutions to get a *valid* (but, perhaps, suboptimal) LP solution. Thus, the network capacity numbers for LP in Figure 2 and 4 are perhaps a slight overestimate of the best possible LP-based solution.



(a) Delay-Factor of 1.25. (b) Delay-Factor of 2.25. (c) Delay-Factor of 3.

Fig. 3. Percentage of dropped flows for varying total network capacity, for various semi-online/online algorithms. Here, the starting network capacity value is the near-optimal capacity requirement as computed by the LP-based algorithm.

changes to about factors of 3 and 2 respectively for a delay factor of 3 (i.e., flows can be delayed up to twice their size). Note that a sizable reduction comes from the ability of "load balancing" via scheduling on less-loaded neighboring BSes.

Semi-Online and Online Algorithms: Capacity Requirements vs. Percentage of Flows Dropped. In this set of plots, we run various semi-online/online algorithms, viz., (i) Semi-Online Global (SOG), (ii) Semi-Online Localized (SOL), and (iii) Purely-Online (PO). We run the above algorithms over Wednesday's data (the results were similar for other weekdays), while using the remaining four weekdays of data to compute the statistical g values used by the semi-online algorithms. For increasing total network capacity, we plot the number of flows that were not completely-served (i.e., dropped). See Figure 3. Here, we vary the network capacity by proportionally increasing all the BS capacities starting *from* (i) the value of the near-optimal LP-based solution (for the non-uniform case) for the given flows, *till* (ii) the algorithm is able to completely-serve all flows. We ran the algorithms for three different delay-factors. We observe that both the traffic indicators as well as the global-matching scheme have considerable impacts on reduction of capacity requirements. In particular, SOG uses very minimal additional capacity (about 5% more) over the offline capacity needs, to completely-serve all the flows. Even with the same capacity as the offline capacity, the drops are marginal (less than 10%).

g-capacities vs. Near-Optimal Capacities. In Figure 4, we show for various delay-factors: (i) the near-optimal (LP) network capacity needed to completely-serve all seven days of flows, and (ii) the g-capacity of the network, based on all seven days of data. We observe that the g-capacity of the network is only slightly higher (only about 5-15%) than the offline capacity, which suggests that SOG needs only slightly higher network capacity to completely-serve all the flows in any of the historical instances. In effect, this simulation result shows that the variation of $f_p(r_m, t)$ values across the historical instances is minimal enough that the inefficiency introduced by semi-online processing of flows as done in SOG is minimal compare to the near-optimal offline processing.

VII. Conclusions

In this paper, we have considered flow deferral in a cellular data network for more efficient scheduling. Essentially, this technique shifts flows temporally from peak to off-peak periods taking advantage in the inherent variability in traffic

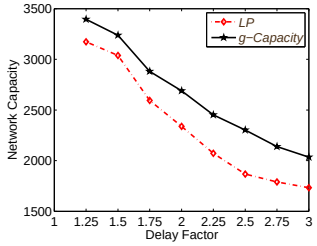


Fig. 4. The g-capacity of the network, and the near-optimal offline capacity requirement, for varying delay-factors.

loads at the base stations. We have developed efficient algorithmic techniques to show that this can reduce network capacity requirements with only modest delays that appropriate delay-tolerant applications (e.g., p2p downloads or background synchs) will be able to withstand. Since our work focuses on algorithms and their efficiencies, we have used modeling abstractions that use several simplifying assumptions, such as BS capacity independent of neighboring BSes, ignoring any form of scheduling or network-controlled handoff overheads, advance knowledge of flow size at flow arrival, availability of historical data, etc. Our goal has been to validate whether such deferral approaches are at all effective with realistic traffic loads. The results do indicate that tremendous cost savings (in terms of network capacity requirements) are indeed possible with only modest delays, with savings increasing with increase in allowable delays. This is certainly encouraging and will motivate wireless carriers to incentivize mobile users to tag deferrable flows either automatically or manually to alleviate network congestions. Our future work will consider eliminating some of the above assumptions and develop practical engineering mechanisms to achieve deferred scheduling. Of particular interest to us is extending our techniques to the case wherein the flow size is not known a priori.

References

[1] Cisco visual networking index: Global mobile data traffic forecast update, 2009-2014. tinyurl.com/d887fkswww.
[2] Verizon to introduce tiered pricing for iPhone.
[3] The impact of mobile computers and smartphones on cdma 2000 networks. www.signalsreseaerch.com, Jan 2011.
[4] K.R. Baker, E.L. Lawler, J.K. Lenstra, and A.H.G. Rinnooy Kan. Preemptive scheduling of a single machine to minimize maximum cost subject to release dates and precedence constraints. *Operations Research*, 1983.
[5] P. Baptiste. An $o(n^4)$ algorithm for preemptive scheduling of a single machine to minimize the number of late jobs. *Operations Research Letters*, 1999.
[6] H. Bonnimann and M. Goodrich. Almost optimal set covers in finite vc-dimension. *Discrete Computational Geometry*, 14, 1995.

[7] M. M. Buddhikot and K. Ryan. Spectrum management in coordinated dynamic spectrum access based cellular networks. In *DySPAN*, 2005.
[8] Paz Carmi, Matthew J. Katz, and Nissan Lev-Tov. Covering points by unit disks of fixed location. In *Proceedings of the 18th international conference on Algorithms and computation*, ISAAC'07, pages 644–655, Berlin, Heidelberg, 2007. Springer-Verlag.
[9] H. Claussen, L. T. W. Ho, and L. G. Samuel. An overview of the femtocell concept. *Bell Lab. Tech. J.*, 13, 2008.
[10] B. DasGupta and M. Palis. Online real-time preemptive scheduling of jobs with deadlines. In *LNCS*. Springer, 2000.
[11] Bo Chen et al. An optimal algorithm for preemptive on-line scheduling. *Operations Research Letters*, 18(3):127 – 131, 1995.
[12] Chunyi Peng et al. Traffic-driven power saving in operational 3g cellular networks. In *Proc. ACM Mobicom*, 2011.
[13] G. Fusco et al. Finding green spots and turning the spectrum dial: Novel techniques for green mobile wireless networks. In *IEEE DySPAN*, 2011.
[14] Grsel A. Ser et al. Minimizing the number of tardy jobs in identical machine scheduling. *Computers & Industrial Engineering*, 25(1-4), 1993.
[15] Himanshu Gupta et al. Connected sensor cover: self-organization of sensor networks for efficient query execution. *IEEE/ACM Trans. Netw.*, 14, 2006.
[16] J. Labetoulle et al. Preemptive scheduling of uniform machines subject to release dates. *Progress in combinatorial optimization*, 1984.
[17] J. V. Leeuwaarden et al. Load balancing in cellular networks using first policy iteration. Technical report, Helsinki University of Technology, 2001.
[18] J.-W. Lee et al. Joint resource allocation and base-station assignment for the downlink in cdma networks. *IEEE/ACM Transactions on Networking*, 14(1), feb. 2006.
[19] Kyunghan Lee et al. Mobile data offloading: how much can WiFi deliver? In *ACM SIGCOMM 2010*, 2010.
[20] Lin Du et al. Intelligent cellular network load balancing using a cooperative negotiation approach. In *IEEE WCNC*, volume 3, 2003.
[21] Nasif Ekiz et al. An overview of handoff techniques in cellular networks. *World Academy of Sci., Eng. and Technology*, 2005.
[22] Sajal K. Das et al. A dynamic load balancing strategy for channel assignment using selective borrowing in cellular mobile environment. *Wirel. Netw.*, 3, Oct. 1997.
[23] Samir Das et al. Dynamic load balancing through coordinated scheduling in packet data systems. In *INFOCOM*, 2003.
[24] Utpal Paul et al. Understanding traffic dynamics in cellular data networks. In *INFOCOM*, april 2011.
[25] D. Everitt and D. Manfield. Performance analysis of cellular mobile communication systems with dynamic channel assignment. *IEEE J. on Selected Areas in Communications*, 1989.
[26] T. Gonzalez and S. Sartaj. Open shop scheduling to minimize finish time. *J. ACM*, 1976.
[27] E. Lawler. A dynamic programming algorithm for preemptive scheduling of a single machine to minimize the number of late jobs. *Annals of Operations Research*, 26, 1990.
[28] J. Leung. *Handbook of Scheduling: Algorithms, Models, and Performance Analysis*. CRC Press, 2004.
[29] Z. Liu and E. Sanlaville. Preemptive scheduling with variable profile, precedence constraints and due dates. *Discrete Applied Math.*, 1995.
[30] NY Times. Limiting the unlimited. Issue of March 2, 2012.
[31] M. L. Pinedo. *Scheduling: Theory, Algorithms, and Systems*. Springer Publishing Company, Incorporated, 3rd edition, 2008.
[32] O.K. Tonguz and E. Yanmaz. The mathematical theory of dynamic load balancing in cellular networks. *IEEE TMC*, 2008.
[33] N.D. Tripathi, J.H. Reed, and H.F. VanLandinoham. Handoff in cellular systems. *IEEE Personal Communications*, 5(6):26 –37, 1998.
[34] W. Zhao, K. Ramamritham, and J. A. Stankovic. Preemptive scheduling under time and resource constraints. *IEEE Trans. on Computers*, 1987.

VIII. Appendix

Lemma 4: Given a cellular network with BSes with varying capacities and flows. Let the capacity of BS j be k_j. Consider a time interval $[0, T]$. We are given real values $\{x_{ij}\}$ for each flow i and BS j, signifying that the flow i must be served by BS j for x_{ij} time. The $\{x_{ij}\}$ values are such that (a) for each BS j, $\sum_i x_{ij} \leq k_j T$, and (b) for each flow i, $\sum_j x_{ij} \leq T$. We claim that there is a schedule of flows onto the BSes (i.e., mapping of BSes to flows, for each time instant) such that at any time instant: (i) A BS uses exactly a unit capacity to serve a flow, and (ii) Each flow is served by at most one BS.

PROOF: We note that proving the claim is tantamount to solving he problem of finding a schedule of flows onto BSes within the given model constraints. We start with giving a high-level outline of the proof. Essentially, the proof of the lemma consists of the following steps.

1) We first prove the claim for the case of a single BS.
2) Then, using (1), we show that it is sufficient to prove the claim for the special case when each BS has only unit capacity.
3) We prove the claim for the special case of unit-capacity BSes in the following steps:
   - We "scale up" the x_{ij} and T values to make all of them positive integers; this can be done by multiplying all of them by an appropriate constant.
   - Next, we add "dummy" flows, "flow-extensions," and BSes such that: (a) For each flow i, $\sum_j x_{ij} = T$, and (b) For each BS j, $\sum_i x_{ij} = T$. This can be easily done, and results in number of flows being equal to the number of BSes.
   - Now, we divide the time interval into T time slots of unit time each. A valid schedule can now be obtained by extracting T perfect matching in an appropriately constructed T-regular bi-partite graph between BSes and flows.
   - Finally, we can remove the dummy BSes, flow-extensions, and flows, and "scale down" the x_{ij} and T values.

Now, we elaborate on each of the above steps.

Single BS Case. For the special case of one BS, we can schedule the given flows as follows. Let the capacity of the BS be k. Since the BS must use exactly a unit capacity to serve a flow, we look at each unit-capacity of the BS as a "server." Thus, we have k servers on which to schedule the flows. We consider the flows in the order of their numbers. We start by scheduling the first flow completely on the first server (starting from $t = 0$). Then, we pick the second flow, and schedule it on the remaining time (i.e., $T - x_{1j}$) of the first server, and if needed, on the second server (starting from $t = 0$). Since $x_{2j} \leq T$, the above ensures that at any time instant, the second flow is scheduled on at most one server. Continuing in the above manner, we can schedule all flows. The above scheduling method ensures that at any time instant, no flow is scheduled on more than one server. Thus, each flow is being served by no more than a unit-capacity, and by the definition of a server, it is being served by at least a unit-capacity. Thus, each flow is being served by exactly a unit-capacity. QED.

Reducing Arbitrary-Capacities Problem to Unit-Capacities Problem. Given an input with BSes of arbitrary capacities, we first schedule the flows on each BS *independently* using the above single-BS approach. Note that this does not yield a valid schedule because at some time instant, a flow may be being

served by multiple BSes. However, the above schedule (even though possibly invalid) gives total time used by each flow on each server of each BS. Note that, we started with only the times x_{ij} used by a flow i on a BS j, and the above (possibly invalid) schedule has given us *some* x_{ij}^h values corresponding to the times used by a i on a server h ($\leq k_j$) of a BS j. It is easy to see that the x_{ij}^h values satisfy the constraints: (i) for each server h of each BS j, $\sum_i x_{ij}^h \leq T$, and (ii) for each flow i, $\sum_{j,h} x_{ij}^h \leq T$. Now, each (j,h) pair can be looked upon as separate BS of unit-capacity, and the derived x_{ij}^h values as the time of a flow i on a unit-capacity (j,h) BS. It is easy to see that a valid schedule of flows $\{i\}$ onto these $\{(j,h)\}$ BSes will give us a valid schedule of flows $\{i\}$ onto the original $\{j\}$ BSes. Thus, if we prove the lemma for the case of unit-capacity BSes, then the lemma for the arbitrary capacity BSes will follow.

Special Case of Base Stations with Unit-Capacities. For this special case, we start with scaling up T and x_{ij} values to make them all integers.[10] Then, we add dummy flows, flow-extensions, and BSes as follows: (i) First, we add dummy flows such that for each BS j, $\sum_i x_{ij}$ becomes T; (ii) Second, we extend the current (dummy plus original) flows, if needed, and store these extensions in new dummy BSes such that for each flow i, $\sum_j x_{ij}$ becomes T. Note that the dummy BSes store all (and only) the dummy-extension of flows. After the above steps, it is easy to see that the number of flows will be equal to the number of BSes.

Given the above setting, we divide the given interval into T time slots of unit-time each. Now, we construct a bi-partite multi-graph between the flows and BSes, wherein we have x_{ij} number of undirected edges between each BS j and flow i. It is easy to see that the graph is T-regular, and the problem of scheduling is tantamount to finding T disjoint perfect matchings in this graph where each matching corresponds to a scheduling of flows on BSes for one time slot. By Hall's Theorem, the constructed bi-partite graph indeed has T disjoint perfect matching. Finally, we can remove the dummy BSes, flow-extensions, and flows, and scale-down the values to get the desired schedule. ■

Proof of Theorem 4. We refer to the offline version of OSF problem the *FSF* problem.

We reduce the discrete unit-disc cover problem to the FSF problem. In a discrete unit-disc cover problem, we are given a set of points and unit-discs in a 2D plane and we find to minimum subset of disks that cover every point [6]. Consider an instance of the discrete unit-disc cover problem, consisting of n discs/sets and total m elements/points. Without loss of generality, let us assume that $m > n$. For this instance, we construct an instance of the FSF problem as follows.

1) For each of the given n discs, we create a BS of unit capacity with the coverage region as the given disc.
2) For each of the given m elements, we create a flow at the location of the point, with an arrival time of 0, size of 1, and deadline of m. We refer to these flows as the *small* flows.
3) In addition to the above, we also create n *large* flows, one for each of the discs. In particular, for each disc, we create a flow with an arrival time of 0, size of m, and deadline of m. The location of each flow is such that it is contained only in its corresponding BS.

We assume unit bit-rates in the above construction. We observe the following.

- First, any BS can completely-serve either (i) its large flow, or (ii) some or all of the small flows contained in its coverage region.
- Second, there is an optimal solution to the constructed FSF instance that completely-serves all the given small flows. This can be easily proved as follows. If there is a small flow i that is not being completely served, then one of the BSes containing it must be serving its large flow. We can replace the large flow by the small flow i, without changing the total number of flows that are completely-served.

Based on the above, it is easy that if the original discrete disc cover instance has a disk cover of size p if and only the constructed scheduling instance has a solution that completely-serves $m+(n-p)$ flows. Thus, an optimal solution of the FSF problem yields an optimal solution of the disk cover problem. ■

Incorporating Uplink Flows. Here, we show how can uplink flows be incorporated in the work done in this paper. Assume a simple model in which uplink flows are separate from downlink flows and we have separate capacities for uplink flows. We now show how to change our solution of MUC, MTC and OSCF problems to incorporate uplink flows.

MUC problem (Section III). To incorporate uplink flows in MUC problem, we make the following change to the LP formulation of the problem originally defined in section III. We make the following modification to the variables.

- We change variable x_{ijt} to denote the amount of time the downlink flow i is served by BS j, during the time interval T_t to T_{t+1}.
- We add variable y_{ijt} to denote the amount of time the downlink flow i is served by BS j, during the time interval T_t to T_{t+1}.

We then modify our set of equations as follow.

- We add $y_{ijt} = 0$ for all t, and for all i,j, where the location l_i of flow i is *not* in the coverage region of cell j.
- We add $y_{ijt} = 0$ for all j, and for all i,t where $T_t < a_i$ (the arrival time) or $T_t \geq d_i$ (the deadline).
- We add $\sum_{t,j} \alpha_{ijt} y_{ijt} = s_i$, for all i, where α_{ijt} is the given bit-rate (a constant in the LP) and s_i is the size of the i flow.
- We add $\sum_i y_{ijt} \leq k(T_{t+1} - T_t)$, for all j,t.

[10]Here, we are implicitly assuming that the given input parameters to the LP (arrival times, deadlines, sizes, bit-rates), are all rational numbers, so that the T and x_{ij} values are rational (to enable scaling-up to integers).

- We add $\sum_j y_{ijt} \leq (T_{t+1} - T_t)$, for all i, t.

MTC problem (Section IV). To incorporate uplink flows in MTC problem, we have to add the following changes to the LP formulation in section IV.

- Add $\sum_i y_{ijt} = k'_j(T_{t+1} - T_t)$, for all j, t.
- Change objective to: minimize $\sum_j k_j + \sum_j k'_j$.

OSCF Problem (section V). To incorporate uplink flows in online scheduling problem, we define $g'(r_m, t)$ for uplink flows the same way that we defined $g(r_m, t)$ for the downlink flows in section V. We then use them to schedule uplink flows the same way we use $g(r_m, t)$ to schedule the downlink flows.