Alleviate Cellular Congestion Through Opportunistic Trough Filling

Yichuan Wang*, Xin Liu

Department of Computer Science, University of California, Davis, CA 95616, USA

Abstract

The demand for cellular data service has been skyrocketing since the debut of data-intensive smart phones and touchpads. However, *not all data are created equal.* Many popular applications on mobile devices, such as email synchronization and social network updates, are delay tolerant. In addition, cellular load varies significantly in both large and small time scales. To alleviate network congestion and improve network performance, we present a set of opportunistic trough filling schemes that leverage the time-variation of network congestion and delay-tolerance of certain traffic in this paper. We consider average delay, deadline, and clearance time as the performance metrics. Simulation results show promising performance improvement over the standard schemes. The work shed lights on addressing the pressing issue of cellular overload.

Received on 08 November 2013; accepted on 19 February 2014; published on 11 April 2014

Keywords: opportunistic scheduling, traffic engineering, cellular operation

doi:10.4108/ws.1.1.e5

1. Introduction

Cellular operators are facing grand challenges in satisfying the fast increasing demand for wireless data service. Mobile Internet is experiencing unforeseen growth in the number of users, capability of devices, and most importantly, volume of traffic. There are 717 million 3G subscribers globally in 2009 [1], and the number will reach 2776 million in 2014 [2]. Data-intensive mobile devices such as smartphones and touchpads are the fastest selling consumer electronics ever. These devices enable game-changing applications, such as video streaming and social networks, on mobile Internet. As a result, mobile Internet has 240,000 terabytes of traffic in 2010, and are expected to grow to 6.3 exabytes (10^6 terabytes) in 2015 [3].

While mobile traffic imposes great pressure on service providers, *not all traffic are created equal.* Many popular applications on mobile devices, such as email synchronization and social network updates, are delay tolerant. In addition, data from a large cellular network shows that there exists a significant lag between content generation and user-initiated upload time, more that 55% uploaded content on mobile network is at least 1 day old [4]. We collected data from regular mobile users, and conservatively assess the delay requirement of each application (details in §6). On average, 65% of the uplink traffic and 70% of the downlink traffic are delay tolerant. In addition to existing delay-tolerant data, service providers are also considering dynamic prices that incent certain applications and users to be more flexible when transmissions occur [5]. We call such data *delay-tolerant data*, which can be leveraged to improve network resource utilization, by opportunistically scheduling its transmission when the network congestion is low and the network condition is more favorable.

The delay tolerance of such jobs varies from subseconds to hours. For example, Emails can tolerate seconds to tens of seconds of delay. Content update, such as Facebook, Twitter, and RSS feeds, can tolerant hundreds of seconds of delay. Content precaching and uploading can tolerant minutes to hours of delay. OS/application updates can tolerant even longer delay. Current cellular networks more or less treat all traffic as equal, which results in performance degradation during high load period.

In this paper, we present a set of opportunistic trough filling schemes that leverage the time-variation of network congestion and delay-tolerance of certain traffic, in order to alleviate network congestion and improve network performance. The key idea is as follows: we classify cellular traffic into two categories: delay-sensitive and delay tolerant. Delay-sensitive

*Corresponding author. Email: yicwang@ucdavis.edu traffic include voice calls, online games, and all user initiated applications, such as web surfing and texting. For delay sensitive jobs, we assume the BS schedules them with high priority and as usual. Then the remaining resource is smartly allocated among delay tolerant jobs using the opportunistic job scheduling policies we propose in the paper. This is referred to as trough filling, i.e., avoiding the high congestion period and filling the valley of network load. We note that network congestion varies in both large time scale (e.g., hours) and small time scale (e.g., seconds). The idea of trough filling is to shape and shift traffic from high congestion period to low congestion period, within the delay expectation range of the delay tolerant jobs.

In such trough filling schemes, intuitively, we want to schedule users in low congestion period and with relatively good channel condition so that resource utilization efficiency is high and congestion is low. At the same time, we need to consider the delay performance of the users. The desirable properties include low average delay, low probability of missing deadline (if the delay tolerant jobs have deadlines), low network congestion, and graceful degradation when the network is overloaded (i.e., not all jobs can be satisfied). The challenge is for the BS to schedule jobs judiciously to achieve such goals.

Compare to machine-job scheduling literature, the main difference of this work is that the channel condition of users is time-varying. In the machine-job scheduling literature, learning effect has been studied, e.g., in [6]. However, learning is a monotonic and deterministic process, while channel variation is a stochastic process. Therefore, their impacts on job scheduling are significantly different. In addition, in the cellular network job scheduling, we consider a pool of resource to be shared among all users in a single cell or cell sector. Therefore, it is a single-server scheduling instead of multiple servers. This property also impacts the selection of scheduling schemes.

This approach also shares features with opportunistic scheduling, also named multi-user diversity, that has been extensively studied and standardized in all 3G/4G cellular systems, e.g., in [7–10]. The basic idea of opportunistic scheduling is to schedule active users in relatively good channel conditions so that resource utilization efficiency is high. At a given time, channel condition information (CQI) is sent back to the BS, and the BS decides the user or the set of users to transmit in a time scale on the order of 1ms. Opportunistic scheduling exploits fast fading of multiple users to improve spectrum efficiency. The main difference is the time scale. The time scale considered in opportunistic trough filling in this paper is much larger, in the order of subseconds to tens of minutes. Because of the time scale difference, we schedule jobs; e.g., a Facebook synchronization or an album upload, while the original multi-user scheduler schedules packets or even bits. In addition, a job has a fixed and known size during arrival, which is useful information in designing scheduling schemes. The time scale difference results in different policies, as well as different performance metrics. For example, job delay and deadlines, instead of (head-of-line) packet delay, are important metrics. In addition, even if all users have the same and static channel condition, intelligent scheduling schemes are still needed in our context for good performance, which is not the same in the multi-user diversity schemes.

The rest of the paper is organized as follows. In Section 2, we present the system model. We then present three problem formulations, minimizing deadline missing probability, average delay, and clearance time in Sections 3, 4, and 5, respectively. Simulations results are then reported, followed by conclusions and future work.

2. System Model

We consider a time-slotted system. The time slot length can be of a subsecond or a few minutes, depending on the time scale of the delay tolerance of the jobs. Note that this time-slot length is much larger than the time slot length in the existing opportunistic scheduling schemes that leverages small-time-scale fast fading, where each time slot is a few milliseconds. Because of this difference, it is possible that a job can be finished in one time slot. For notation convenience, we assume each time slot is one unit of time.

At time slot t, the available resource for DTJ (Delay Tolerant Jobs) is noted as $\eta(t)$. For example, $\eta(t)$ can be considered as the number of available resource blocks in LTE systems, the number of subcarriers in WiMAX systems, or available power in UMTS systems. We emphasize here $\eta(t)$ is available resource left after allocating resources to delay-sensitive jobs, not capacity. We do not make specific assumptions on the distributions of $\eta(t)$. At time t, the resource utilization efficiency of user i is denoted as $Z_i(t)$. We note that $Z_i(t)$ varies over different time slots due to fading environment change and user mobility. We assume that $Z_i(t)$ remains constant at time slot t. Therefore, if user i is given $x_i(t)$ units of resource at time slot t, it can transmit data of volume of $x_i(t) * Z_i(t)$ (recall that each time slot is one unit of time). Note that $x_i(t)$ is the decision variable. Let $F_i(t)$ be the remaining file size of user i at time t. We note that a rational allocation scheme will have $x_i(t) * Z_i(t) \leq F_i(t)$.

The job scheduling process proceeds as follows: at the beginning of the time slot t, the BS allocates (or estimate the need for) resource to delay-sensitive users first, and thus obtain the amount of available resource $\eta(t)$. The BS obtains channel condition and file size information of users (i.e., $Z_i(t)$s and $F_i(t)$s). Then the

BS decides the resource allocation $x_i(t)$. By the end of time slot t, each user has a remaining file size of $F_i(t+1) = \max(0, F_i(t) - x_i(t) * Z_i(t))$. For simplicity, we assume that $x_i(t)$ is a continuous variable, i.e., the BS can allocate resource in arbitrary slices. The assumption is reasonable because we are considering a relatively large time scale, at least subseconds, and each frame size is on the order of 1ms. Therefore, if needed, BS can share resource among users in a TMD fashion.

To simplify the presentation, we assume that each user has at most one job. If a user has multiple jobs, we treat them as a single job. When different jobs have different deadlines, we could handle them in an iterative manner. Alternatively, we can treat different jobs as different users. In addition, we note that in each time slot (which is relatively large), one could still perform small time scale opportunistic scheduling schemes with other DTJs and delay sensitive jobs, e.g., using a proportional fair scheduler in frames. We assume that the impact of such small time scale transmission scheduling can be approximated in the value of $Z_i(t)$, and thus ignored here for simplicity. We hope to better explore this issue in future research.

We summarize the notations used in the paper as follows.

- i: user index
- t: time index
- $Z_i(t)$: channel condition of user i at time t
- $F_i(t)$: remaining file size of user i at time t
- $\eta(t)$: the available resource for delay tolerant jobs at time t
- $x_i(t)$: the amount of resource allocated to user i at time t, the decision variable
- A_i: arrival time of job i
- D_i: deadline of job i
- K_i: time when job i is finished
- J: job completion time

In the following sessions we present three problem fomulations with different optimizing goals: missing deadline, average delay, and job completion time. We also propose a practical index policy in each section to address the complexity in the standard solutions such as MDP (Markov Desicion Process).

3. Deadline

In this case, we assume each job, when generated, has a deadline attached. We consider the problem of minimizing the number of missed deadlines. Let A_i and D_i denote the arrival time and the deadline of the ith job. Therefore, $F_i(A_i)$ is the original file size of job i. The problem can be formally stated as:

$$\begin{array}{ll} \min & \limsup_{t\to\infty} \frac{1}{t}\mathbf{I}\left(\sum_{t=A_i}^{D_i} x_i(t)Z_i(t) < F_i(A_i)\right) \quad (1) \\ \text{subject to} & \sum_i x_i(t) \le \eta(t). \end{array}$$

The objective of the function is to minimize the time-average of the expected probability of missing a deadline. The expectation is taken over the distributions of random channel conditions ($Z_i(t)$s) and random available resource ($\eta(t)$). The constraint is the resource availability constraint at each time slot t. In the problem formulation, $\mathbf{I}(\cdot)$ is the indicator function and $\sum_{t=A_i}^{D_i} x_i(t)Z_i(t) < F_i(A_i)$ is the event that job i does not finish by its deadline. The left hand side ($\sum_{t=A_i}^{D_i} x_i(t)Z_i(t)$) is the total service received by job i between its arrival and its deadline, and the right hand side ($F_i(A_i)$) is the file size.

To solve this problem, one can use a standard MDP (Markov Decision Process) framework, in principle. However, in addition to requiring the stochastic models of both user channel condition and network congestion, the complexity of the above problem is prohibitively high because of the random arrival process and the dynamics of file sizes. Therefore, to address this issue, we propose the following index policy. We first define an index $I_i(t)$ for user i at time t:

$$I_i(t) = f(F_i(t)) \cdot g(Z_i(t)) \cdot h(D_i - t),$$

where $f(\cdot)$ indicates the importance of file size, $g(\cdot)$ represents the importance of instantaneous channel condition, and $h(\cdot)$ is the function that takes into account the urgency of the file. Examples of the above functions are $f(x) = 1$, $g(x) = x$, and $h(D_i - t) = 1/(D_i - t)$, where $D_i > t$. These function types are inspired by machine-job scheduling literature and by opportunistic scheduling literature. The intuition is to favor jobs that are close to their deadlines.

Based on the index, we allocate resource to users in a greedy manner. More specifically, we rank users according to their index from high to low. We first allocate as much resource as possible to the user with the highest index to finish its job; i.e.,

$$x_i(t) = \min\left(\frac{F_i(t)}{Z_i(t)}, \eta(t)\right)$$

If there is additional resource available, we allocate to the next user, and so on until all available resource is depleted or all jobs are scheduled. In practice, we may have extra constraint on the amount of resource allocated to users, e.g., due to device constraint such as power or upper-layer protocol constraints. We can simply include such constraints in the allocation.

4. Average Delay

In this case, we consider the average delay performance of the jobs. Recall that A_i and $F_i(A_i)$ denote the arrival time and the job size of the ith job. Let K_i be the time when job i is finished. The problem can be formally stated as:

$$\begin{array}{rc} \text{min} & \limsup_{N\to\infty} \frac{1}{N}\sum_{i=1}^{N}(K_i - A_i) \\ \text{subject to} & \sum_i x_{ij} \le \eta_j \\ \text{where} & K_i = \min\{T : \sum_{t=A_i}^{T} x_i(t)Z_i(t) \ge F_i(A_i)\}. \end{array} \quad (2)$$

The objective of the function is to minimize average delay of all jobs. In the problem formulation, there is an implicit assumption on network stability. If the network is not stable, then the delay goes to infinity. Similar to the deadline case, in principle, the above problem can be solved using MDP, but suffers high complexity. Therefore, we propose the following index policy. We define an index $I_i(t)$ for user i at time t:

$$I_i(t) = f(F_i(t)) \cdot g(Z_i(t)),$$

We note that to minimize average delay, one would favor smaller jobs, as in machine-job scheduling literature. An example is $I_i(t) = Z_i(t)/F_i(t)$. Similar to the deadline case, a greedy allocation policy can be applied based on the index from high to low.

5. Job Completion Time

Last, we consider the static case where all jobs arrive at the beginning of time slot 1 and there are no new arrivals. In this case, we consider the problem of minimizing job completion time, which is also referred to as clearance time in machine job scheduling literature. In other words, we want to minimize the time where all jobs are finished. Define J as the clearance time, the problem can be formulated as:

$$\text{min} \quad E(J) \quad (3)$$
$$\text{subject to} \quad \sum_i x_i(t) \le \eta(t), \forall t, \quad (4)$$
$$\sum_{t=1}^{J} x_i(t)Z_i(t) \ge F_i(1). \quad (5)$$

In this formulation, the objective is to minimize the expected total completion time. The second constraint is the constraint that all jobs are finished by time J. Note that $F_i(1)$ represents the original file size of user i and the LHS of Eq. (5) represents the total capacity allocated to user i. In this problem formulation, because deadlines do not exist, we propose the following index $I_i(t)$:

$$I_i(t) = g(Z_i(t)). \quad (6)$$

We then rank users according to their index. The users with higher index values have higher access priority. In the simulation, we considered $g(x) = x$. Since our goal is to finish all work earlier, favoring better channel condition improves the throughput when there is no deadline constraint. There is a little bit of subtlety in this formulation. In wireless channels, when the number of users is large, the overall (opportunistic) capacity is higher. For example, suppose that our policy is to choose the best of users to transmit. Then the larger the number of active users, the higher the capacity. It has been well studied in the opportunistic scheduling literature that this capacity gain increases as $\log(\log(n)$, where n is the number of users. Therefore, in principle, to minimize the completion time, one should also consider the number of (remaining) users in the system. However, since the capacity gain increases as $\log(\log(n)$, the difference is significant only when the number of users is small. Therefore, for simplicity, we ignore its impact in the current algorithm. It is our future work to further explore this situation.

6. Real-life Trace

For realistic evaluations on the performance of the proposed algorithm, it is crucial to collect real-life traces from the general public and conduct trace-driven simulations with diverse system parameters.

In this paper, we strive to realistically evaluate the benefit of leveraging delay tolerant traffic for the general public in real life scenarios. To achieve this goal, we collected data from the users of an Android application called PhotoSync[1]. We published PhotoSync on Google Play[11] in July 2012. There have been more than 10,000 downloads in the first eight months. Among them, with clear privacy notification, about 1700+ users participate in our data collection. After filtering out the profiles with zero-length and corrupted data, we ended up with the profiles from $\sim$ 700 users. Among them, we have more than 100 users with 30+ days of profiles, which are used in our performance evaluations.

We collect from users, among other metrics, 3G signal strength, WiFi connectivity, and application traffic. We observed that the top 50 applications contribute $\sim$ 80% of the traffic, and the top 10 applications contribute $\sim$ 60% of the traffic. We also roughly classify the applications into two groups, applications that generate: (i) delay tolerant and (ii) real-time traffic. For downlink traffic, we consider Dropbox, social network content pre-fetching, and application update

[1]The app follows the required privacy guideline, fully discloses the information collected, and allows users to easily opt out from sending their profiles to us. We observe that a large number of users choose to opt out, an indication of user privacy-awareness and the effectiveness of the privacy disclosure. Furthermore, we only keep a hashed user ID through a one-way hash function, which is not reversible to ensure anonymity.

as delay tolerant; and Android browser and YouTube as real-time. For uplink traffic, we consider Dropbox, social network photo backup as delay tolerant; and browser, messaging, and video conferencing as real-time. Examples of the applications generating delay tolerant traffic include cloud storage applications (such as Dropbox) and social network applications (such as Facebook and Google+), which may tolerate delays in the order of minutes, and photo sharing applications (such as Instagram), which may tolerate delays in the order of hours [12]. For applications that are in the grey area, we consider them as real-time traffic, to be conservative. We then compute the per-user delay tolerant traffic fraction of the top 50 applications (in each direction). There are on average 65% (uplink) and 70% (downlink) delay tolerant traffic, which shows the potential of the proposed algorithm.

Real-life simulations use the same setup described in §7.1. The network condition $Z_i(t)$, file size $F_i(t)$, and job arrival time A_i used in the simulations are extracted from the user traces.

7. Evaluation

In this section we evaluate the proposed algorithm using simulations with both synthetic traces and real-life traces collected from the general public. We compare the performance of our algorithm under three different optimization goals, namely average delay, missed deadline, and completion time, with a naive algorithm that evenly allocates resources to all users and a baseline algorithm that allocate resources solely base on channel condition.

7.1. Simulation Setup

First, we explain the simulation setup. We assume the time is slotted. There are totally N users in the system who have exactly one file to transmit during the test. In the completion time tests, all files arrive at time slot 1.

When a job arrives, it has a deadline D_i (if necessary, not considered in the average delay case and the clearance case). At each time slot, each user experiences a channel condition $Z_i(t)$ distributed between 1 and 4. At each time slot, the available resource is between 1 and 50.

In a single test, at the beginning of each time slot j, we observe the current network condition $Z_i(t)$ for all users who have file to transmit. Each time slot, the system has $\eta(t)$ units of resources left for delay tolerant jobs. The proposed algorithm allocates $x_i(t)$ units of resources to each job i. The allocated resources for each file is guaranteed not to exceed needed resources nor the total available resource. Thus the final transmission rate per time slot for each file is $x_i(t) \cdot Z_i(t)$. The remaining size of each file is updated after transmission at the end of each time slot.

Objective	Index	Priority
Deadline	$Z_i(t)/(F_i(t)* max(D_i - j, 1))$	Jobs close to deadline
Delay	$Z_i(t)/F_i(t)$	Short jobs
Completion	$Z_i(t) * F_i(t)$	Long jobs

Table 1. Summary of different objectives and index functions.

Next, we evaluate the proposed index policies via simulation with both synthetic and real-life traces. In both simulations, we compare the index policies with different objectives against two baseline policies, a channel-only policy and an even policy. The index algorithm is noted as *index* in all figures. Channel-only policy is a special index policy with $I_i(t) = Z_i(t)$, which favors users with the best channel conditions, noted as *channel-only* in the figures. Finally the baseline policy where resource $\eta(t)$ is evenly allocated to all users, is noted as *even* in all figures. The average delay, missed deadline, and completion time of the index algorithm is compared against the previously mentioned two baseline policies. Note that different index functions are used for different objectives. We summarize the index functions used in the policies and their intuition in Table 1. In all the index functions, we favor users with good channel conditions with a linear function, which takes inspiration from the majority of optimal policies in multi-user diversity scheduling schemes. In addition, under different objective, file size and deadline are considered differently. In the simulations, different functions of file size and deadline are also considered, including log and square root functions, but not reported here individually for conciseness.

The completion time J is calculated for each test, which is the latest time all job finishes. The average delay of all jobs and the number of jobs which missed its deadline are also collected for all tests.

7.2. Synthetic Trace

First we evaluate the proposed algorithm using synthetic traces to evaluate its performance in a theoretic context where we can control system parameters and isolate specific behaviors. For example, the system load is hard to control when the file size are from real traces, and we cannot control parameters such as max file size and channel variation.

For average delay and missed deadline tests, file i has a random arrival time A_i, uniformly distributed in the test horizon (which is similar to the effect of Poisson arrival).

The deadline D_i is uniformly distributed between the current time and the horizon. Each file is of a random size uniformly distributed between the minimum file size 1 and the maximum file size 100. The difference reflects the diversity of applications. At each time slot, each user experiences a random channel condition, such that $Z_i(t)$ is uniformly distributed between 1 and 4. At each time slot, the available resource is randomly chosen between 1 and 50.

In synthetic simulations, we assume all users have homogeneous channel conditions. This is to allow us to focus on the impact of file size and load. If the users have heterogeneous channel conditions (e.g., being close to or far away from the base station), appropriate normalization may be considered; e.g., normalizing over its own average condition so that high value indicates relatively good channel condition.

Each test is repeated 500 times to be statistically meaningful. The parameters such as system load are adjust to study the impact of that particular argument. System load represents the ratio of total traffic versus available system resource available during the whole test. If we define total number of file as N and the total length of the simulation as H, the system load is defined as

$$L = \frac{N \cdot E(F_{i0})}{H \cdot E(\eta) \cdot E(Z_{ij})}. \tag{7}$$

The load is an indicator of how busy the network is.

7.3. Simulation Results

Average Delay. We first evaluate the performance of the proposed index algorithm under the objective of minimizing average delay. The index function used is

$$I_i(t) = Z_i(t)/F_i(t) \tag{8}$$

We note that the proposed index policy favors jobs with shorter remaining time and thus results in lower average delay compared to the channel only policy and even policy.

For the synthetic simulation, average delay are plotted in Figure 1 for the three policies as the system load varies from 0.7 to 1.5. For the real-life simulation, average delay are plotted in Figure 2 for the three policies as the system load varies from 0.5 to 1.5. From the figures, we observe that both index policy and channel-only policy show significant improvement over the *even* policy, which indicates the user diversity gain. The index algorithm outperform the channel only algorithm by over 20% in terms of average delay.

Figure 3 shows the throughput of all three algorithms. Throughput defined as the total size of all *completed* jobs during the simulation. While the index policy focuses on average delay in this cases, its throughput remain similar to channel only policy.

Furthermore, in Figure 4, we illustrate the traffic smoothing impact of the proposed opportunistic trough filling scheme. The solid line indicates the load without opportunistic trough filling (OTF) and the dashed line is the load after trough filling. The load before OTF

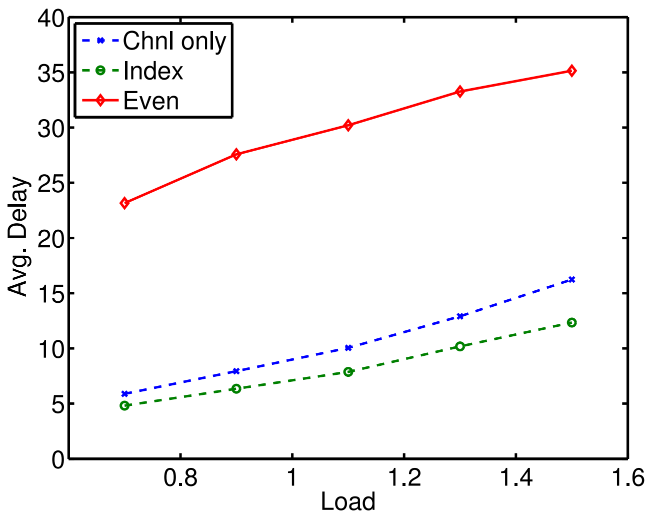

Figure 1. Average Delay when Index Policy Objective is Minimizing Average Delay, Synthetic Trace

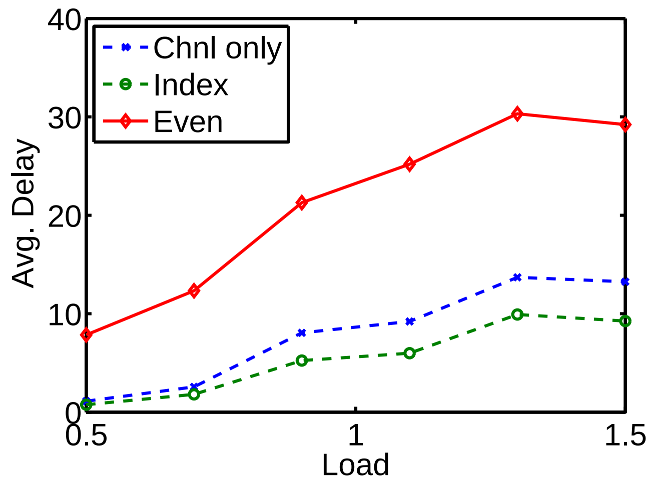

Figure 2. Average Delay when Index Policy Objective is Minimizing Average Delay, Real-life Trace

sometimes exceeds the available resource, which is capped at 50. After trough filling, the traffic is much more smooth. This figure is an intuitive illustration of “trough filling”; i.e., filling the valley and reducing the peak.

Deadline. We study the case where each job has a given deadline in this section. The index function used is

$$I_i(t) = Z_i(t)/(F_i(t) * max(D_i - j, 1)) \tag{9}$$

For synthetic simulations, missed deadlines and average delay are plotted in Figure 5 and Figure 6 respectively for the three policies as the system load varies from 0.7 to 1.5.

In synthetic simulation, we also test the impact of the file size variation. As shown in Fig. 7, the percentage of

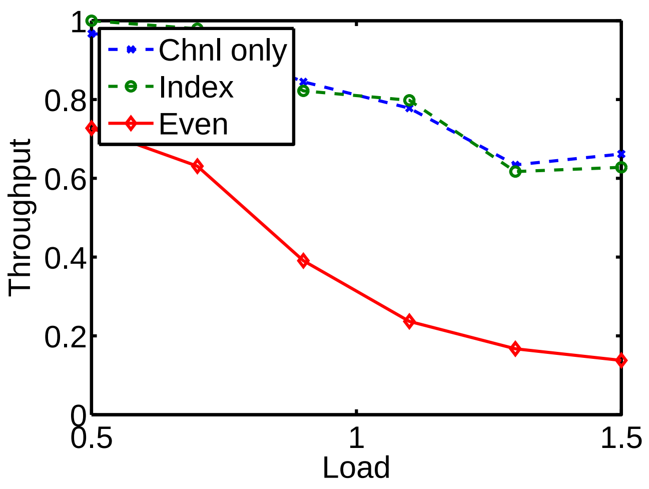

Figure 3. Throughput when Index Policy Objective is Minimizing Average Delay, Real-life Trace

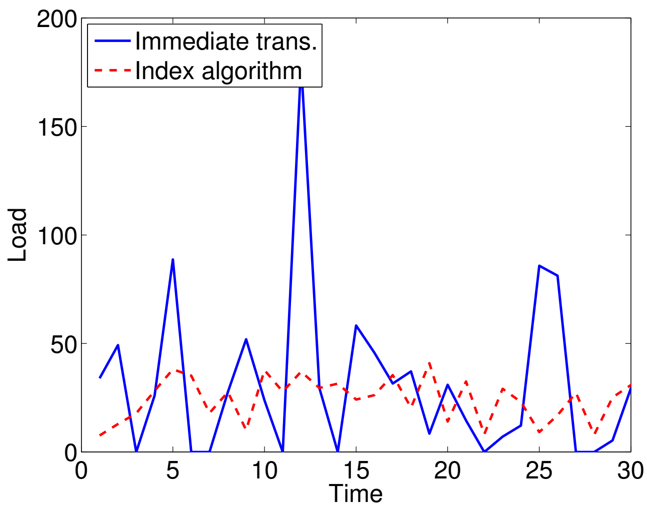

Figure 4. Traffic smoothing

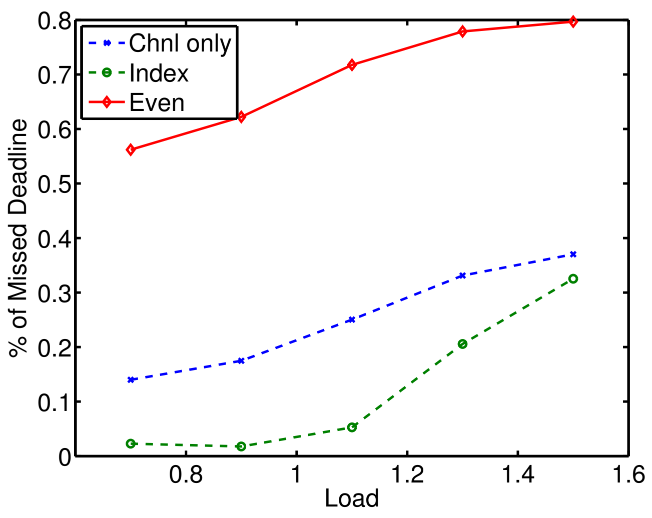

Figure 5. Percentage of Jobs Missed Deadline when Index Policy Objective is Minimizing Missed Deadlines, Synthetic Trace

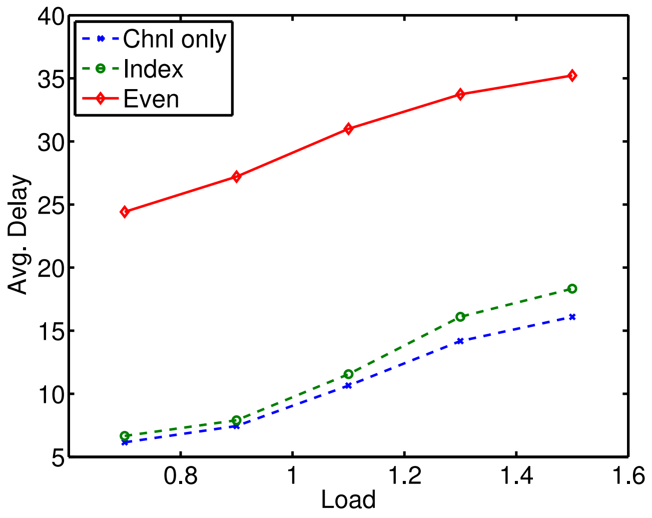

Figure 6. Average Delay when Index Policy Objective is Minimizing Missed Deadlines, Synthetic Trace

jobs missed the deadline increases as the maximum file size increases in the synthetic trace. Intuitively when the maximum and average file size increases, even the system load stay the same, the it is more likely that the larger jobs will miss the deadline.

For real-life simulations, missed deadlines and average delay are plotted in Figure 8 and Figure 9 respectively for the three policies as the system load varies from 0.7 to 1.5.

In Figure 5 and Figure 8, the x-axis is the load, the y-axis is percentage of jobs which missed their deadlines. Compared with the channel-only policy, the proposed index policy favors users close to the deadline, and thus has much lower missing rate. The cost of it is the slightly longer average delay.

In Figure 10 shows the throughput of all three algorithm in this simulation. The index policy strive to complete jobs within their deadlines, thus having a higher throughput then the channel-only policy

Job Completion Time. As shown in Figure 11, in the job completion time simulation, our index policy is the same as channel-only policy, with $I_i(t) = Z_i(t)$. At the first sight, this may look different from the classic job scheduling literature, where the scheduling of the largest jobs often significantly impacts the completion time. However, in our case, there is one single pool of resource that is shared by all jobs similar to a single server queue with time-varying capacity. If users' channel condition does not change over time, then any work-conserving scheduling scheme will result in the same performance. Therefore, channel-only scheduling

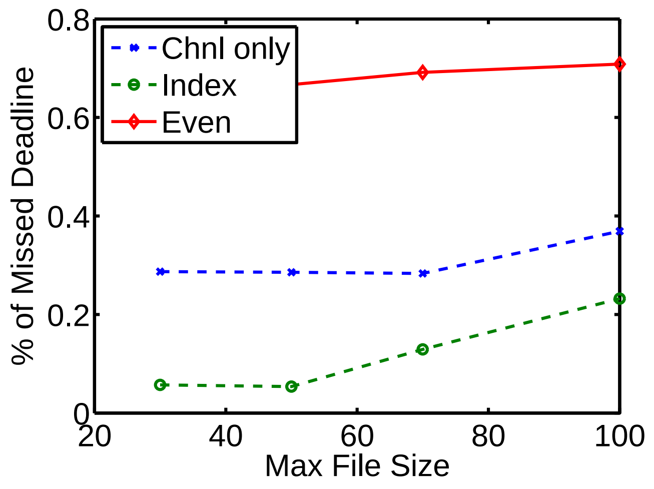

Figure 7. Percentage of Missed Deadline with Different Maximum File Size, Synthetic Trace

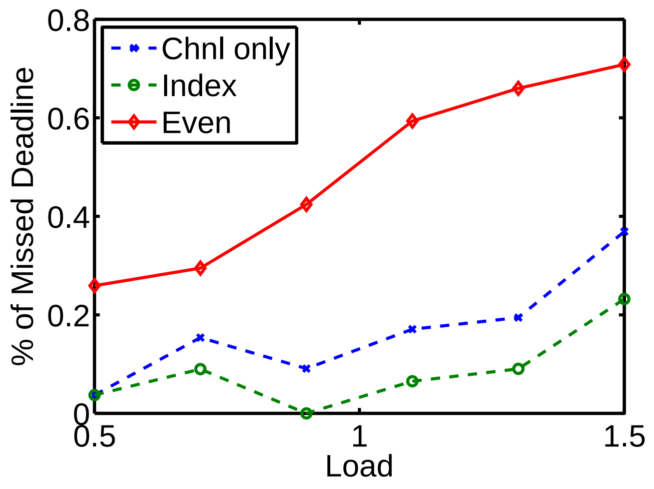

Figure 8. Percentage of Jobs Missed Deadline when Index Policy Objective is Minimizing Missed Deadlines, Real-life Trace

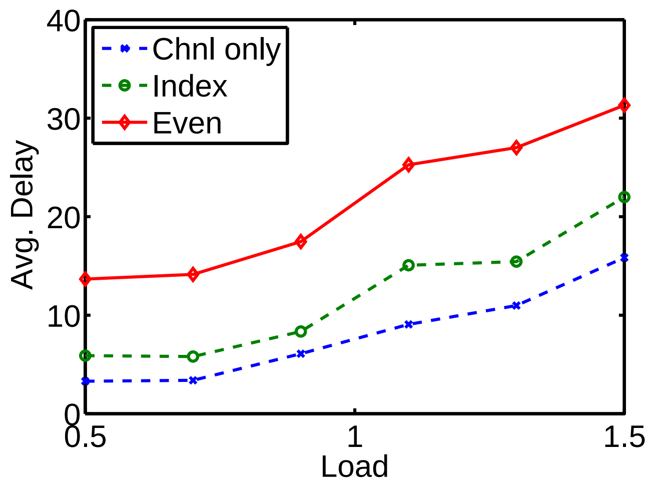

Figure 9. Average Delay when Index Policy Objective is Minimizing Missed Deadlines, Real-life Trace

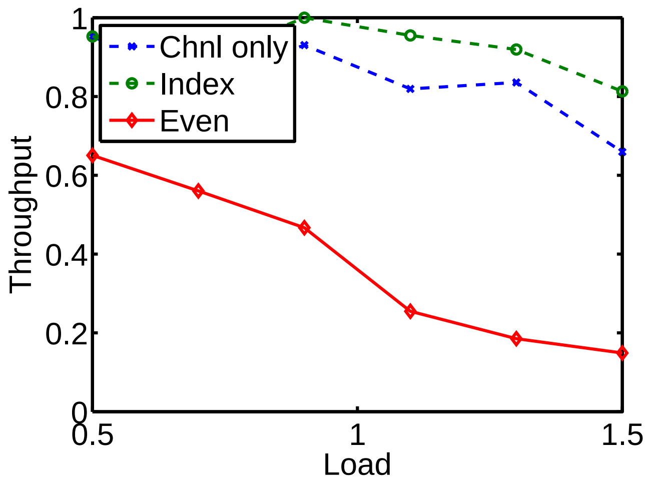

Figure 10. Throughput when Index Policy Objective is Minimizing Missed Deadlines, Real-life Trace

works well in our simulations by opportunistically favor users in good channel conditions.

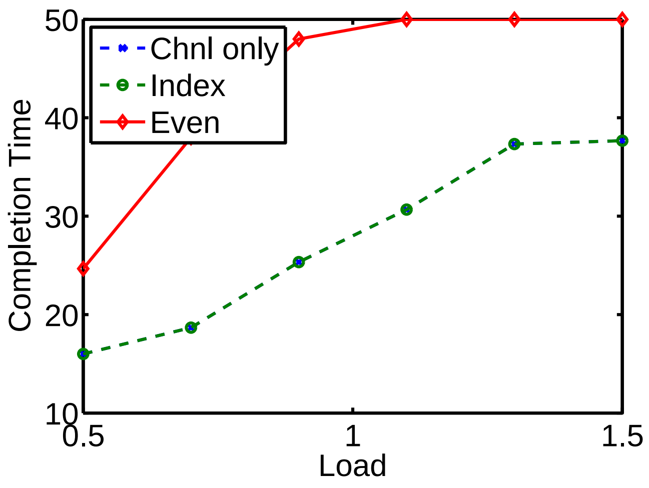

Figure 11. Job Completion Time when Index Policy Objective is Minimizing Average Delay

8. Conclusion and Future Work

This paper is a first attempt to develop large-time-scale opportunistic trough-filling schemes in cellular networks. The objective is to alleviate network congestion and improve cellular performance. Two factors are considered: first, a large amount of jobs are delay-tolerant, we can delay them to times when the network is less congested. Second, because of this delay-tolerance, users can be scheduled to transmit in relatively good channel conditions that improves spectrum efficiency; e.g., due to fading environment or user mobility. Both effects alleviate network congestion and improve user experience. We have considered three distinct objectives: minimizing delay, minimizing missed deadlines, and minimizing clearance time. Because of the practical and numerical complexity involved in their optimal solutions, we resort to heuristic index policies that take into account file size, channel condition, and deadline. Numerical results are promising, indicating large performance gain over naive algorithms that do not perform such opportunistic trough filling.

Many challenging problems are to be addressed. First, our current schemes are heuristics, mostly inspired by the existing literature on machine job scheduling and opportunistic scheduling. We hope to evaluate them in a more rigorous manner, using metrics such as throughput optimality, average delay, and average queue flow probability. We hope to further develop optimal schemes to study the impact of user channel variation in reality. In addition, we hope to consider practical issues, including signaling overhead and distributed implementations.

References

[1] (2010), World Telecommunication/ICT Indicators Database, http://www.itu.int/ITU-D/ict/statistics/.

[2] (2009), The Mobile Internet Report, http://www.morganstanley.com/institutional/techresearch/pdfs/mobile_internet_report.pdf.

[3] (2010), Cisco Visual Networking Index: Global Mobile Data Traffic Forecast Update, 2010âĂŞ2015, http://www.cisco.com/en/US/solutions/collateral/ns341/ns525/ns537/ns705/ns827/white_paper_c11-520862.html.

[4] Trestian, I., Ranjan, S., Kuzmanovic, A., and Nucci, A. (2011) Taming user-generated content in mobile networks via drop zones. In *Proc. of IEEE INFOCOM'11*.

[5] Joe-Wong, C., Ha, S. and Chiang, M. Time-dependent internet pricing.

[6] Gur and Mosheiov (2001) Scheduling problems with a learning effect. *European Journal of Operational Research* **132**(3): 687 – 693. doi:10.1016/S0377-2217(00)00175-2, URL http://www.sciencedirect.com/science/article/pii/S0377221700001752.

[7] Liu, X., Chong, E.K.P. and Shroff, N.B. (2003) A framework for opportunistic scheduling in wireless networks. *Computer Networks* **41**: 451–474.

[8] Lin, X., Shroff, N.B. and Srikant, R. A tutorial on cross-layer optimization in wireless networks. *IEEE Journal on Selected Areas in Communications, Special Issue on "Non-Linear Optimization of Communication Systems", vol. 24, no. 8, August 2006.* .

[9] Bodas, S., Shakkottai, S., Ying, L. and Srikant, R. (2009) Scheduling in multi-channel wireless networks: rate function optimality in the small-buffer regime. In *Proceedings of the eleventh international joint conference on Measurement and modeling of computer systems*, SIGMETRICS '09 (New York, NY, USA: ACM): 121–132. doi:http://doi.acm.org/10.1145/1555349.1555364, URL http://doi.acm.org/10.1145/1555349.1555364.

[10] Shreeshankar, B., Sanjay, S., Lei, Y. and R., S. (2010) Low-complexity scheduling algorithms for multi-channel downlink wireless networks. In *Proceedings of the 29th conference on Information communications*, INFOCOM'10 (Piscataway, NJ, USA: IEEE Press): 2222–2230.

[11] Photosync. https://play.google.com/store/apps/details?id=com.metaisle.photosync.

[12] Trestian, I., Ranjan, S., Kuzmanovic, A. and Nucci, A. (2011) Taming user-generated content in mobile networks via drop zones. In *Proc. of IEEE INFOCOM'11* (Shanghai, China): 2040–2048.