

# Performance Tradeoffs in Mobile Computing: To Fetch Or Not To Fetch?

Aditya Dua  
Department of Electrical  
Engineering  
Stanford University  
Stanford, CA 94305  
dua@stanford.edu

Nicholas Bambos  
Department of Electrical  
Engineering  
Stanford University  
Stanford, CA 94305  
bambos@stanford.edu

Jatinder Pal Singh  
Deutsche Telekom  
Laboratories  
Ernst-Reuter-Platz 7  
10587 Berlin, Germany  
jatinder.singh@telekom.de

## ABSTRACT

As portable wireless devices have become commonplace today, the popularity and acceptance of a broad range of mobile applications is higher than ever. Acceptable user experience warrants low latency of execution of computational tasks on the mobile terminals which owing to their portability requirements are typically constrained in memory and storage capacity. It is hence important to judiciously fetch new tasks from application servers while background applications are running on the device. In this work we use a dynamic programming (DP) framework to capture the tradeoff between *congestion* caused due to background tasks running on a mobile device and latency of execution of new tasks fetched from central server over time-varying wireless channel. Adopting a baseline model for wireless channel variations, rate of task execution, and congestion cost per unit time experience at a mobile terminal, we establish the optimality of a *switchover* policy which makes a decision to fetch or not to fetch depending on the number of tasks queued up for the mobile terminal at the central server and at the mobile terminal itself. We develop a heuristic-based approximation to the optimal control using *policy iteration* methodology and present an quasi-static algorithm fetch-or-not (FON) which updates the optimal switchover policy every time epoch to make a decision whether or not to fetch a computational task. We further present modeling extensions to the baseline scenario and discuss finite state Markov chain modeling for wireless channel, power control measures for task transmission, and time-varying congestion at the mobile terminal.

## Keywords

Mobile computing, Buffer management, Dynamic programming, Tandem queues

## 1. INTRODUCTION

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to publish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.

Copyright 200X ACM X-XXXXX-XX-X/XX/XX ...\$5.00.

The advent of portable devices with wireless communication capability (e.g., PDAs, mobile phones) has provided great impetus to mobile computing applications. These applications encompass a broad spectrum — location based services, streaming compressed media (e.g., video) to mobile users, distributed execution of parallelizable computational tasks over multiple cooperating mobile computers, sensor networks, wearable computing, etc. All these applications are executed on nodes with limited processing power, battery life, and memory. Moreover, these nodes communicate with a central server/controller over error prone wireless links with fluctuating quality. Intelligent resource management and robust adaptation to variations in the environment are therefore essential for optimizing the performance of mobile computing applications.

This paper focuses on the problem of adaptive memory management at mobile terminals. Broadly speaking, the goal is to minimize the latency experienced by computational tasks, while judiciously utilizing the scarce memory resources available at the mobile terminal. More specifically, we are interested in a mobile computing scenario, where a wireless radio equipped mobile terminal (MT) sequentially fetches computational “tasks” from a central server (CS) over an error prone wireless link. It takes a random amount of time to transmit the task from the server to the mobile terminal due to wireless channel fluctuations. Further, it takes a random amount of time to complete the execution of each task at the MT. The MT chooses the times at which it wants to fetch tasks from the CS.

A similar problem of joint buffer management and power control was addressed by Gitzenis and Bambos [5] in the context of client/server interaction for predictive caching. The power aware prefetching problem was also studied by Cao in [3]. Dua and Bambos [4] examined buffer management for wireless media streaming, where the objective was to minimize buffer underflows to ensure smooth media play-out, at the same time using the limited buffer at the mobile terminal in a careful manner. Buffer management for media streaming was also studied by Kalman et al. [8], Li et al. [9], etc. In other work on memory management in mobile computing scenarios, Ip et al. [7] proposed an adaptive buffer control scheme to prevent buffer overflows at MTs in a real-time object-oriented computing environment, and Yokoyama et al. [13] proposed a memory management architecture for implementing shared memory between the CS and the MT. To the best of our knowledge, the *latency vs. buffer tradeoff* in mobile computing studied in this paper has

not been addressed in the existing literature.

For convenience, we will refer to a set of related tasks as an *application*. An application could be stand-alone, involving only the CS and the MT, or could also be part of a larger distributed computation involving multiple MTs. From an application performance perspective, the objective is to minimize the total latency incurred in executing a set of tasks. Clearly, from this point of view, the best strategy is to buffer all the tasks at the MT as quickly as possible, i.e., the MT fetches tasks from the CS at every possible opportunity. However, *memory is a limited and expensive resource* at a typical MT and is shared by several applications (each one with its own set of tasks) which are executed concurrently. These applications could comprise of computational tasks fetched from a different CS or neighboring MT(s), or could also be system specific processes related to the operating system, wireless connectivity etc.

The MT needs to be “smart” in terms of the number of tasks it buffers locally for each application, because allocating a large chunk of memory to one application to improve its performance is likely to hurt the performance of other applications. From this perspective, the MT should request a new task from the CS as conservatively as possible.

An exact analysis of the tradeoffs involved in this situation would involve modeling the dynamics of each application individually and considering the interactions/coupling induced between them by the shared memory resource. This holistic approach, however, is cumbersome (both analytically and computationally) and also not scalable. An alternative approach, which we adopt here, is to focus on the dynamics of one application and model other applications as “background congestion” for this foreground application, and capture the coupling between them through a minimal set of parameters.

Since the foreground and background applications share a common limited resource — the memory — the background applications create *congestion* for the foreground application. Thus, their presence can be captured effectively through a *congestion cost*. For example, if there were no background applications, the congestion cost would be 0 and the entire available memory could be dedicated to the foreground application. On the other hand, if there were a large number of background applications, the congestion cost would be quite large, and as a consequence, the foreground application would be allocated only a small chunk of the memory.

Based on the foregoing discussion, the dilemma faced by the MT in each decision epoch is the following: Fetch a task and possibly incur an additional congestion cost or not fetch a task and possibly increase the latency experienced by the application. To fetch or not to fetch?

In this paper, we study the above decision making problem in a dynamic programming (DP) framework [2] and obtain the optimal tradeoff between the congestion cost and the latency cost. In Section 2, we examine in detail a baseline model which captures the key tradeoffs in the problem. The baseline model is essentially a discrete time two queue tandem network and is described by three parameters — the probability of successful transmission of a computational block over the wireless link from the CS to the MT ( $s$ ), the rate at which tasks are executed at the MT ( $\mu$ ), and the congestion cost per unit time experienced at the MT due to the presence of background applications ( $c$ ). We

establish structural properties of the optimal control for the baseline model, the most important one being the optimality of a *switchover policy* [12]. In Section 3, we employ the technique of policy iteration to develop an approximation to the optimal control, which can be computed analytically in terms of  $s$ ,  $\mu$ , and  $c$ , without explicitly solving any dynamic programs. The approximations are illustrated via numerical examples. We leverage this approximation to develop a low complexity heuristic “fetching algorithm” for scenarios with dynamically changing  $s$ ,  $\mu$ , and  $c$ . We also develop several extensions of the baseline model in Section 4 to demonstrate how dynamic scenarios can be studied formally in a DP framework. A tradeoff between model complexity and accuracy arises naturally in these extensions. We conclude the paper in Section 5 and also briefly describe our ongoing research efforts as well as directions for future work.

The focus of this paper on the problem formulation aspects and parsimonious mathematical modeling of a complex stochastic system. The two queue tandem formulation is novel in the context of mobile computing scenarios. Also, the two queue tandem is a very hard control problem to analyze, as is evident from [10, 6, 11] and other similar literature. Our work is a first attempt to develop systematic approximations to the optimal control policies associated with this class of problems. Our strategy here was to develop low complexity approximations and practical algorithms based on *provable* structural properties of the model and evaluate their accuracy via numerical examples. We are currently involved in an extensive experimental evaluation of the proposed algorithm in realistic test scenarios.

## 2. BASELINE MODEL

The baseline model which we study in detail is a controlled two queue tandem network (in discrete/slotted time), as depicted in Fig. 1. In the model,  $\mathcal{Q}_1$  represents the queue at the central server (CS) and  $\mathcal{Q}_2$  represents the queue at the mobile terminal (MT). As stated before, an application comprises of a set of tasks. For the moment, we assume that each task is contained in a single computational block or network block. We will therefore use the words block, block, and task interchangeably throughout the paper. At most one block can be transmitted over the air interface from the CS to the MT in a time-slot. At the beginning of a time-slot, the MT has the option of fetching a block from the CS over an error-prone wireless link.

For the baseline model, we will consider a two state independent and identically distributed (i.i.d.) wireless channel. A more general Markovian wireless channel model with multiple states will be considered in Section 4. In every time-slot, the channel is in the ON state w.p.  $s$  and in the OFF state w.p.  $1 - s$ , independent of its state in past/future time-slots. If the channel is in ON state, a block transmission over the channel is successful w.p. 1, and if the channel is in a OFF state the transmission fails w.p. 1. We will assume that the MT knows  $s$ , but *does not* know the actual channel realization (ON or OFF) at the time of making a decision (to fetch or not). Given this channel model, the transmission time of each block over the wireless link from the CS to the MT is a geometrically distributed random variable (RV) with mean  $1/s$ . We will also assume that the processing/execution times of tasks at the MT is geometrically distributed with mean  $1/\mu$ , independently for every task. The randomness in execution times is attributed to

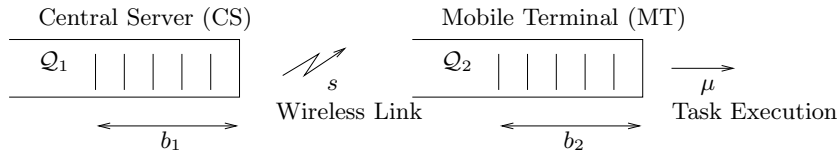


Figure 1: System Model

two reasons — (i) variable sized tasks<sup>1</sup> and (ii) competition for shared processor with tasks from other application being executed at the MT.

## 2.1 State and costs

Let  $\mathbf{b} = (b_1, b_2)$  denote the *state* of the system, where  $b_1$  is the number of remaining tasks at the CS (in queue  $\mathcal{Q}_1$ ) for the application of interest and  $b_2$  is the number of tasks waiting to be processed at the MT (in queue  $\mathcal{Q}_2$ ). A cost of 1 unit per task is incurred for every time-slot that a task spends waiting in  $\mathcal{Q}_1$  and a cost of  $c > 1$  units per task is incurred for every time-slot that a task spends in  $\mathcal{Q}_2$ .

The parameter  $c$  captures the congestion cost as experienced by the foreground application at the MT and formalizes our earlier claim that memory is an expensive resource at the MT (shared by several applications) and therefore needs to be used judiciously. In the extreme case, if  $c \ll 1$ , the MT will fetch all the tasks it needs to process as quickly as possible and thereby reduce the overall latency experienced by the application. On the other hand, if  $c \gg 1$ , the MT is unlikely to fetch a task from the CS until its buffer empties, resulting in higher latency for the application. Thus, the parameter  $c$  captures the tradeoff between the congestion cost and the latency cost. It also captures the coupling between the background and foreground applications in a parsimonious fashion, without explicitly modeling the former. A well chosen value for  $c$  ensures that the MT requests tasks judiciously from the CS — infrequently enough to prevent buffer overflows (and hence avoid disrupting other applications) and frequently enough to prevent buffer underflows (and hence avoid processor underutilization). While a *fixed* congestion cost is assumed for the baseline model (i.e., memory requirements of background applications are fixed), extensions to time-varying congestion will be considered in Section 4.

Typically, buffering is cheap at the CS. Then why should tasks queued at the CS incur a holding cost of 1 unit per time-slot? This cost creates a *backlog pressure*, which drives down the overall latency experienced by the foreground application. To see this argument clearly, consider a scenario where tasks queued at the CS incur a zero backlog cost. In this case, the MT will request a new task only when it has finished processing the currently executing task. Since block transmission times over the wireless medium are random, the consequence would be potential under-utilization of the processor at the MT, which is bad from an application latency perspective. A non-zero holding cost at the CS prevents such a scenario from arising. If the MT requests tasks from the CS very infrequently, the backlog pressure at the CS will start building up, which will eventually force

the MT to request a task. When only a few tasks remain at the CS, the MT can request new tasks more conservatively and hence drive down the congestion costs it incurs by using its expensive resource, viz. the memory. Since any non-zero holding cost at the CS will suffice for generating the requisite backlog pressure, we set it to a normalized value of 1, without any loss of generality.

To summarize, the baseline model is parameterized by three parameters —  $s, \mu$ , and  $c$ , as depicted in Fig. 1.

## 2.2 Actions and system dynamics

In the baseline model, we will assume that given the initial system state at time  $t = 0$ , i.e., the number of tasks in  $\mathcal{Q}_1$  and  $\mathcal{Q}_2$ , no further tasks arrive to  $\mathcal{Q}_1$  thereafter. Thus, the eventual state of the system when all tasks have been executed is  $(0, 0)$ . In other words,  $(0, 0)$  is a *terminal state* for the system. Given the system state  $\mathbf{b}$  at the beginning of a time-slot, the MT has to choose one of two actions:

1. FE: Fetch a task from the CS, or
2.  $\overline{\text{FE}}$ : do not fetch a task from the CS.

Our objective is to determine the “optimal” choice of action in every system state. The set of actions are collectively known as a *policy*. More formally,

*Definition 1.* A policy  $\pi$  is a mapping  $\pi : \mathbb{Z}_+ \times \mathbb{Z}_+ \mapsto \{\text{FE}, \overline{\text{FE}}\}$ , which assigns one of the two possible actions (FE or  $\overline{\text{FE}}$ ) to each system state  $(\mathbf{b})$ .

If policy  $\pi$  chooses action  $\overline{\text{FE}}$  in state  $(b_1, b_2)$ , one of the following state transitions occur (assuming  $b_2 > 0$ ):

1. A task finishes processing at the MT, and the new state is  $(b_1, b_2 - 1)$  w.p.  $\mu$ .
2. No task finishes processing at the MT, and state continues to be  $(b_1, b_2)$  w.p.  $1 - \mu$ .

If  $\pi$  chooses action FE in state  $(b_1, b_2)$ , one of the following state transitions occur (assuming  $b_1, b_2 > 0$ ):

1. A task finishes processing at the MT and the transmission from the CS to the MT is successful. The new state is  $(b_1 - 1, b_2)$  w.p.  $s\mu$ .
2. A task finishes processing at the MT but the transmission from the CS to the MT fails. The new state is  $(b_1, b_2 - 1)$  w.p.  $(1 - s)\mu$ .
3. No task finishes processing at the MT but the transmission from the CS to the MT is successful. The new state is  $(b_1 - 1, b_2 + 1)$  w.p.  $s(1 - \mu)$ .
4. No task finishes processing at the MT and the transmission from the CS to the MT fails. The state continues to be  $(b_1, b_2)$  w.p.  $(1 - s)(1 - \mu)$ .

<sup>1</sup>The size of a task here refers to the amount of computational effort it requires. Two tasks encoded in identical sized network blocks can have very different computational requirements.

If  $b_1 = 0, b_2 > 0$ , i.e. no more tasks remain at the CS, the only possible action is FE. If  $b_1 > 0, b_2 = 0$ , i.e. the buffer at the MT is empty, both actions are feasible. The possible state transitions under each of the actions can be described in the above fashion.

### 2.3 Dynamic programming (DP) formulation

Given the system dynamics, we are interested in computing the *optimal policy*  $\pi^*$ , which minimizes the total expected cost incurred in reaching terminal state  $(0, 0)$ , starting in any state  $(b_1, b_2)$ . This is a *stochastic shortest path* (SSP) problem and is amenable to analysis in a DP framework. Denoting by  $V(b_1, b_2)$  the *value function*, i.e., the expected cost incurred under the optimal policy  $\pi^*$ , starting in state  $(b_1, b_2)$ , we know from the theory of DP that it satisfies the following *Bellman's equations* for  $b_1, b_2 > 0$ :

$$V(\mathbf{b}) = \min\{\mu V(\mathbf{b} - \mathbf{e}_2) + \bar{\mu}V(\mathbf{b}), \\ s\mu V(\mathbf{b} - \mathbf{e}_1) + \bar{s}\mu V(\mathbf{b} - \mathbf{e}_2) + \\ s\bar{\mu}V(\mathbf{b} - \mathbf{e}_1 + \mathbf{e}_2) + \bar{s}\bar{\mu}V(\mathbf{b})\} + \langle \mathbf{c}, \mathbf{b} \rangle, \quad (1)$$

where  $\mathbf{b} = (b_1 \ b_2)$ ,  $\mathbf{e}_1 = (1 \ 0)$ ,  $\mathbf{e}_2 = (0 \ 1)$ ,  $\mathbf{c} = (1 \ c)$ ,  $\bar{s} = 1 - s$ ,  $\bar{\mu} = 1 - \mu$ , and  $\langle \mathbf{c}, \mathbf{b} \rangle = b_1 + cb_2$ . The first argument of min in (1) is the optimal expected future cost if action  $\overline{\text{FE}}$  is chosen in state  $\mathbf{b}$ . The second argument of min is the optimal expected future cost if action FE is chosen in state FE. Similar equations capturing the boundary conditions can be written for the case when  $b_1 = 0$  or  $b_2 = 0$ . Finally, we have  $V(0, 0) = 0$ .

#### 2.3.1 Switchover property

The recursive DP equations in (1) cannot be solved in closed form and a numerical method like value iteration or policy iteration is typically required. However, some interesting structural properties of the optimal policy  $\pi^*$  and the value function  $V(\cdot)$  can be established, which provide intuition and insight into the decision tradeoffs inherent in the problem and also help develop approximate solutions. One particularly interesting property is the optimality of a switchover policy.

*Definition 2.* A policy  $\pi$  is of *switchover* type if there exists a non-decreasing switchover curve  $\psi : \mathbb{Z}_+ \mapsto \mathbb{Z}_+$  such that  $\pi$  chooses action  $\overline{\text{FE}}$  in state  $(b_1, b_2)$  if  $b_2 > \psi(b_1)$  and chooses action FE else.

A switchover policy splits the state-space into two distinct decision regions, one corresponding to each of the actions FE and  $\overline{\text{FE}}$ . The optimal policy of interest here, viz.  $\pi^*$  is of switchover type, in the sense of the definition above.

**THEOREM 1.** *The optimal policy  $\pi^*$  is of switchover type.*

**PROOF.** The proof is based on a combination of value iteration and mathematical induction. We omit the details due to lack of space.  $\square$

*Remark 1.* Theorem 1 is a discrete time analogue of the continuous time result established in [10], where the block transmission times and task execution times are exponentially distributed.

Thus, computing the optimal policy  $\pi^*$  boils down to computing the optimal switchover curve  $\psi^*$ . Even though we

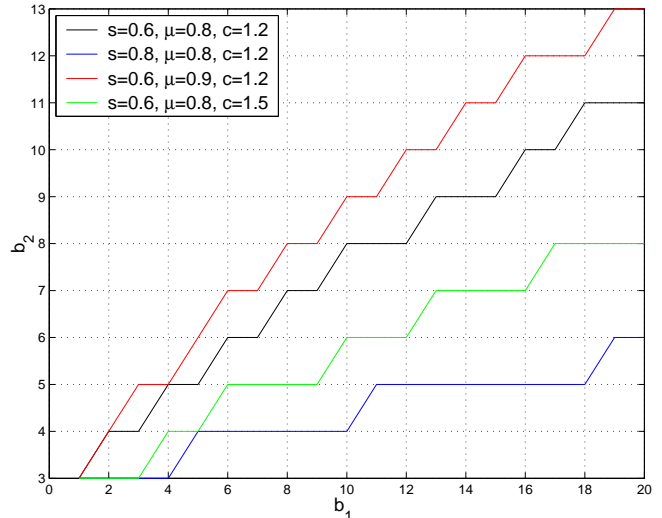


Figure 2: Numerical Example 1

know the structure of  $\psi^*$  by virtue of Theorem 1, the DP equations in (1) need to be solved explicitly to determine  $\psi^*$ . The computations can be quite cumbersome, especially if  $\psi^*$  needs to be recomputed frequently in a dynamically changing environment. We will therefore now focus on a technique to approximate  $\psi^*$  without actually solving any DP equations.

*Numerical Example 1.* This numerical example illustrates the behavior of the optimal switchover curve  $\psi^*$  for different model parameters. Fig. 2 depicts the optimal switchover curve (computed from (1)) for different combinations of  $s$ ,  $\mu$ , and  $c$ . We draw the following observations from the figure:

- The decision region for action FE gets smaller as  $s$  increases for fixed  $\mu, c$ . As the wireless channel becomes more reliable, the MT can afford to fetch blocks from the CS less frequently, since fewer attempts are needed to an average to fetch a block successfully.
- The decision region for action FE grows bigger as  $\mu$  increases for fixed  $s, c$ . Since increasing  $\mu$  decreases average task execution times, the MT has to fetch blocks more regularly from the CS in order to prevent idling of the processor.
- The decision region for action FE gets smaller as  $c$  increases for fixed  $s, \mu$ . A higher value of  $c$  indicates a higher congestion cost (due to background applications) for the foreground applications, forcing the MT to be more frugal in fetching tasks from the CS.

*Remark 2.* While we empirically observed trends in the behavior of  $\psi^*$  as a function of the model parameters, each of the above mentioned properties can be established formally via analytical arguments as well. Doing so, however, is not the focus of this paper.

## 3. APPROXIMATIONS & HEURISTICS

In this section, we will develop a randomized approximation RAND to  $\pi^*$  based on the method of policy iteration,

and leverage RAND to design a decision algorithm fetch-or-not (FON) for dynamic environments. Our first step is to compute the expected cost incurred under two “extreme” policies

1.  $\pi_N$ : Never fetches a task from the CS, until the buffer at the MT is empty, and
2.  $\pi_A$ : Continues to fetch tasks from the CS, until the buffer at the CS has been exhausted.

### 3.1 The “never fetch” policy

Consider a policy  $\pi_N$  which *never* chooses to fetch a task from the CS, except when the buffer at the MT ( $\mathcal{Q}_2$ ) is empty, i.e.,  $\pi_N$  selects action  $\overline{\text{FE}}$  in all states  $(b_1, b_2)$  with  $b_2 > 0$ . We are interested in computing the expected cost incurred under  $\pi_N$  in reaching terminal state  $(0, 0)$ , as a function of the initial state  $(b_1, b_2)$ . Denoting this cost by  $C_N(\mathbf{b})$ , we see that it satisfies:

$$C_N(\mathbf{b}) = \mu C_N(\mathbf{b} - \mathbf{e}_2) + \bar{\mu} C_N(\mathbf{b}) + \langle \mathbf{c}, \mathbf{b} \rangle$$

when  $b_2 > 0$  and

$$C_N(\mathbf{b}) = s\mu C_N(\mathbf{b} - \mathbf{e}_1) + s\bar{\mu} C_N(\mathbf{b} - \mathbf{e}_1 + \mathbf{e}_2) + \bar{s} C_N(\mathbf{b}) + \langle \mathbf{c}, \mathbf{b} \rangle$$

when  $b_2 = 0$ . Rearranging and combining terms we get

$$C_N(\mathbf{b}) = C_N(\mathbf{b} - \mathbf{e}_2) + \frac{\langle \mathbf{c}, \mathbf{b} \rangle}{\mu}$$

$$C_N(b_1, 0) = C_N(b_1 - 1, 0) + \left( \frac{\bar{\mu}}{\mu} + \frac{1}{s} \right) b_1 + \frac{(c-1)\bar{\mu}}{\mu},$$

implying

$$C_N(b_1, 0) = \left( \frac{\bar{\mu}}{\mu} + \frac{1}{s} \right) \frac{b_1^2}{2} + \left[ \frac{1}{2} \left( \frac{\bar{\mu}}{\mu} + \frac{1}{s} \right) + \frac{(c-1)\bar{\mu}}{\mu} \right] b_1. \quad (2)$$

$$C_N(\mathbf{b}) = C_N(b_1, 0) + \frac{b_1 b_2}{\mu} + \frac{c b_2 (b_2 + 1)}{2\mu}. \quad (3)$$

Thus,  $C_N(\mathbf{b})$  is given by (2) and (3) and is a quadratic polynomial in  $b_1$  and  $b_2$ .

### 3.2 The “always fetch” policy

Now, in contrast to  $\pi_N$ , consider a policy  $\pi_A$  which *always* chooses to fetch a task from the CS if available, i.e., it chooses the action FE in all states  $\mathbf{b}$ . We are interested in computing the expected cost incurred under  $\pi_A$  in reaching terminal state  $(0, 0)$ , as a function of the initial state  $(b_1, b_2)$ . We will denote the cost by  $C_A(\mathbf{b})$ . It is not possible to compute  $C_A(\mathbf{b})$  precisely in closed form; therefore we will approximately compute  $C_A(\mathbf{b})$  for a *fluid caricature model*. We will denote the corresponding cost in the fluid model by  $C_A^f(\mathbf{b})$ . The fluid model mimics the “mean behavior” of the discrete time block based model we have been studying so far. The key attributes of the model are:

- $\mathcal{Q}_1$  and  $\mathcal{Q}_2$  buffer infinitesimally divisible fluids instead of discrete blocks.
- Time is continuous instead of being slotted.
- Fluid flows at a constant rate  $s$  from  $\mathcal{Q}_1$  to  $\mathcal{Q}_2$  and flows out of  $\mathcal{Q}_2$  at a constant rate  $\mu$ .
- A backlog cost at unit rate for every unit of fluid is incurred at  $\mathcal{Q}_1$ , and a congestion cost at a rate  $c$  for every unit of fluid is incurred at  $\mathcal{Q}_2$ .

Similar to the discrete-time queuing model, no fluid arrives to  $\mathcal{Q}_1$  after time  $t = 0$ . We will consider two distinct cases:

1.  $s \geq \mu$ : Since  $s \geq \mu$ ,  $\mathcal{Q}_1$  drains faster than  $\mathcal{Q}_2$ . Denoting the initial amount of fluid in  $\mathcal{Q}_1$  and  $\mathcal{Q}_2$  by  $b_1$  and  $b_2$  respectively,  $\mathcal{Q}_1$  first becomes empty at time  $T_0 = b_1/s$  and stays empty thereafter. For  $t \leq T_0$ , the amount of fluid in  $\mathcal{Q}_1$  at time  $t$  is given by  $b_1 - st$ . Thus, the total backlog cost incurred at  $\mathcal{Q}_1$  over the interval  $[0, T_0]$  is  $\int_0^{T_0} (b_1 - st) dt = \frac{b_1^2}{2s}$ . Over the same interval, the amount of fluid in  $\mathcal{Q}_2$  at time  $t$  is given  $b_2 + (s - \mu)t$ . Thus, the total congestion cost incurred at  $\mathcal{Q}_2$  over  $[0, T_0]$  is  $\int_0^{T_0} c(b_2 + (s - \mu)t) dt = \frac{c b_1 b_2}{s} + \frac{c b_1^2}{2s} \left( 1 - \frac{\mu}{s} \right)$ . Now, the amount of fluid in  $\mathcal{Q}_2$  at time  $T_0$  is  $B_2(T_0) = b_2 + b_1 \left( 1 - \frac{\mu}{s} \right)$ . Thus,  $\mathcal{Q}_2$  drains completely at time  $T'_0 = T_0 + \frac{B_2(T_0)}{\mu}$  and the amount of fluid in  $\mathcal{Q}_2$  at time  $t$  for  $t \in [T_0, T'_0]$  is given by  $B_2(T_0) - \mu(t - T_0)$ . Consequently, the congestion cost incurred over the interval  $[T_0, T'_0]$  at  $\mathcal{Q}_2$  is given by  $\int_{T_0}^{T'_0} c(B_2(T_0) - \mu(t - T_0)) dt = \frac{c}{2\mu} \left[ b_2 + b_1 \left( 1 - \frac{\mu}{s} \right) \right]^2$ . The total cost  $C_A^f(\mathbf{b})$  for the case  $s \geq \mu$  is given by the sum of the three costs computed above.

2.  $s < \mu$ : We need to consider two further sub-cases. To this end, define  $T_1 = b_2/(\mu - s)$  and  $T_0 = b_1/s$  as before. If  $T_1 \leq T_0$ , then  $\mathcal{Q}_2$  drains before  $\mathcal{Q}_1$ . The backlog cost incurred at  $\mathcal{Q}_1$  over the interval  $[0, T_1]$  is  $\int_0^{T_1} (b_1 - st) dt = b_1 T_1 - \frac{s T_1^2}{2}$  and the corresponding congestion cost incurred at  $\mathcal{Q}_2$  is  $\int_0^{T_1} c(b_2 - (\mu - s)t) dt = c \left( b_2 T_1 - \frac{(\mu - s) T_1^2}{2} \right)$ . The amount of fluid in  $\mathcal{Q}_1$  at the end of the interval is  $B_1(T_1) = b_1 - s T_1$ . An additional backlog cost of  $\int_{T_1}^{T_0} (B_1(T_1) - st) dt = B_1(T_1)(T_0 - T_1) - \frac{s(T_0 - T_1)^2}{2}$  is incurred over the interval  $[T_1, T_0]$  at  $\mathcal{Q}_1$ . The cost incurred at  $\mathcal{Q}_2$  over this interval is negligible. Thus,  $C_A^f(\mathbf{b})$  for the case  $s < \mu, T_1 \geq T_0$  is given by the sum of the three costs computed above.
- If  $T_1 < T_0$ , then  $\mathcal{Q}_2$  drains before  $\mathcal{Q}_1$ . The computation of  $C_A^f$  in this case is identical to the case  $s \geq \mu$  considered above.

We have now computed  $C_A^f(\mathbf{b})$  as a function of  $\mathbf{b}$ ,  $s$ , and  $\mu$ . We propose to use  $C_A^f(\mathbf{b})$  as an approximation to  $C_A(\mathbf{b})$ , which is the expected cost incurred under policy  $\pi_A$ , starting the system in state  $\mathbf{b}$ . Observe that in all cases,  $C_A^f(\mathbf{b})$  is a quadratic polynomial in  $b_1$  and  $b_2$ .

*Numerical Example 2.* This example illustrates the goodness of  $C_A^f(\mathbf{b})$  as an approximation to  $C_A(\mathbf{b})$ . The left side of Fig. 3 shows the fractional approximation error as a function of  $b_2$  for three different values of  $b_1$  for the case  $s = 0.8$ ,  $\mu = 0.6$  ( $s > \mu$ ). The right side of the figure depicts the

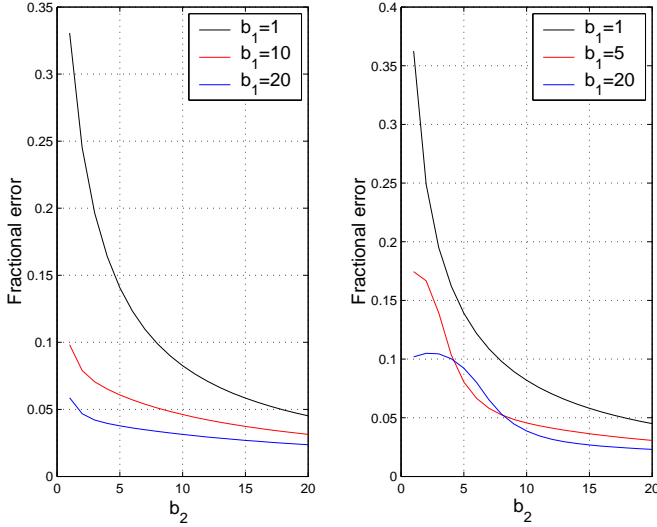


Figure 3: Numerical example 2

same plots for the case  $s = 0.6$ ,  $\mu = 0.8$  ( $s < \mu$ ). Observe that the accuracy of the approximation increases as  $b_1$  and  $b_2$  increase. The relative error is below 5% even for moderately large values of  $b_1$  and  $b_2$ . The error is as much as 30% for small values of  $b_1$  and  $b_2$ . For these cases, however,  $C_A(\mathbf{b})$  can be computed exactly with only a few computations. The “kinks” in the plot on the right correspond to the points at which  $T_1$  exceeds  $T_0$  (as defined in case 2 above). Note that  $T_0$  is fixed since  $b_1$  is fixed on each curve, and  $T_1$  increases linearly with  $b_2$ .

### 3.3 Policy Iteration

*Policy iteration* is a well known numerical technique for iteratively solving Bellman’s equations. Given a feasible policy  $\pi$ , each iteration in policy iteration comprises of two steps:

1. *Policy evaluation*: In this step, the expected cost incurred under policy  $\pi$ , denoted  $V_\pi(\mathbf{b})$ , is evaluated for every state  $\mathbf{b}$ .
2. *Policy improvement*: In this step, the policy  $\pi$  is “improved” to obtain a new policy  $\pi'$ . The cost under policy  $\pi'$  is computed as follows:

$$V_{\pi'}(\mathbf{b}) = \min\{\mu V_\pi(\mathbf{b} - \mathbf{e}_2) + \bar{\mu} V_\pi(\mathbf{b}), s\mu V_\pi(\mathbf{b} - \mathbf{e}_1) + \bar{s}\mu V_\pi(\mathbf{b} - \mathbf{e}_2) + s\bar{\mu} V_\pi(\mathbf{b} - \mathbf{e}_1 + \mathbf{e}_2) + \bar{s}\bar{\mu} V_\pi(\mathbf{b})\} + \langle \mathbf{c}, \mathbf{b} \rangle, \quad (4)$$

The policy  $\pi'$  is called a one step improvement of policy  $\pi$ . We are interested in computing one step improvements of the policies  $\pi_N$  and  $\pi_A$ . We have already performed the policy evaluation step for the two policies in Section 3.1 and Section 3.2 to compute the cost functions  $C_N(\mathbf{b})$  and  $C_A(\mathbf{b})$ , respectively.

Recall that both  $C_N$  and  $C_A$  are quadratic polynomials in  $b_1$  and  $b_2$ . We will therefore compute the one step improvement of an arbitrary policy  $\pi$  with a quadratic cost function of the form:

$$V_\pi(b) = \alpha_1 b_1^2 + \alpha_2 b_2^2 + \gamma b_1 b_2 + \beta_1 b_1 + \beta_2 b_2. \quad (5)$$

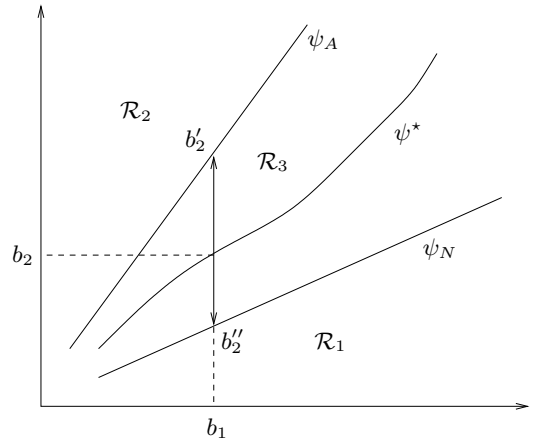


Figure 4: Bounding the optimal switchover curve

For convenience, define

$$V_\pi^1(\mathbf{b}) \triangleq V_\pi(\mathbf{b} + \mathbf{e}_1) - V_\pi(\mathbf{b}) \\ V_\pi^2(\mathbf{b}) \triangleq V_\pi(\mathbf{b} + \mathbf{e}_2) - V_\pi(\mathbf{b}).$$

The policy improvement equation (4) can be rewritten as

$$V_{\pi'}(\mathbf{b}) = \min\{0, s\mu[V_\pi^2(\mathbf{b} - \mathbf{e}_2) - V_\pi^2(\mathbf{b} - \mathbf{e}_1)] + s[V_\pi^2(\mathbf{b} - \mathbf{e}_1) - V_\pi^1(\mathbf{b} - \mathbf{e}_1)]\} + V_\pi(\mathbf{b}) - \mu V_\pi(\mathbf{b} - \mathbf{e}_2). \quad (6)$$

It easily follows from (5) that

$$V_\pi^1(\mathbf{b}) = 2\alpha_1 b_1 + \gamma b_2 + \alpha_1 + \beta_1 \\ V_\pi^2(\mathbf{b}) = \gamma b_1 + 2\alpha_2 b_2 + \alpha_2 + \beta_2. \quad (7)$$

Substituting (7) in (6) gives

$$V_{\pi'}(\mathbf{b}) = \min\{0, \ell_1(\mathbf{b})\} + \ell_2(\mathbf{b}), \quad (8)$$

where  $\ell_1(\mathbf{b})$  and  $\ell_2(\mathbf{b})$  are linear functions of  $b_1$  and  $b_2$ . Note that the decision of policy  $\pi'$  in state  $\mathbf{b}$  is completely determined by the sign of  $\ell_1(\mathbf{b})$ . That is,  $\pi'$  chooses action  $\overline{\text{FE}}$  in state  $\mathbf{b}$  if  $\ell_1(\mathbf{b}) > 0$ , and action FE else. Since  $\ell_1(\mathbf{b})$  is linear (i.e., of the form  $a_1 b_1 + a_2 b_2 + a_3$  for some  $a_1, a_2, a_3 \in \mathbb{R}$ ), the two dimensional state-space  $(b_1, b_2)$  gets split into two distinct decision regions corresponding to the two decisions FE and  $\overline{\text{FE}}$ . In other words, policy  $\pi'$  is of switchover type in the sense of Definition 2. It is easy to show that the slope of the switchover curve which characterizes  $\pi'$  is given by

$$m = \frac{\gamma - 2\alpha_1}{\gamma - 2\alpha_2}. \quad (9)$$

Based on the above analysis and the expressions for cost functions derived in Section 3.1 and 3.2, we can compute the slopes of the switchover curves  $m_A$  and  $m_N$  for the policies  $\pi'_A$  and  $\pi'_N$ , which are one step improvements of the policies  $\pi_A$  and  $\pi_N$ , respectively. We will refer to these two switchover curves as  $\psi_A$  and  $\psi_N$ .

### 3.4 An approximate randomized policy

The switchover curves  $\psi_A$  and  $\psi_N$  are approximations to the optimal switchover curve  $\psi^*$ , which is obtained by solving Bellman’s equations in (1). In fact,  $\psi_A$  bounds  $\psi^*$  from above and  $\psi_N$  bounds  $\psi^*$  from below, as depicted in Fig. 4. Recall that  $\psi_A$  and  $\psi_N$  are straight lines on the  $(b_1, b_2)$  plane.

Roughly speaking, we have bounded the optimal switchover curve in a “conical” region defined by two straight lines. The cone splits the state-space into three distinct regions —  $\mathcal{R}_1$ ,  $\mathcal{R}_2$ , and  $\mathcal{R}_3$ , as depicted in Fig. 4. In region  $\mathcal{R}_1$ , which lies above  $\psi_A$ , the optimal action is  $\overline{\text{FE}}$ , i.e. the MT does not fetch a block from the CS. In region  $\mathcal{R}_3$ , which lies below  $\psi_N$ , the optimal action is FE, i.e. the MT fetches a block from the CS. Region  $\mathcal{R}_2$ , which is the interior of the cone, is a region of uncertainty. Our approximations tell us that the optimal switchover curve  $\psi^*$  lies somewhere in region  $\mathcal{R}_2$ , but not exactly where.

We overcome the uncertainty in region  $\mathcal{R}_2$  by employing a randomized decision mechanism. In particular, consider a state  $\mathbf{b} = (b_1, b_2) \in \mathcal{R}_2$ . We know that  $\exists b'_2 \geq b_2$  such that the state  $(b_1, b'_2)$  lies on the surface of the cone (on line  $\psi_A$ ) and the optimal decision is  $\overline{\text{FE}}$  in all states  $(b_1, y)$  with  $y > b'_2$ . Similarly, we know that  $\exists b''_2 \leq b_2$  such that the state  $(b_1, b''_2)$  lies on the surface of the cone (on line  $\psi_N$ ) and the optimal decision is FE in all states  $(b_1, y)$  with  $y < b''_2$ . If  $(b_1, b_2)$  is closer to  $\psi_A$  than  $\psi_N$ , then the optimal decision is more likely to be  $\overline{\text{FE}}$ , and if  $(b_1, b_2)$  is closer to  $\psi_N$  than  $\psi_A$ , then the optimal decision is more likely to be FE. In particular, for any state  $(b_1, b_2) \in \mathcal{R}_2$ , we will make a randomized decision based on the policy RAND, as described in Table 1.

RAND	
Select action $\overline{\text{FE}}$ w.p.	$\frac{b'_2 - b_2}{b'_2 - b''_2}$
Select action FE w.p.	$\frac{b_2 - b''_2}{b'_2 - b''_2}$

Table 1: RAND: A randomized policy

*Numerical Example 3.* This numerical example illustrates the bounding of the optimal switchover curve  $\psi^*$  in a conical region generated by the switchover curves  $\psi_A$  and  $\psi_N$  for two different sets of parameter values  $s$ ,  $\mu$ , and  $c$ . The results are depicted in Fig. 5. Observe that in both cases the conical region bounds  $\psi^*$  reasonably tightly, especially considering the low complexity involved in computing the conical region due to availability of closed form expressions.

### 3.5 A heuristic algorithm

The optimal policy for the baseline model  $\pi^*$  and its randomized approximation RAND are both applicable to a “static” scenario where the successful transmission probability  $s$ , mean task execution time  $1/\mu$ , and congestion cost rate  $c$  are fixed. However, one expects to encounter dynamic scenarios in a real system where these parameters will vary over time. For instance,  $s$  may vary due to a fluctuating interference from other links active in the same wireless environment in which the CS and MT are operating. In Section 4, we will formally incorporate the dynamics of these static parameters into our modeling framework and solve the resultant DP to obtain the optimal policy for dynamic scenarios. An alternate approach is to leverage the knowledge of  $\pi^*$ , the optimal policy in a static environment, to design a heuristic policy for dynamic environments. In particular,

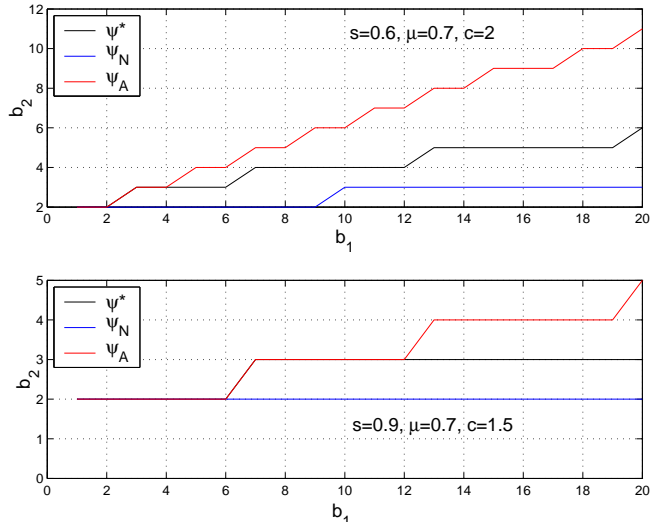


Figure 5: Numerical Example 3

Fetch-or-not (FON)	
In time-slot $t$ ,	
<b>Given</b>	
	estimate of success probability $\hat{s}(t)$ ,
	choice of congestion cost rate $\hat{c}(t)$ ,
	estimate of task execution rate $\hat{\mu}(t)$
<b>Compute</b>	
	$\pi^*$ or RAND with $s = \hat{s}(t)$ , $\mu = \hat{\mu}(t)$ , and $c = \hat{c}(t)$
<b>Select</b> action FE/ $\overline{\text{FE}}$ based on outcome of $\pi^*$ or RAND	
<b>Update</b> $\hat{s}$ , $\hat{\mu}$ , and $\hat{c}$	

Table 2: A heuristic algorithm: FON

we propose algorithm FON, which uses  $\pi^*$  in each time-slot in a *quasi-static* fashion. FON is described in Table 2.

In a real implementation of FON, the updates of  $\hat{s}$  would be based on channel measurements at the MT and the updates for  $\hat{\mu}$  would be based on observations of the execution times of past tasks. The latter information could also be furnished at the application layer. The updates of  $\hat{c}$  would be based on observations of the current and past memory requirements of other concurrently executing applications. A variety of update mechanisms can be envisioned — moving average, sliding window average, fixed step size update etc. Through appropriate and frequent updates of the three parameters which define the baseline model and by employing the “instantaneously optimal” policy  $\pi^*$  (or its low complexity version RAND) for the baseline model, FON can robustly adapt to different dynamic environments.

## 4. MODELING EXTENSIONS

So far, we have invested substantial effort in constructing and analyzing a baseline model for the problem of latency aware buffer management in mobile computing. While the baseline model did not capture all the dynamics in the system, its analysis provided useful insights into the fundamental tradeoffs inherent in the problem and also helped us design a low complexity heuristic algorithm with easily tunable parameters. The baseline model can be extended significantly to incorporate a variety of other system dy-

namics. The enhanced models are also amenable to analysis in a dynamic programming framework. We discuss a few such extensions here.

## 4.1 Time-varying wireless channel

In the baseline model we assumed that the wireless channel from the CS to the MT is an i.i.d. Bernoulli process, i.e., it is ON in every time-slot w.p.  $s$  and OFF w.p.  $1 - s$ , independent of past/future time-slots. In practice, the evolution of a wireless channel could be correlated across time and is therefore more accurately modeled as a finite state Markov chain (FSMC). Each state of the FSMC represents a different probability of successful block transmission from the CS to the MT. Denote the current channel state by  $m \in \{1, \dots, M\}$ , the success probability in state  $m$  by  $s(m)$ , and the state transition matrix by  $P = \{P(m, m')\}$ . The channel state is appended to the system state  $\mathbf{b}$  to get the new system state  $(\mathbf{b}, m)$ . The problem of minimizing the expected block holding costs at  $\mathcal{Q}_1$  and  $\mathcal{Q}_2$  until both buffers are drained is an SSP problem as before, whose associated value function satisfies the following Bellman's equations:

$$V(\mathbf{b}, m) = \min\left\{\sum_{m=1}^M P(m, m')[\mu V(\mathbf{b} - \mathbf{e}_2, m') + \bar{\mu} V(\mathbf{b}, m')], \sum_{m=1}^M P(m, m')[s(m)\mu V(\mathbf{b} - \mathbf{e}_1, m') + \bar{s}(m)\mu V(\mathbf{b} - \mathbf{e}_2, m') + s(m)\bar{\mu} V(\mathbf{b} - \mathbf{e}_1 + \mathbf{e}_2, m') + \bar{s}(m)\bar{\mu} V(\mathbf{b}, m')]\right\} + \langle \mathbf{c}, \mathbf{b} \rangle, \quad (10)$$

where  $\bar{s}(m) \equiv 1 - s(m)$ .

*Remark 3.* Note that the baseline model is a special case of the above formulation with  $M = 1$ . The channel behavior can be captured in sufficient detail by selecting appropriately the number of channel states  $M$ .

### 4.1.1 Power control at the CS

Power control is an important issue in an interference limited wireless environment [1]. With power control, the success probability becomes a function of the channel state  $m$  as well as transmit power  $p$ , and is denoted by  $s(p, m)$ . In each time-slot, the CS may choose a different transmit power level to modulate the probability of successful block transmission to the MT, depending upon the wireless channel condition and the queue backlogs. Since the CS “stresses” the wireless environment by transmitting at a high power, a *power cost*  $\rho(p)$  is imposed if the CS chooses to transmit at power  $p$  in a time-slot. Here  $\rho : \mathbb{R}_+ \mapsto \mathbb{R}_+$  could be any non-negative, non-increasing, convex function.

Always transmitting at a low power will increase the application latency, while always transmitting at high power will invite a huge power cost. Thus, introducing power control into the formulation creates a three way dilemma between interference management, application latency, and memory management.

The state of the system is again  $(\mathbf{b}, m)$ . The decisions to be made are — to *fetch* or not to fetch, and if yes, at what power? The decision  $\overline{\text{FE}}$  can be thought of as a special case of decision FE, executed at zero transmit power. The problem is once again amenable to analysis in a DP framework

and the associated value function satisfies:

$$V(\mathbf{b}, m) = \inf_{p \geq 0} \left\{ \sum_{m=1}^M P(m, m')[s(p, m)\mu V(\mathbf{b} - \mathbf{e}_1, m') + \bar{s}(p, m)\mu V(\mathbf{b} - \mathbf{e}_2, m') + s(p, m)\bar{\mu} V(\mathbf{b} - \mathbf{e}_1 + \mathbf{e}_2, m') + \bar{s}(p, m)\bar{\mu} V(\mathbf{b}, m')] + \rho(p) \right\} + \langle \mathbf{c}, \mathbf{b} \rangle, \quad (11)$$

with  $s(0, m) = 0 \forall m$ . Further constraints such as a peak power constraint ( $p \leq p_0$ ), finite number of power levels ( $p \in \{p_1, \dots, p_N\}$ ) etc. can be easily incorporated into this optimization framework.

*Remark 4.* If the power cost were zero, clearly the optimal fetching policy would always fetch blocks at maximum power (as dictated by the peak power constraint), if it chose to fetch. This was the case in [10], which is why the optimal policy the authors obtained was of *bang bang* type.

## 4.2 Time-varying congestion

So far we have worked with the assumption that applications concurrently being executed on the MT cause “fixed” background congestion for the foreground application of interest. We modeled this effect via the congestion cost parameter  $c$ . A small value of  $c$  indicates a small congestion cost, allowing the foreground application to use a larger fraction of the shared resource, viz. the memory, and vice-versa. In a dynamic computational environment, the congestion experienced by the foreground application will vary over time. In other words, the cost parameter  $c$  will vary. To be consistent with our DP based approach, we can model the congestion as an FSMC with state  $j \in \{1, \dots, J\}$  and state transition matrix  $Q = \{Q(j, j')\}$ . The congestion cost in state  $j$  is denoted  $c(j)$ . Assuming a static channel and no power control for simplicity, the enhanced system state is given by  $(\mathbf{b}, j)$ . The value function in this case satisfies:

$$V(\mathbf{b}, j) = \min\left\{\sum_{j=1}^J Q(j, j')[\mu V(\mathbf{b} - \mathbf{e}_2, j') + \bar{\mu} V(\mathbf{b}, j')], \sum_{j=1}^J Q(j, j')[s\mu V(\mathbf{b} - \mathbf{e}_1, j') + \bar{s}\mu V(\mathbf{b} - \mathbf{e}_2, j') + s\bar{\mu} V(\mathbf{b} - \mathbf{e}_1 + \mathbf{e}_2, j') + \bar{s}\bar{\mu} V(\mathbf{b}, j')]\right\} + \langle \mathbf{c}(j), \mathbf{b} \rangle, \quad (12)$$

where  $\mathbf{c}(j) = (1 \ c(j))$ . Observe that the baseline model is a special case of the above formulation with  $J = 1$ .

## 4.3 Other extensions

### 4.3.1 Dynamic task arrivals

In all the models we have considered thus far, baseline as well as enhanced, we have assumed that no further tasks (blocks) arrive to  $\mathcal{Q}_1$  at the CS after  $t = 0$ . This has permitted us to formulate our optimal control problems as stochastic shortest path problems, with  $\mathbf{b} = \mathbf{0}$  being the natural terminal backlog state of the system. Within a DP framework, we can also treat a scenario where tasks dynamically arrive to the queue at the CS according to some stochastic process. In this case, a more natural formulation is to minimize the expected discounted cost over an infinite horizon or to minimize the average cost per time-slot. Any Markovian arrival

process can be studied in this setup. For instance, we can consider an arrival process governed by an FSMC with state  $k \in \{1, \dots, K\}$  and state transition matrix  $R = \{R(k, k')\}$ , such that in state  $k$ , a block arrives to  $\mathcal{Q}_1$  w.p.  $\lambda(k)$ . Processes which allow more than one block arrival per time-slot can also be considered. We suppress the DP equations for this case in the interest of brevity.

### 4.3.2 Time-varying service processes

The assumption of geometrically distributed task execution times at the MT pervades all the models described above. The assumption implies that in every time-slot, the probability that a task completes execution given that it did not finish execution in the previous time-slot is  $\mu$ . We can relax this assumption and allow  $\mu$  to be modulated by an FSMC with state  $l \in \{1, \dots, L\}$  and state transition matrix  $S = \{S(l, l')\}$ . If the FSMC is in state  $l$ , the probability that a task completes execution given that it did not finish execution in the previous time-slot is  $\mu(l)$ . We once again suppress the DP equations for this case in the interest of space.

*Remark 5.* In principle, it is possible to combine all the elements discussed above into a joint model which captures the system dynamics in great detail. The state of the system under the joint model is  $(\mathbf{b}, m, j, k, l)$  and the dynamics are governed by the transition matrices  $P, Q, R$ , and  $S$ . Such a model, however, is tremendously complicated to analyze or even simulate. Also, the optimal policy for such a complicated model is very hard to approximate parsimoniously due to its dependence on a multitude of parameters. In contrast, the baseline model is simple yet insightful, and the decision algorithm FON developed on the basis of the optimal policy  $\pi^*$  for the baseline model can be implemented with very low complexity.

## 5. CONCLUSIONS & FUTURE WORK

This paper examined buffer management at mobile terminals in a mobile computing environment. A single mobile terminal which sequentially requests computational tasks from a central server over an unreliable wireless channel was considered. While buffering tasks locally at the mobile terminal is advantageous from a latency perspective, memory is a precious and limited resource at the terminal, which is shared by multiple applications. This tradeoff, in conjunction with the time varying nature of the wireless channel, gives rise to the dilemma at the mobile terminal — “to fetch or not to fetch” a task from the central server at any given instant. This decision problem was formulated as an optimal control problem for a two queue tandem and studied in a dynamic programming framework. Key structural properties of the optimal policy were leveraged to construct a low complexity randomized approximation for a static environment, and eventually a practical heuristic algorithm (FON) applicable to dynamic environments. Several relevant modeling extensions (e.g., power control) of the baseline model were also considered.

The simplified two queue tandem abstraction studied here clearly demonstrates the latency vs. buffer tradeoff which arises in mobile computing scenarios involving resource limited mobile terminals, while keeping the model complexity and analysis manageable. While control of a two queue tandem has been examined in theoretical settings in the liter-

ature, this paper provides a way of approximating the optimal policy in closed form, which is otherwise can only be computed numerically.

Our ongoing research efforts involve a thorough experimental evaluation of FON in a variety of realistic settings, as well as contrasting its performance to state-of-the-art benchmarks. On the theoretical front, we are analyzing in detail various modeling extensions described in the paper to establish insightful structural properties of the associated optimal controls.

## 6. REFERENCES

- [1] N. Bambos. Toward power-sensitive networks architectures in wireless communications: concepts, issues and design aspects. *IEEE Pers. Commun. Mag.*, 5(3):50–59, Jun. 1998.
- [2] D. Bertsekas. *Dynamic Programming and Optimal Control*. Athena Scientific, Belmont, MA, second edition, 2000.
- [3] G. Cao. Proactive power-aware cache management for mobile computing systems. *IEEE Trans. Computers*, 51(6):608–621, Jun. 2002.
- [4] A. Dua and N. Bambos. Buffer management for wireless media streaming. Technical Report SU-Netlab-2007-03/01, Stanford University Library, Stanford, CA, Mar. 2007.
- [5] S. Gitisen and N. Bambos. Power-controlled data prefetching/caching in wireless packet networks. In *Proc. IEEE Infocom'02*, pages 1405–1414. IEEE, Jun. 2002.
- [6] B. Hajek. Optimal control of two interacting service stations. *IEEE Trans. Autom. Control*, 29(6):491–499, Jun. 1984.
- [7] M. Ip, W. Lin, A. Wong, T. Dillon, and D. Wang. An adaptive buffer management algorithm for enhancing dependability and performance in mobile-object-based real-time computing. In *Proc. IEEE ISORC'00*, pages 138–144. IEEE, May 2001.
- [8] M. Kalman, E. Steinbach, and B. Girod. Adaptive media playout for low-delay video streaming over error-prone channels. *IEEE Trans. Circuits Syst. Video Technol.*, 14(6):841–851, Jun. 2004.
- [9] Y. Li, A. Markopoulou, J. Apostolopoulos, and N. Bambos. Joint power-playout control for media streaming over wireless links. *IEEE Trans. Multimedia*, 8(4):830–843, Aug 2006.
- [10] Z. Rosberg, P. Varaiya, and J. Walrand. Optimal control of service in tandem queues. *IEEE. Trans. Autom. Control*, 27(3):600–610, Jun. 1982.
- [11] K. Schiefermayr and J. Weichbold. A complete solution for the optimal stochastic scheduling of a two-stage tandem queue with two flexible servers. *Journal Appl. Prob.*, 42(3):778–796, 2005.
- [12] J. Walrand. *An Introduction to Queueing Networks*. Prentice Hall, Englewood Cliffs, NJ, 1988.
- [13] S. Yokoyama, T. Okuda, T. Mizuno, and T. Watanabe. A memory management architecture for a mobile computing environment. In *Proc. IEEE ICPADS'00*, page 23. IEEE, Jul. 2000.