

Fast and Optimal Scheduling over Multiple Network Interfaces

Matei A. Zaharia (matei@matei.ca)

Advised by Srinivasan Keshav (keshav@uwaterloo.ca)

University of Waterloo

May 15, 2007

Abstract

Most of today's mobile devices have at least two wireless network interface cards (NICs). Future devices are expected to have as many as twelve wireless NICs, with different data transmission capacities and a wide range of energy and dollar costs. When multi-NIC devices are running multiple applications, each unit of application data ought to be scheduled on to the network interface that maximizes user satisfaction. The scheduling algorithm must take into account not only user and application preferences, such as user cost-tolerance and application delay-tolerance, but also future availability of NICs due to device mobility. Furthermore, to run on small, resource-limited devices, it must be highly efficient.

We addressed the problem of efficiently computing an optimal transmission schedule if future connectivity is known. We showed that simple greedy algorithms are incorrect and that classical optimization techniques are too inefficient for CPU- and memory-constrained mobile devices. We developed a hill-climbing algorithm that finds optimal schedules 10-100 times faster than classical techniques by exploiting problem structure, and is therefore efficient enough to employ on a mobile device. Our algorithm also supports efficient incremental rescheduling. Finally, we also implemented our algorithm on a real system that runs on Java-enabled laptops, PDAs and cell phones.

Introduction

Mobile devices have multiple network interfaces with varying characteristics, such as bandwidth, dollar cost, and power cost, which become available intermittently as the user travels by access points. As technologies such as WiMAX become widely deployed, more types of networks will be available. Many mobile applications are delay-tolerant, so scheduling future transmissions can save money and power. For example, a cell phone user might be willing to delay sending a non-urgent email message for up to an hour if this means sending it over a low-cost interface. Ideally, users ought to be able to specify high-level goals (e.g. “send this email urgently”), and the device should schedule communications automatically to maximize utility.

We addressed the problem of computing an optimal schedule assuming future connectivity is known. Although this assumption is strong, we believe that it is not limiting. Previous research has shown that mobile users have surprisingly predictable schedules [6], so past history and location lead to good predictions. Furthermore, if a user is not following a predicted schedule, we can reschedule according to a more pessimistic outlook. Finally, our work provides a first step towards scheduling with uncertainty. For example, stochastic optimization [12] can be used to find schedules given a probability distribution over future NIC states.

Related Work

Policy-based selection of network interfaces was first introduced in [8], and has been explored in the context of *vertical handoffs* [9, 4], where the device selects the network that optimizes data rates and power consumption while preserving seamless connectivity. This formulation, however, uses only one NIC at any time and does not maximize user utility by exploiting delay-tolerance. Scheduling over multiple interfaces has been studied for cellular base stations [2], which must service multiple users efficiently and fairly. Our work differs because it schedules delay-tolerant data at the mobile device itself, over a much larger future window. Local search has been used in various NP-complete optimization problems [1]. Our algorithm differs because it is optimal – we use local search for efficiency. Finally, we use a step size technique based on simulated annealing [3].

Mathematical Model

We fragment messages into fixed-size *bundles*, and divide the periods of predicted uptime of each NIC into *time slots*, each of duration equal to the time it takes to transmit a bundle over that interface. We categorize messages into several *service classes*, each with a *utility function* that decreases over time. This function represents the “time value of data” - how much more useful it is to send the message at an earlier time. Finally, we assign a *cost* for sending a bundle over each interface.

The problem is then to assign a set of bundles to time slots so as to maximize total utility. The utility of a schedule is the sum of the utilities of each bundle at the time it was sent, minus the costs incurred.

We use the following notation: K = number of classes, L = number of NICs, N = number of bundles, T = number of time slots, $u_i(t)$ = utility of a bundle of class i at time t , $U(\mathcal{S})$ = utility of a schedule $\mathcal{S}$.

We make one assumption about the utility functions, motivated by real problems. We assume that the u_i ’s are decreasing, with rates of decrease that are consistently *ordered*, i.e. $u'_1(t) \leq u'_2(t) \leq \dots \leq u'_K(t) \leq 0$. This means that the “relative urgency” of each class remains constant: bundles that are losing utility quickly “now” will continue to lose utility quickly in the future. This allows for a variety of utility functions, including sets of linear functions $\{u_i(t) = a_i - b_i t\}$. We have also generalized our algorithm to functions whose slope ordering changes at a small number of points in time.

The main disadvantage of our model is that utility is calculated per bundle. One could argue that users gain no utility from a fraction of a message. However, we believe that this is not a problem in practice, for several reasons. First, streaming applications, such as video, generally *do* provide utility for each bundle. Second, applications where complete messages must be sent, such as email, often have messages that fit into just one bundle. Finally, many types of data can be sent progressively, so having even the first few bundles of a message provides utility. For example, JPEG and PNG images begin with a low-quality version.

Classical Approaches

We showed that three naive greedy algorithms are incorrect: Highest Utility First, Earliest Deadline First, and Fastest Decreasing Utility First. These fail to find optimal schedules even over a single NIC.

We also formulated the problem using two classical optimization techniques: linear programming and min-cost flow, by reducing scheduling to an assignment problem. Although these techniques find optimal schedules, they are too inefficient to run on CPU- and memory-constrained mobile devices.

Hill-Climbing Algorithm

Our algorithm is based on two observations about what characterizes “good” schedules:

- Because utility is decreasing, it is never beneficial to leave a time slot empty on a NIC, then use a later slot on the same NIC. Any schedule that does so can be improved by moving the later bundle earlier.
- Because relative urgencies remain constant, it is never beneficial to send a less urgent bundle before a more urgent bundle. Any schedule that does so can be improved by swapping the two bundles.

We defined a *simple schedule* as one in which neither of these situations occurs. That is, there are no unused slots followed by non-empty slots on the same NIC, and bundles are transmitted in order of urgency.

We showed that it is sufficient to consider simple schedules in a search for the optimum. First, we note that a simple schedule is uniquely determined by the number of bundles of each class that it sends and the number of slots it employs on each NIC. We showed that for any schedule $\mathcal{S}$, the simple schedule with the same values for these numbers as $\mathcal{S}$, called the *simplification* of $\mathcal{S}$, has greater or equal utility than $\mathcal{S}$. Thus, there exists an optimal simple schedule, and we only need to search over simple schedules.

We developed an algorithm for finding an optimal schedule by *hill-climbing* in the space of simple schedules, starting from any schedule and gradually improving it by moving to a “neighbouring” schedule with higher utility until we reach a local maximum. We proved that, in fact, this method finds a *global* maximum.

Define a *neighbour* of a simple schedule $\mathcal{S}$ as a schedule $\mathcal{S}'$ such that $\mathcal{S}'$ can be obtained from $\mathcal{S}$ by:

- Either moving a bundle from one interface to another, or changing the contents of a single slot (adding a bundle into an empty slot, or removing a bundle, or changing the type of a bundle).
- Simplifying the resulting schedule.

We proved that any non-optimal simple schedule $\mathcal{S}$ has a simple neighbour $\mathcal{S}'$ with $U(\mathcal{S}') > U(\mathcal{S})$. This implies that the hill-climbing algorithm will terminate and will find a globally optimal schedule.

We greatly increased performance of hill-climbing using an optimization similar to simulated annealing [3]. Instead of modifying the schedule by adding/moving/removing one bundle per iteration to generate neighbours, modify it by acting on larger groups of k bundles, and decrease k over time. For example, first try adding/moving/removing groups of $k = 1024$ bundles at a time until no improvement can be made, then groups of 512 bundles, then 256 bundles, etc. Formally, our scheduling algorithm is as follows:

```

for  $k = 2^n, 2^{n-1}, 2^{n-2}, \dots, 4, 2, 1$  do
  while  $\exists$  neighbour  $\mathcal{S}'$  of  $\mathcal{S}$ , obtained by modifying  $k$  bundles, with  $U(\mathcal{S}') > U(\mathcal{S})$  do
     $\mathcal{S} = \mathcal{S}'$ 
  end while
end for

```

On each iteration, we must evaluate all neighbours of the current schedule, which are up to $O(K^2 + L^2)$ in number. We showed that the utility of a simple schedule can be evaluated in $O(KL \log^2 T)$ time, with $O(KLT)$ precalculation. Furthermore, if the time slots have additional structure, such as forming some small number M of contiguous intervals, this can be reduced to $O(KL \log T \log M)$.

The actual number of iterations depends on the problem. Empirically, on random problems, the number of iterations is proportional to $\log N$. For example, on a problem size of 10000 bundles, the algorithm rarely requires more than 200 iterations. Because the utility of $\mathcal{S}$ always increases, the number of iterations is bounded by the number of simple schedules, which is $O(N^{K+L})$.

We compared the performance of our algorithm with the commercial packages CPLEX [10], for the simplex method, and CS2 [11], for network flow. On realistic problem sizes (~ 5000 bundles), with a variety of values of K and L and randomly generated time slot times, hill-climbing is approximately 10x faster than network flow and 30x faster than CPLEX. We also evaluated performance on a 200 MHz Java smartphone, where a 5000-bundle problem takes 913ms. If the time slots form contiguous intervals, faster performance is possible.

Finally, unlike classical network flow and simplex approaches, our algorithm is ideally suited for incremental updates, which allows it to be efficiently re-run in response to packet arrivals. One can simply begin hill-climbing from the previous schedule when a new message is submitted to the scheduler.

Contributions

To our knowledge, this work represents the first attempt at computing optimal transmission schedules for multi-NIC devices. Our algorithm lets a device intelligently use transmission opportunities on multiple NICs without user intervention. We believe that such an approach is crucial for any realistic usage of multi-NIC devices. Our algorithm is computationally efficient, provably optimal, and also suited for incremental updates. Finally, we have not only mathematically studied the problem, but also implemented our algorithm on a real system that runs on laptops and cell phones [5], demonstrating its use in realistic conditions.

References

- [1] E. Aarts and J.K. Lenstra, *Local Search in Combinatorial Optimization*, J. Wiley, 1997.
- [2] M. Andrews. “A survey of scheduling theory in wireless data networks,” Proc. 2005 IMA Summer Workshop on Wireless Communications.
- [3] S. Kirkpatrick, C.D. Gelatt Jr, M.P. Vecchi, “Optimization by Simulated Annealing,” Science, 1983
- [4] T. Pering, Y. Agarwal, R. Gupta, R. Want, “CoolSpots: Reducing Power Consumption Of Wireless Mobile Devices Using Multiple Radio Interfaces,” Proc. ACM/USENIX MOBISYS, 2006.
- [5] A. Seth, D. Kroeker, M. Zaharia, S. Guo, and S. Keshav, “Low-cost Communication for Rural Internet Kiosks Using Mechanical Backhaul,” Proc. ACM MOBICOM, 2006.
- [6] V. Srinivasan, M. Motani, and W. Ooi, “Analysis and Implications of Student Contact Patterns Derived from Campus Schedules,” Proc. ACM MOBICOM, 2006
- [7] M. Stemm and R. Katz, “Vertical Handoffs in Wireless Overlay Networks,” In Mobile Networks and Applications, Volume 3, Number 4, Pages 335-350, 1998.
- [8] H. Wang, R. Katz, and J. Giese, “Policy-Enabled Handoffs across Heterogeneous Wireless Networks,” In Mobile Computing Systems and Applications, 1999.
- [9] F. Zhu and J. McNair, “Optimizations for Vertical Handoff Decision Algorithms,” Proc. WCNC, 2004.
- [10] ILOG CPLEX, <http://www.ilog.com/products/cplex/>
- [11] Andrew Goldberg’s Network Optimization Library, <http://www.avglab.com/andrew/soft.html>
- [12] Stochastic Optimization - Wikipedia, http://en.wikipedia.org/wiki/Stochastic_optimization