Fast and Optimal Scheduling over Multiple Network Interfaces

Matei A. Zaharia (matei@matei.ca)
Advised by Srinivasan Keshav (keshav@uwaterloo.ca)
University of Waterloo

Problem and Motivation

Most of today's mobile devices have at least two wireless network interface cards (NICs) - cellular and WiFi. As technologies such as WiMAX become widely deployed, more types of networks will become available. 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 [1]. When multi-NIC devices are running multiple applications, each unit of application data ought to be scheduled on the network interface that maximizes user satisfaction. For example, a cellphone user might be willing to delay sending a photograph for up to an hour if this means sending it over a low-cost network interface. Scheduling wireless transmissions thoughtfully can save hours of battery life and hundreds of dollars. Today, users schedule transmissions manually, by turning network interfaces on and off. Ideally, users ought to be able to specify high-level goals (e.g. "send this email urgently" or "send this photo by tomorrow"), and the device should schedule communications automatically to maximize utility. 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, in order to run on small, resource-limited devices, the scheduling algorithm must be highly efficient.

As a first step towards automatic scheduling, we address 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 [7], so past history combined with the user's current location can lead to good predictions. In addition, if a user is not following a predicted schedule, we can always reschedule according to a more pessimistic outlook. Finally, our work can be a useful building block for a scheduling with uncertainty. For example, stochastic optimization [15] can be used to find schedules given a probability distribution over future NIC states, if we have an efficient algorithm for finding the optimal schedule for a fixed connectivity scenario.

In this work, we first showed that simple greedy algorithms for the scheduling problem 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 globally 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 as new data is queued to be sent. Finally, we have implemented our algorithm in a real system that runs on Java-enabled laptops, PDAs and cell phones.

Background and Related Work

To our knowledge, the growing importance of multi-NIC devices was first articulated by Bahl et al [3]. Policy-based selection of network interfaces was first introduced in [8], and has been explored in the context of *vertical*

handoffs [10,9,11], 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 [4], 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.

Intelligent selection of network interfaces with session persistence is also being explored in the Haggle project [12]. However, Haggle is focused on infrastructure-less systems where devices communicate with each other in an ad-hoc manner. Further, the scheduling decisions made by Haggle's Resource Manager do not take into account future connectivity patterns, as we do.

In the algorithms literature, linear programming and network flow are widely-used optimization techniques. Local search has been used in various NP-complete optimization problems [2]. Our algorithm differs because it is optimal - we use hill-climbing for efficiency. Our decreasing step size technique is inspired by simulated annealing [5].

Approach and Uniqueness

In this section, we cover our formal model for the problem, our investigations showing that classical approaches are inadequate, and our novel and provably optimal hill-climbing algorithm.

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(*S*)= utility of a schedule *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. $u_1'(t) \leq u_2'(t) \leq ... \leq u_K'(t) \leq 0$. 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, such as JPEG and PNG images, can be sent progressively, so having even the first few bundles of a message provides utility.

Classical Approaches

We showed that three naive greedy algorithms are incorrect: Highest Utility First, Earliest Deadline First, and Most Urgent 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 there always exists an optimal schedule which is simple, so it is sufficient to search within the state of 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.

Our approach is unique because it shows that in the scheduling problem, a local search strategy in a carefully selected search space can lead to a global optimum. Effectively, by considering the space of simple schedules, we make the problem convex.

A schedule's "neighbours," introduced above, are intuitively defined: they are those simple schedules obtained by adding, moving, or removing a single bundle at a time from one network interface to another, or changing the type of a single bundle, within the original schedule. There are $O(K^2+L^2)$ neighbours, and each of them can be explored quickly, in $O(K\ L\ \log^2 T)$ time with an $O(KLT)$ precalculation.

We also greatly increased performance of this basic hill-climbing technique by an *decreasing step size* technique similar to simulated annealing [5]. Instead of modifying the schedule by adding/moving/removing one bundle per iteration to generate neighbours, we 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. In practice this reduces the number of iterations to about O(logN), while still finding the optimal schedule. For example, on random 10000-bundle problems, the algorithm rarely requires more than 200 iterations.

Finally, unlike classical network flow and linear programming 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.

Results and Contributions

We compared the performance of our algorithm with two state-of-the-art commercial optimization packages: CPLEX [13], for linear programming, and CS2 [14], for network flow. We ran these tests on a high-performance SGI Altix 3700 server with 64 1.4 GHz Intel Itanium2 CPUs. On realistic problem sizes (N ≈ 5000 bundles), with a variety of values of K and L and randomly generated uptime slot times, hill-climbing is approximately 10-20x faster than network flow and 30-100x faster than CPLEX. For example, figure 1 plots the performance of our algorithm versus linear programming (CPLEX) and network flow (CS2) for 10 service classes and 5 NICs.

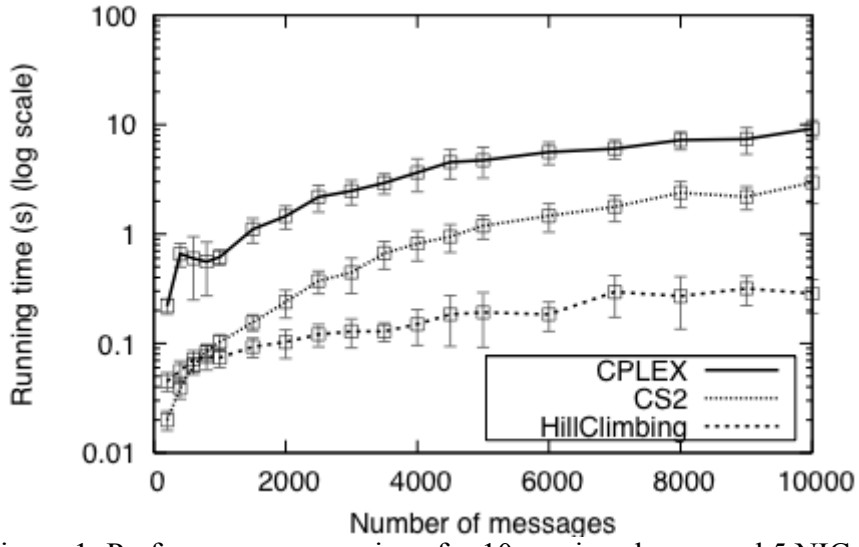

Figure 1: Performance comparison for 10 service classes and 5 NICs.

We also implemented our algorithm within the Java-based OCMP framework for opportunistic communication [6], which can run on cellphones, PDAs and laptops. On a 200 MHz iMate smartphone, a 5000-bundle problem takes 913 ms with an untuned Java implementation. With a faster smartphone (400 MHz is common today), or using native code, an additional factor of two could easily be extracted. Finally, when we run OCMP on a 2.8 GHz Pentium 4 laptop instead of a cellphone, the same problem is solved in 25 ms, far less than the latency in many long-range wireless networks.

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. We have also implemented our algorithm on a real system that runs on laptops and cellphones, demonstrating its use in realistic conditions.

References

[1]

Intel PXA270 Processor Based Reference Design, `http://www.embeddedintel.com/catalog/datasheet.php?ds=5`, March 2007.

[2]

E. Aarts and J.K. Lenstra, *Local Search in Combinatorial Optimization*, J. Wiley, 1997.

[3]

V. Bahl, A. Adya, J. Padhye, and A. Wolman, "Reconsidering the Wireless LAN Platform with Multiple Radios," ACM SIGCOMM FDNA Workshop, 2003.

[4] M. Andrews. "A survey of scheduling theory in wireless data networks," Proc. 2005 IMA Summer Workshop on Wireless Communications.

[5] S. Kirkpatrick, C.D. Gelatt Jr, M.P. Vecchi, "Optimization by Simulated Annealing," Science, 1983

[6] 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.

[7] V. Srinivasan, M. Motani, and W. Ooi, "Analysis and Implications of Student Contact Patterns Derived from Campus Schedules," Proc. ACM MOBICOM, 2006

[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] M. Stemm and R. Katz, "Vertical Handoffs in Wireless Overlay Networks," In Mobile Networks and Applications, Volume 3, Number 4, Pages 335-350, 1998.

[11] 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.

[12] J. Scott, J. Crowcroft, P. Hui, C. Diot, "Haggle: A Networking Architecture Designed Around Mobile Users," Conference on Wireless On-demand Network Systems and Services (WONS), 2006.

[13] ILOG CPLEX, http://www.ilog.com/products/cplex/

[14] Andrew Goldberg's Network Optimization Library, http://www.avglab.com/andrew/soft.html

[15] Stochastic Optimization - Wikipedia, http://en.wikipedia.org/wiki/Stochastic_optimization