Energy-Delay Tradeoffs in Smartphone Applications*

Moo-Ryong Ra† Jeongyeup Paek† Abhishek B. Sharma†
Ramesh Govindan† Martin H. Krieger* Michael J. Neely*
Computer Science Dept.† School of Policy, Planning, and Development* Electrical Engineering Dept.*
University of Southern California, Los Angeles, CA, USA
{mra, jpaek, absharma, ramesh, krieger, mjneely} @ usc.edu

ABSTRACT

Many applications are enabled by the ability to capture videos on a smartphone and to have these videos uploaded to an Internet-connected server. This capability requires the transfer of large volumes of data from the phone to the infrastructure. Smartphones have multiple wireless interfaces – 3G/EDGE and WiFi – for data transfer, but there is considerable variability in the availability and achievable data transfer rate for these networks. Moreover, the energy costs for transmitting a given amount of data on these wireless interfaces can differ by an order of magnitude. On the other hand, many of these applications are often naturally delay-tolerant, so that it is possible to delay data transfers until a lower-energy WiFi connection becomes available. In this paper, we present a principled approach for designing an optimal online algorithm for this *energy-delay tradeoff* using the Lyapunov optimization framework. Our algorithm, called SALSA, can automatically adapt to channel conditions and requires only local information to decide whether and when to defer a transmission. We evaluate SALSA using real-world traces as well as experiments using a prototype implementation on a modern smartphone. Our results show that SALSA can be tuned to achieve a broad spectrum of energy-delay tradeoffs, is closer to an empirically-determined optimal than any of the alternatives we compare it to, and, can save 10-40% of battery capacity for some workloads.

Categories and Subject Descriptors

C.4 [**Performance of Systems**]: Design Studies—*Energy Management on Smartphones*

*This research was sponsored by the USC/CSULB METRANS Transportation Center and by the Army Research Laboratory under Cooperative Agreement Number W911NF-09-2-0053. The views and conclusions contained in this document are those of the authors and should not be interpreted as representing the official policies, either expressed or implied, of the METRANS center, the Army Research Laboratory or the U.S. Government. The U.S. Government is authorized to reproduce and distribute reprints for Government purposes notwithstanding any copyright notation hereon. In addition, the first author, Moo-Ryong Ra, was supported by Annenberg Graduate Fellowship.

MobiSys'10, June 15–18, 2010, San Francisco, California, USA.

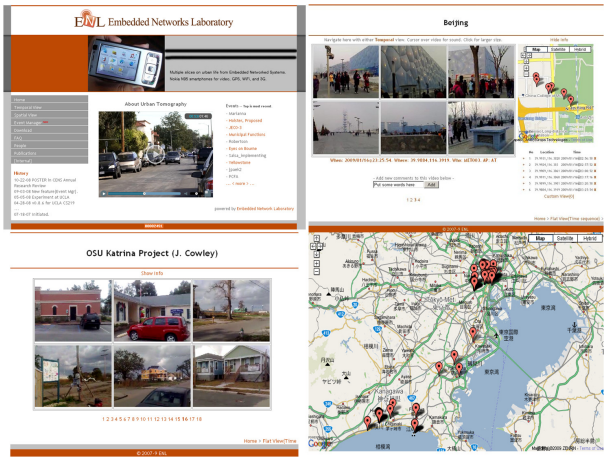

Figure 1: Urban Tomography System

General Terms

Algorithms, Design, Experimentation, Measurement, Theory, Performance

Keywords

WiFi, Interface Selection, Smartphone, Lyapunov Optimization

1. INTRODUCTION

As video-enabled smartphones become more prevalent, many new and interesting applications will be enabled. Our Urban Tomography system [25, 13] is a good example. It allows a user to capture video clips, and then automatically uploads them in the background to a server. The system has been operational for over a year and has found several, qualitatively different, uses. A team of security officials, equipped with smartphones, has been using it for surveillance at a large transportation hub in Los Angeles. The team is able to visually document parts of the facility not covered by fixed cameras, is able to provide *in situ* views of developing situations, and, because the videos are automatically uploaded to a server, the team's supervisors are able to accurately assess a developing situation. A company that specializes in behavior analysis of developmental disabilities in children has also been piloting the system. Their mobile childcare specialists visit area schools, and record the behavior of children for analysis by parents and medical experts. A professor of public planning and her students have used our system to document construction in post-Katrina Mississippi, with the goal of evaluating zoning regulations and revising existing ordinances.

These, and other, users have generated a corpus of over 5000

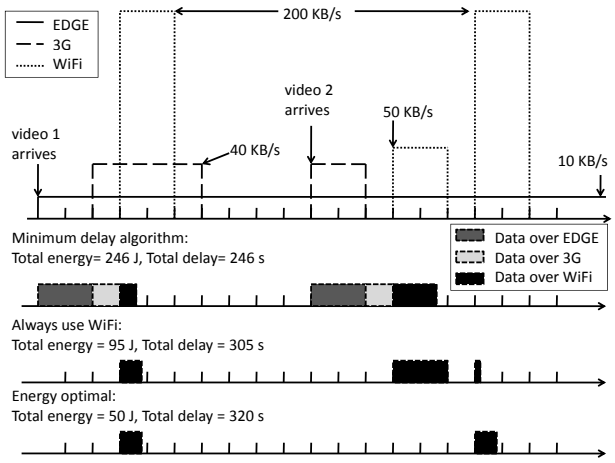

Figure 2: Example

videos. Figure 1 presents a screenshot of the system's Web interface, showing some user-generated video-clips from our users. Our users report that battery lifetime is a critical usability issue, and video uploads use a significant fraction of the energy in our system. This paper explores robust methods for reducing this cost.

Recent smartphones have multiple wireless interfaces – 3G/EDGE (Enhanced GPRS) and WiFi – that can be used for data transfer. These two radios have widely different characteristics. First, their nominal data rates differ significantly (from hundreds of Kbps for EDGE, to a few Mbps for 3G, to ten or more Mbps for WiFi). The achievable data rates for these radios depends upon the environment, can vary widely, and are sometimes far less than the nominal values. Second, their energy-efficiency also differs by more than an order of magnitude [4, 6]. While the power consumption on the two kinds of radios can be comparable, the energy usage for transmitting a fixed amount of data can differ an order of magnitude or more because the achievable data rates on these interfaces differ significantly. Finally, the availability characteristics of these two kinds of networks can vary significantly. At least as of this writing, the penetration of some form of cellular availability (EDGE or 3G) is significantly higher than WiFi, on average. A similar observation has been made in [22] where the authors report 99% and 46% experienced availability, respectively, in their traces for EDGE and WiFi. Thus, uploading or downloading large data items using WiFi can be more energy-efficient than using the cellular radio, but WiFi may not always be available.

Fortunately, many uses of video capture are naturally delay-tolerant, to differing degrees, so that it is possible to delay data transfers until a lower-energy WiFi connection becomes available. In general, our users would like captured videos to appear on the server "as quickly as possible" (so that they, or their colleagues or supervisors, can quickly review the captured video), and are willing to tolerate some delay in upload in exchange for high-quality video capture and extended phone lifetime. However, different users have different delay tolerances: surveillance experts can be, depending on the situation being monitored, less tolerant of delay than behavioral analysts or public policy experts.

This paper explores this energy-delay trade-off in delay-tolerant, but data-intensive, smartphone applications. The example in Figure 1 illustrates this trade-off. The topmost plot in the figure depicts a scenario in an urban environment where the availability and the achievable data transfer rate over three different wireless networks – EDGE, 3G, and WiFi – varies with time (each tick on the x-axis marks a 30 seconds interval). In this example, EDGE is always available but can only support 10 KB/s data rate. WiFi APs are available over 3 short time periods and provide 200 KB/s data transfer rate in two of those periods but only 50 KB/s in the other. Finally, 3G is available for the similar duration of time as WiFi but at different times, and provides a lower data rate (40 KB/s). Application data arrives at time t = 0 and t = 300s as video files with size equal to 5 MB each. Suppose that the power consumption of the 3G/EDGE and the WiFi interface on the smartphone is 1W (this roughly matches our measurements on the Nokia N95 smartphone).

In Figure 1, we depict the data transmission decisions of three different data upload decision strategies, and their performance in terms of total energy consumption on the smartphone and the delay in uploading the data. Whenever data is available for upload, the *Minimum-delay* strategy selects the link with the fastest data transfer rate whereas the *Always-use-WiFi* strategy uploads data using only WiFi APs. For comparison, we also show the *Energy-Optimal* decision strategy that would result in minimum energy consumption in this scenario. We can see from Figure 1 that the *Minimum-delay* algorithm achieves the smallest delay but consumes (almost) 2.5 and 5 times more energy than the *Always-use-WiFi* and the *Energy-optimal* strategies, respectively. Hence, in this scenario, delaying data upload to avoid using 3G/EDGE networks leads to significant energy savings at the expense of 1-1.5 minutes of additional delay. However, reducing the energy consumed in data transfer is not simply a matter of choosing WiFi over 3G/EDGE. The *Energy-optimal* strategy consumes only half as much as energy as the *Always-use-WiFi* strategy by not using the (poor quality) WiFi AP with 50 KB/s rate at the expense of only slightly higher delay.

The previous example illustrates several *decisions* involved in managing data intensive and delay-tolerant smartphone applications in an energy-efficient manner. How long should the system wait before using the energy expensive but nearly ubiquitous cellular network? If several WiFi APs are available, which AP should it choose? How can the system estimate the quality of a new WiFi AP? At their core, all these decisions involve an energy-delay tradeoff.

The problem we consider in this paper is the design of an algorithm for making this energy-delay tradeoff. More precisely, the problem can be formulated as a *link selection problem* (§2): given a set of available links (cellular, WiFi access points), determine *whether to use any of the available links to transfer data (and, if so, which), or to defer a transmission in anticipation of a lower energy link becoming available in the future, without increasing delay indefinitely*. Because it trades off delay for energy, the link selection problem can be naturally formulated using an optimization framework.

Contributions. In this paper, we present a principled approach for designing an online algorithm for this *energy-delay* tradeoff using the Lyapunov optimization framework [11, 16]. We formulate the link selection problem as an optimization formulation which *minimizes the total energy expenditure* subject to *keeping the average queue length finite*. The Lyapunov optimization framework enables us to design a control algorithm, called SALSA (**S**table and **A**daptive **L**ink **S**election **A**lgorithm), that is guaranteed to achieve *near-optimal* power consumption while keeping the average queue finite (§3). Specifically, we show that, in theory, SALSA can achieve power consumption arbitrarily close to the optimal. To our knowledge, prior work has not explored this link selection problem, and our use of the Lyapunov framework for solving this problem is also novel (§6).

Our second contribution is an exploration of two issues that arise in the practical implementation of SALSA. First, although control algorithms based on the Lyapunov framework have a single parameter V, the theory does not give any guidance on how to set that

parameter V. We design a simple but effective heuristic for a time-varying V, which allows users to tune the energy-delay tradeoff across a broad spectrum. Second, SALSA requires an estimate of the potentially achievable transmission rate on available link, in order to make its control decision. We devise a hybrid online-offline estimation mechanism that learns link rates with use, but uses an empirically derived mapping between an RSSI reading and the average achieved transfer rate during the learning phase.

Our third contribution is an extensive trace-driven evaluation of SALSA using video arrivals from users of our Urban Tomography system and link arrivals obtained from three different locations in the Los Angeles area. Our trace-based simulations show that SALSA, which makes its transmission decisions based on three factors, transmission energy, the volume of backlogged data, and the link quality is significantly better than other alternatives that do not incorporate all of these factors in their decisions. Moreover, SALSA's energy-delay tradeoff can be tuned across a wide spectrum using a single parameter α. Finally, SALSA can save between 10 and 40% of the *total energy capacity* of a smartphone battery, relative to a scheme that does not tradeoff increased delay, on many of our video traces.

Finally, we validate our trace-based simulations using extensive experiments on a SALSA implementation as part of a video transfer application on the Nokia N95 phones. Our experimental results are strikingly consistent with our trace-based results, suggesting that our conclusions are likely to hold in real-world settings.

2. PROBLEM STATEMENT, MODEL AND OBJECTIVE

To precisely describe the problem we consider in this paper, let $L[t]$ denote the set of links visible to a smartphone at time t. A link denotes a cellular radio connection (EDGE, 3G or other standard, depending upon the carrier) or a connection to a visible WiFi access point (AP). In general, current smartphone software does not provide applications with the ability to select between different visible cellular radio networks, or control which cell tower to associate with, so we do not assume this capability. However, it is possible, at least on certain smartphone operating systems, to select a WiFi AP for data transfer. $L[t]$ is time-varying: as the user moves, the availability of cellular connectivity will vary, as will the set of visible WiFi APs.

The problem we consider in this paper is the *link selection problem*: if at time t, the smartphone has some data to upload, which link in $L[t]$, *if any*, should it select for the data transfer so as to conserve energy? Our goal in the paper is to design a *link selection algorithm* that solves the link selection problem. One important feature that distinguishes our work from prior work is that the link selection algorithm *can choose to defer the transfer in anticipation of a future lower energy transmission opportunity.* Thus, our link selection algorithm trades off increased delay for reduced energy. Because different applications may have different delay tolerances, our link selection algorithm must provide the ability to control the trade-off.

The link selection problem can be naturally formulated using one of many optimization frameworks. The formulation we choose is based on the following intuition. Suppose that the application data generated on the smartphone is placed in a queue. For delay tolerant applications, it might be acceptable to hold the data in the queue and defer transmission in anticipation of a lower energy link becoming available in the future, but *not indefinitely*. In other words, as the queue becomes longer, it may reach a point where it may no longer be appropriate to trade-off additional delay for energy. One natural optimization formulation that arises from this intuition is to *minimize the total energy expenditure* subject to *keeping the average queue length finite*.

It is this formulation we adopt in the paper, and we introduce a model and associated notation to formally state the optimization objective and constraint. Our model provides a framework for the design and analysis of our online interface selection algorithm, discussed in §3. For ease of exposition of the model, we assume that time is slotted; our model and algorithm can easily be generalized to the continuous time case (indeed, our implementation, described in §5, assumes continuous time).

Let $A[t]$ represent the size of video data in bits generated during time slot t. $A[t]$ represents the arrival process, and we model it as a discrete random variable. We denote by $P[t]$ the power consumption due to data transmission during the t-th time slot. $P[t]$ is zero if the link selection algorithm chooses to defer transmissions during this time slot. If the algorithm chooses a cellular link, $P[t]$ is P_C, and if it chooses a WiFi link, $P[t]$ is P_W. More generally, the framework we discuss below is capable of incorporating transmit power control, but since smartphones do not support that capability, we have not incorporated it.

Let $\mu[t]$ denote the amount of data transferred during timeslot t. This value depends on several factors. First, $\mu[t] > 0$ only if our interface selection algorithm decides to transmit data during slot t; it is zero otherwise. If $\mu[t] > 0$, then it also depends on the following factors: (i) the quality of the link selected for data transfer, (ii) the transmit power, and (iii) the amount of data available for transmission.

As we have discussed above, video data generated for uploads are queued awaiting transmission. Let $U[t]$ denote the *queue backlog* (number of bits in queue) at the beginning of timeslot t. For a link $l \in L[t]$, let $S_l[t]$ denote the quality of the wireless link. We model $S_l[t]$ as a random variable that takes values from a finite set $\mathscr{S}$ according to probability distribution π_s for all t. We model $\mu[t]$ as the random output of a function as defined next.

$$\mu[t] \triangleq C(I[t], l, S_l[t], U[t], P[t]) \tag{1}$$

where $I[t]$ is an indicator random variable that is equal to 1 if the smartphone decides to transmit data during slot t and 0 otherwise. If $I[t] = 0$, the smartphone does not transmit during slot t (regardless of the other inputs l, $S_l[t]$, $U[t]$, etc.). l denotes the link selected for transmission and $S_l[t]$ denotes the quality of link l during slot t. Since $U[t]$ denotes the queue backlog at the beginning of slot t, we have $\mu[t] \leq U[t]$ always. $P[t]$ denotes the transmit power.

Over time, the queue backlog evolves as follows:

$$U[t+1] = U[t] - \mu[t] + A[t] \tag{2}$$

where $\mu[t]$ (defined in (1)) is the amount of data transferred during timeslot t, and $A[t]$ is the application data added to the queue during slot t.

Given this notation, we are now ready to formally state the queuing constraint we impose on our link selection algorithm, called *stability*. We define the queue $U[t]$ to be stable if:

$$\overline{U} = \limsup_{t \to \infty} \frac{1}{t} \sum_{\tau=0}^{t-1} \mathbb{E}\{U[\tau]\} < \infty \tag{3}$$

The stability constraint ensures that the average queue length is finite.

Under this constraint, we seek to design a link selection algorithm that minimizes the *time average transmit power expenditure*,

defined as:

$$\overline{P} = \limsup_{t\to\infty} \frac{1}{t} \sum_{\tau=0}^{t-1} \mathbb{E}\{P[\tau]\} < \infty \quad (4)$$

where $P[\tau] \in \{0, P_C, P_W\}$ depending on the link selected for transmission during slot τ.

3. THE LINK SELECTION ALGORITHM

In this section, we describe our link selection algorithm. This algorithm is designed using the Lyapunov optimization framework [11, 16], and has the property that it is guaranteed to be stable, and can provide near-optimal energy consumption even with varying channel conditions, under some idealized assumptions. Accordingly, we call our algorithm SALSA (**S**table and **A**daptive **L**ink **S**election **A**lgorithm). We first present SALSA, briefly describe its design using the Lyapunov framework and state its performance properties, and finally discuss its practical application to a real-world system.

3.1 SALSA

SALSA decides, every timeslot t, whether to transmit data from its queue, and which (if any) of its available links to use. To do this, it observes the amount of new application data $A[t]$ and its current queue backlog $U[t]$. For a parameter $V > 0$ (we describe later how to select this parameter), it chooses a link $\tilde{l}[t]$ for data transfer during timeslot t as follows:

$$\tilde{l}[t] = \underset{l \in L[t]\cup\emptyset}{\operatorname{argmax}} \; (U[t] \times \mathbb{E}\{\mu[t] \mid l, S_l[t], P_l[t]\} - V \times P_l[t]) \quad (5)$$

where $\tilde{l}[t] = \emptyset$ represents both the cases – when no link is available or when the smartphone chooses not to use any of the available links. $\mathbb{E}\{\mu[t] \mid l, S_l[t], P_l[t]\}$ is an *estimate* of the transfer rate that can be achieved on link l, given the current channel condition $S_l[t]$ and the transmit power $P_l[t]$. In a later section, we discuss how to estimate this value.

To understand the intuition behind this control decision, consider a specific WiFi link l such that $P_l[t] = P_W$. If V is fixed, this control decision chooses link l only when either the queue backlog $U[t]$ is high or the available rate on link l is high. Thus, the algorithm implicitly queues data for "long enough" or sends if it sees a good quality link. When P_W is higher, the bar for transmission is automatically raised. Of course, the performance of this algorithm critically depends upon the choice of V, and we discuss this later. SALSA may decide not to use any of the available links if and only if $U[t] \times \mathbb{E}\{\mu_l[t] \mid l, S_l[t], U[t], P_l[t]\} - V \times P_l[t] < 0$ for all $l \in L[t]$. Such a situation will typically arise if the data transfer rate to all the available links is small, either because the nominal rate of the link is small, or the effective transfer rate is small as a result of poor channel conditions.

3.2 Theoretical Properties of SALSA

We have formally derived SALSA's control decision (5) using the Lyapunov optimization framework [11, 16]. This framework enables the inclusion of optimization objectives – energy expenditure, fairness, throughput maximization etc. – while designing an algorithm to ensure queue stability using *Lyapunov drift* analysis. Lyapunov drift is a well-known technique for designing algorithms that ensure queue stability. The technique involves defining a non-negative, scalar function, called a *Lyapunov function*, whose value during timeslot t depends on the queue backlog $U[t]$. The Lyapunov drift is defined as the expected change in the value of the Lyapunov function from one timeslot to the next. The Lyapunov optimization framework guarantees that control algorithms that minimize the Lyapunov drift over time will stabilize the queue(s) and achieve *near-optimal* performance for the chosen optimization objective – for SALSA, power consumption.

We have discussed the derivation in Appendix A. Our derivation is similar to that of other optimization formulations that use the framework [16], but, to our knowledge, we are the first to apply this framework to the link selection problem defined in Section 2.

It is possible to derive an analytical bound on the time average power consumption achieved by SALSA compared to an *optimum* value. We state the following theorem, and prove it in Appendix B:

Theorem 1 Suppose the arrival process $A[t]$ and the channel states are i.i.d. across timeslots with distributions p_A and π_s, respectively. We assume that the data arrival rate λ is strictly within the network capacity region. For any control parameter $V > 0$, SALSA achieves a time average power consumption and queue backlog satisfying the following constraints:

$$\overline{P} = \limsup_{t\to\infty} \frac{1}{t} \sum_{\tau=0}^{t-1} \mathbb{E}\{P[\tau]\} \le P^* + \frac{B}{V} \quad (6)$$

$$\overline{U} = \limsup_{t\to\infty} \frac{1}{t} \sum_{\tau=0}^{t-1} \mathbb{E}\{U[\tau]\} \le \frac{B + VP^*}{\varepsilon} \quad (7)$$

where $\varepsilon > 0$ is a constant meaning the distance between arrival pattern and the capacity region boundary, P^* is a theoretical lower bound on the time average power consumption, and B is an upper bound on the sum of the variances of $A[t]$ and $\mu[t]$ (each of which is assumed to have finite variance).

The theorem shows that SALSA can achieve an average power consumption $\overline{P}$ arbitrarily close to P^* (with a corresponding delay trade-off) while maintaining queue stability. However, this reduction in power consumption is achieved at the expense of a larger delay because the average queue backlog $\overline{U}$ grows linearly in V. This $[O(1/V), O(V)]$ trade-off between power consumption and delay is a fundamental aspect of all control algorithms designed using the Lyapunov optimization techniques [11]. Moreover, this trade-off does not assume prior knowledge of the distributions of the stochastic processes $A[t]$ (data arrival) and $S_l[t]$ (link quality), merely that the variances of the arrival process and the transfer rates are finite.

3.3 Practical Considerations for SALSA

The SALSA algorithm discussed above is idealized in several respects. It uses fixed timeslots, assumes that the available rate on a link $\mu_l[t]$ is known a priori, and does not specify how to select the parameter V. When implementing it in practice, it is easy to change the fixed timeslot assumption and invoke the control decision whenever data is inserted into the queue or a new link becomes visible. In this section, we discuss how to deal with the other idealizations.

Choosing a "good" V. In general, the Lyapunov optimization framework is elegant because its control algorithms depend on a single parameter V. However, the framework itself does not give any guidance on parameter selection. Intuitively, V can be thought of as a threshold on the queue backlog beyond which the control algorithm decides to transmit ((5)), so V controls the energy-delay tradeoff. Most existing work in this area chooses not to address the parameter selection issue explicitly, and simply explores the sensitivity of their results to the choice of parameters.

However, since we are interested in implementing a system based on this framework, we need to explicitly address parameter selection. One obvious choice is to estimate the parameter V online: as the system runs, we can adapt V (e.g., using a binary search) to find a setting where the energy delay trade-off is optimal. This can take

a long time to converge, since at each step we would have to have run the system long enough for the average queue length to have converged.

We design a technique to determine the value of V automatically with two goals in mind. Our first goal is to pick a V value that achieves *good* power consumption vs. delay trade-off. The second one is to enable some degree of explicit control over the energy-delay tradeoff — recall from §1 that different video capture applications have different delay tolerances.

To identify a good V value, observe that the upper bound on the time average power consumption from (6) is proportional to $1/V$. Based on this, we make a simplifying assumption that the actual time averaged power consumption $\overline{P} \approx P^* + B/V$ ((6)). Since P^* is a constant, $\overline{P}$ is a hyperbolic function that exhibits diminishing returns, beyond a point, in energy reduction with increasing V.

Thus, *a good operating point would be to pick a V value where a unit increase in V yields a very small reduction in $\overline{P}$.* At this point, the energy gains may not be worth the delay increase resulting from increasing V (since delay is proportional to V). More formally, we can choose an $\alpha > 0$ that satisfies the following equation (α is the slope of $\overline{P}$ curve):

$$\frac{d(P^* + B/V)}{dV} = \frac{-B}{V^2} = -\alpha$$
$$\implies V = \sqrt{\frac{B}{\alpha}} \quad (8)$$

In setting V according to (8), we need to determine the value of the constant B, which involves estimating the variance of the arrival process $A[t]$, and the transmission process $\mu[t]$. SALSA computes B based on all the $A[t]$ and $\mu[t]$ values observed over some large time window. It initializes $V = 0$ and then updates its value according to (8) whenever the estimate for B is updated.

To achieve our second goal, we adapt V to the *instantaneous* delay in data transfer using an application-specified parameter. Such a mechanism enables a smartphone application to express its delay-tolerance. Rather than use a fixed V during each timeslot, SALSA modifies (8) as follows:

$$V[t] = \sqrt{\frac{B[t]}{\alpha \times (D[t]+1)^{\alpha}}} \quad (9)$$

where $D[t]$ denotes the *instantaneous* delay in data transfer (i.e., the time that the bit at the head of the queue has been resident in the queue) measured at the beginning of timeslot t. Note that the upper bound B now becomes $B[t]$. However, the intuition is simple: as data stays longer in the queue, $V(t)$ decays (at a rate determined by α, which can be controlled by the application) until it becomes low enough to trigger a transmission by (5). Hence, SALSA reacts to an increase in instantaneous delay by trying to transmit data whenever an access point is available. While this reduces the delay in data transfer, it can result in higher power consumption as SALSA may select an access point for data transfer that is not energy-efficient instead of delaying transmissions till an access point with high data transfer rate appears. Thus, applications that can tolerate delay and would prefer to maximize energy savings can set α close to zero, while less delay-tolerant applications can set α to be larger at the expense of energy usage (we explore the behavior of the algorithm to different α values in §4).

Note that the parameter α appears twice in the denominator in (9) – as an multiplicative term and also as the exponent of $(D[t]+1)$. We need α as a multiplicative term in order to get a "good" V value when $D[t] = 0$ (in which case (9) reduces to (8)). Instead of using a different parameter β as the exponent of $(D[t]+1)$ in (9), we chose to use α in order to have only one free parameter in SALSA. As we show in §4, a single parameter is sufficient to explore a range of delay-tolerances.

The bounds on average power consumption and average queue size in (6)-(7) hold when $V[t] = V$ for all t. For the case of time varying V values, it is difficult to derive similar bounds. However, we can easily see that, compared to the case of a fixed V, SALSA with time-varying V values achieves smaller average queue backlog (hence, smaller average delay) at the expense of higher average power consumption. That is because the instantaneous-delay based term triggers transmissions earlier in SALSA with time-varying V than in SALSA with fixed V, at the possible cost of increased energy incurred by transmitting on a less-than-optimal link.

Rate estimation. In practice, the transfer rate on link l during slot t, $\mu_l[t]$, may not be known. SALSA uses a combination of *offline* and *online* estimation.

In *online* rate estimation, as the smartphone uses each link l, SALSA computes $\mu_l[t]$ as the average rate achieved over the last, say, 10 uses of link l for data transfer. This windowed average, because it is specific to a link, can be accurate but would require several uses of a link before a reliable estimate could be found.

Until a reliable estimate is available, SALSA uses results from an *offline* rate estimation technique that samples several access points to obtain a distribution of achievable rates. There are many ways of doing this, but the simplest (and the one we use), estimates the distribution of achievable transfer rates as a function of the Received Signal Strength Indicator (RSSI) for a given environment. SALSA simply derives a rate estimate from this distribution for each link, based on its RSSI. Admittedly, this is a very coarse characterization, since data rates are only partially dependent upon RSSI. However, as we show in this paper, even this rough estimate results in excellent SALSA performance.

3.4 Extensions

SALSA is also flexible enough to accommodate extensions that may be desirable for smartphone applications. We now discuss two such extensions, but have left their evaluation to future work. In addition to these, SALSA can be extended to accommodate prioritized data transmissions, or bounds on average power consumption. We have omitted a discussion of these for lack of space.

SALSA for download. SALSA can also be extended for link selection for data downloads. Many applications can live with a delay-tolerant download capability. Such applications download, in the background, large volumes of data (e.g., videos, images, maps, other databases) from one or more Internet-connected servers in order to provide context for some computation performed on the phone. A good example is Skyhook's [24] WPS hybrid positioning service, which prefetches relevant portions of precomputed hotspot location database.

To get SALSA to work for such applications, we need to change the definition of $A[t]$ and $U[t]$. Specifically, we define $A[t]$ as the size of the request by an application during timeslot t, and $U[t]$ as the backlog of content that has not been downloaded yet. In applying SALSA to the download scenario, we assume that it is possible to know the size of the content requested by an application prior to downloading the content. This is certainly feasible for static content hosted by a server, and for dynamically generated content for which the server is able to estimate size.

SALSA for peer-assisted uploads. In a peer-assisted upload, data is opportunistically transferred to a peer smartphone with the expectation of reducing the latency of upload. In general, peer-assistance will require the right kinds of incentives for peers to participate.

However, for certain cooperative participatory sensing campaigns, where a group of people with a common objective collectively set out to gather information in an area, peer-assisted uploads are a viable option to increasing the effective availability of network connectivity.

For the peer-assisted upload case, we can model the connection to the peer as a link. The important change is that the achievable rate $\mu_l[t]$ of this link takes into account an estimate of the upload delay (the time when the peer expects to meet a usable link). When a smartphone meets a peer, it queries the peer to get an estimate of $\mu_l[t]$ on the link l between them. The peer computes this quantity by estimating the time that it is likely to meet the next AP (say t_m), and the achievable rate r to that AP. It advertises $\mu_l[t]$ as $\frac{r}{t_m}$, which is an estimate of the effective data rate that would be observed by a transfer handed-off to this peer. Recent work [17] suggests that it might be possible to accurately forecast r, and t_m can be estimated using GPS and trajectory prediction.

4. EVALUATION

In this section, we present our evaluation of SALSA using trace-driven simulations. We motivate and describe our methodology, then discuss our results. We have also implemented SALSA on the Nokia N95 smartphone as part of the Urban Tomography system (§1): in the next section, we use this implementation to validate our simulation results.

4.1 Methodology

Overview. In our evaluation of SALSA, we are interested in two questions: How does SALSA perform over a wide range of scenarios? How does it compare to other plausible link selection algorithms?

The performance of SALSA (or any other algorithm) depends upon two characteristics: the arrival process $A[t]$, and the time variation in link quality and availability as defined by $\mu_l[t]$. To understand the performance of SALSA over a wide range of arrival processes and link availability and quality characteristics, we use trace-driven simulation, with arrival traces derived from users of our urban tomography system in real-world settings, and link availability traces generated empirically by carrying a smartphone on a walk across different environments. We describe the methodology in detail below.

We also compare SALSA against two baseline algorithms, one which attempts to minimize delay and the other which always uses WiFi to conserve energy, as well as two other threshold-based algorithms.

We begin a detailed discussion of the simulator and traces that we use. Then, we discuss the alternative strategies we use for comparison. We conclude this section with a discussion of our metrics.

Simulator Details. We wrote a custom simulator to explore the performance of SALSA and compare with other algorithms. Our simulator allows us to explore the impact of different application data arrival patterns and link availability characteristics on an algorithm's performance. It also enables us to characterize the effect of our heuristic for determining the V value and our rate estimation scheme on SALSA's performance.

Our simulator takes three different inputs – (1) the power consumption of the different radio interfaces on the smartphone, (2) the application data arrival patterns, and (3) the link availability. All our simulation results are for a timeslotted system with 20-second time slots.

Based on our measurements on the Nokia N95 smartphones, we set the transmit power consumption of the 3G/EDGE interface and

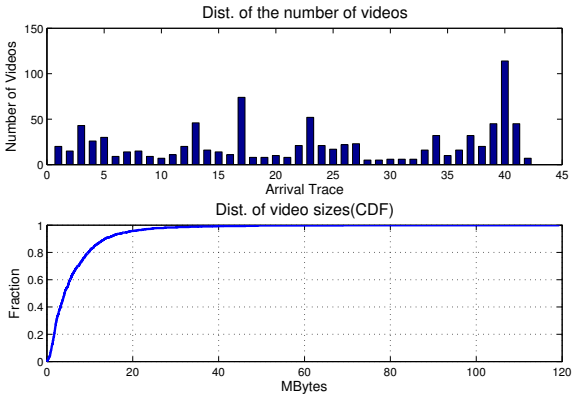

Figure 3: Arrival Patterns

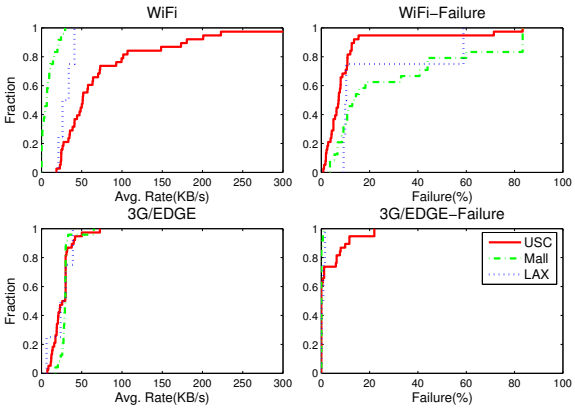

Figure 4: (CDF) Link availability with failure probability

the WiFi interface to 1.15 W and 1.1 W, respectively. We assume that interfaces are briefly turned on at the beginning of each timeslot to check for availability; only the radio selected for the transfer (if any) is kept on for the duration of the transfer. We ignore the energy cost of checking for availability (which may include the cost of scanning for access points): relative to the large volumes of data we transfer, this cost can be made negligible by tuning the frequency of scanning, as we show later in this section. Furthermore, all algorithms are more or less equally affected by this simplification, and since we compare algorithms, we do not expect the relative performance of these algorithms to change significantly if these costs were taken into account.

Our simulator uses two kinds of traces: an *arrival trace* and a *link trace*.

An *arrival trace* captures a data arrival pattern, and consists of a timestamped sequence of video arrivals. We use a total of 42 arrival patterns (consisting of a total of 935 videos), derived from actual use of the Urban Tomography system. In that system, users can create "events" that mark a collection of related videos (usually representing the documentation of a real-world event, such as a commencement ceremony, or a business trip). Each arrival trace is generated from one event. Arrival traces have widely varying characteristics; for example, Figure 3 shows the distribution in the number and total size of videos across different traces.

Each *link trace* is a timestamped sequences of available APs (3G/EDGE and WiFi) together with data transfer rates. We collected link traces while we were experimenting (at different times over several months) with our system at several different locations – the USC campus, a large shopping mall near Los Angeles (Glendale Galleria), and the Los Angeles International Airport (LAX).

We collected 38 traces on the USC campus, 24 traces at the Glendale Galleria, and 4 traces at LAX. At these locations, WiFi (specifically 802.11b/g) is available to different extents. On the USC campus, WiFi is deployed across most of the campus, and is freely available to registered clients. The Glendale Galleria has a few open WiFi hotspots. At the LAX airport, we purchased four T-Mobile hotspot accounts and scripted the login procedure so that association with those hotspots does not require manual intervention.

Our link traces are collected by walking in the corresponding environment for an hour or more with a smartphone which periodically scans for APs, records the SSID (or cell tower ID) and the RSSI value of the available APs, and estimates the data transfer rate for these APs by uploading test data. The left column in Figure 4 is a CDF of the average transfer rate per 20-second window (the timescale at which our link selection algorithm works) observed at different locations. For each trace, we divide entire time duration into 20-second windows, associate all available APs with the corresponding windows and then compute the time average data transfer rate per trace. Thus, in close to 20% of the 38 traces collected on the USC campus, we encountered WiFi APs with average rate better than 100 KB/s. From these figures, it is clear that the WiFi environment at our three locations varied widely: the USC campus has more dense and perhaps faster WiFis compared to LAX and the Glendale Galleria. On the other hand, the performance of the 3G/EDGE network is roughly the same at all the three locations.

The average transfer rate alone is a little bit misleading, because often our TCP-based data transfers fail. During our trace collection, we also record instances of upload failure for each AP. Specifically, for each trace, we compute the number of failed attempts and divide it by the total number of data transfer attempts to compute the failure rate associated with each AP. Figure 4 (right column) shows the CDF of failure rates for the traces collect at each of three locations. Note that around 20% of the traces collected at the Glendale Galleria and the LAX have a failure rate of more than 60% for WiFi APs associated with them! Interestingly, the 3G/EDGE network traces collected at USC contained higher instances of failures compared to the traces from the Glendale Galleria and the LAX. Non-trivial failure rates at these environments implies that it is important to incorporate such failures (and the energy cost associated with them) in our simulations, and we do.

A single *simulation run* uses one arrival trace and one link trace as input. Each simulation run lasts until either all the data is uploaded from the smartphone or 50,000 slots (equivalent to about 12 days) have elapsed. Overall, we have 2,772 simulation runs for each algorithm we evaluate (see below).

The failure rate associated with each trace is used to model data transfer failures in our simulations as follows. For a simulation run which uses a link trace with an associated failure rate p, we assume that data transfer failures are i.i.d. Bernoulli random variables with parameter p. Our failure model provides a simple abstraction to capture the variability in *instantaneous* data transfer $\mu[t]$ that our analytical model allows for (refer to (1)).

Finally, depending on the number and quality of links available in the trace as well as arrival patterns in the arrival trace, the time needed to upload all the data can be quite large. However, our longest link trace is close to 3 hours long. In order to complete a simulation run that is longer than its corresponding link availability trace, we continuously repeat the trace. In this way, an arrival pattern sees the variability associated with a particular environment, but repetitively: this methodology allows us to explore longer arrival patterns, while still subjecting uploads to link availabilities derived from real environments.

Comparison. We compare SALSA's performance against four different link selection algorithms – MINIMUM-DELAY, WIFI-ONLY, STATIC-DELAY, and KNOW-WIFI.

The MINIMUM-DELAY algorithm always transfers data when an AP is available. It never considers the energy cost of using an AP, and is designed to minimize the amount of time application data is buffered on the smartphone awaiting transmission.

The WIFI-ONLY algorithm uses only WiFi APs. This algorithm is motivated by the observation that data transfer using WiFi APs is much more energy-efficient compared to using the 3G/EDGE network. Hence, it aims to minimize the energy consumption, and is oblivious to the delay in data transmission.

The STATIC-DELAY algorithm attempts to achieve an energy vs. delay trade-off using the following heuristic: it waits for a WiFi AP to become available for up to a (configurable) period of T timeslots from the creation time of the corresponding file, and if it encounters a WiFi AP within this period, it uses it. In the event that it has not seen any WiFi AP in the past T timeslots, it uses the first link that becomes available (whether 3G/EDGE or WiFi). Thus, this algorithm behaves like WIFI-ONLY for up to T timeslots, and then starts behaving like MINIMUM-DELAY. The parameter T controls the energy vs. delay trade-off. Ideally, it should depend on an application's delay-tolerance, and the availability of WiFi APs. Without detailed information about WiFi availability, a big challenge in using this algorithm is to determine the parameter T.

Finally, the KNOW-WIFI algorithm assumes information about the availability of WiFi APs in the future. It is therefore an idealized algorithm although it may be possible to estimate this availability as described in [17], or determine it based on user input (for example, when a user knows the when she is going to have access to a good WiFi AP, such as at home, work, or a coffee shop) in advance. It checks for the availability of a "good" WiFi AP within the next T timeslots. We define a good WiFi AP as one that has a data transfer rate at least twice the maximum achievable 3G/EDGE rate, obtained from the corresponding link trace. If such an AP exists (i.e., the user will encounter it within the next T timeslots), the KNOW-WIFI algorithm waits until it can use that AP, and then transfers as much data as possible using it. It then resets the maximum wait period for a good AP back to T timeslots. In situations where the KNOW-WIFI algorithm knows that no good AP will appear within the next T timeslots, it behaves like the MINIMUM-DELAY algorithm, and starts using any available link. Apart from the fact that this algorithm requires knowledge of WiFi APs available in future, another practical challenge is determining the right value for T.

Performance metrics. Finally, we analyze the performance of each algorithm using a novel approach that attempts to characterize the macroscopic performance of each algorithm across all our simulation runs. At the end of each simulation run, we first derive two metrics for each link selection algorithm: (1) the average energy consumed per byte ($\overline{E}$), and (2) the average delay per byte $\overline{D}$. Consider a VCAPS based application that generates N videos with size $S_1, \ldots, S_N$ bytes. Let E_i and D_i denote the total energy consumed and the delay, respectively, in transmitting the video of size S_i. We define the average energy consumed per byte, $\overline{E}$, and the (weighted) average delay per byte, $\overline{D}$, as follows.

$$\overline{E} = \frac{\sum_{i=1}^{N} E_i}{\sum_{i=1}^{N} S_i}, \quad \overline{D} = \frac{\sum_{i=1}^{N} (D_i \times S_i)}{\sum_{i=1}^{N} S_i} \tag{10}$$

Each simulation run results in a point on the E-D plane. The convex hull of all the points for a given algorithm in the E-D plane represents its *envelope* of performance. We present examples of en-

velopes and discuss desirable properties in §4.2. In that section, we also compare the envelopes of performance of different algorithms.

We also use another metric, called *dispersion*, to characterize how far off each algorithm is, on average, from an idealized optimal. For each pair of arrival trace and link trace, we can compute the minimum achievable energy per byte ($\overline{E_m}$) and the minimum achievable delay per byte ($\overline{D_m}$), if each were separately optimized (instead of jointly, as SALSA does). Specifically, $\overline{E_m}$ is the energy per byte used if all data were transmitted using the highest rate link in a trace. Similarly, $\overline{D_m}$ is the delay per byte incurred using MINIMUM-DELAY, assuming no transmission failures. For $(\overline{E_m}, \overline{D_m})$ pair, we can also obtain for each algorithm, the Euclidean distance on the normalized $E - D$ plane between the achieved $(\overline{E}, \overline{D})$ and $(\overline{E_m}, \overline{D_m})$. In general, the latter point may not be achievable by any algorithm that trades-off delay for reduced energy, but it represents a lower bound. For a given algorithm, the average distance of each simulation run from the corresponding "optimal", across all simulation runs, is defined to be the dispersion of the algorithm.

4.2 Performance Results

Performance against Baseline Algorithms. We first compare SALSA against the two baseline algorithms, MINIMUM-DELAY and WIFI-ONLY. Figure 5 plots the performance of each of these algorithms on the *E-D* plane. For SALSA, we use an α of 0.2: we later explore the performance of SALSA across a range of α values.

As discussed in §4.1, each point on the *E-D* plane corresponds to one simulation run. One way of characterizing the overall performance of the algorithm is to understand the shape of its *envelope*: the convex hull of all the points on the *E-D* plane. For the class of algorithms that make an energy delay trade-off, what characterizes a good envelope? Intuitively, a good algorithm should be capable of achieving a "good" balance between energy and delay: neither the delay per byte, nor the energy per byte should be too large. In other words, the points on the *E-D* plane should be clustered around the origin, and the envelope should be compact. We use this intuition to compare different algorithms throughout this section.

In Figure 5, MINIMUM-DELAY exhibits low average delay but, its energy performance is spread out over a relatively wide range. This is expected: MINIMUM-DELAY does not attempt to explicitly trade-off delay for energy. Moreover, MINIMUM-DELAY does not take channel capacity into account, so it can incur more transmission failures and, as a result, more energy. At the other end of the spectrum, WIFI-ONLY exhibits an average delay spanning the whole range of *D* values. WIFI-ONLY's performance is highly dependent upon the availability of high-quality WiFi APs. In some of our traces (especially the Glendale one), good WiFi APs are few and far between, so WIFI-ONLY can incur significantly high delay. In some traces where there is no usable WiFi, WIFI-ONLY has infinite average delay values: these are omitted in the figure. WIFI-ONLY's energy performance is also poor: in some of our link traces, the achievable rate with WiFi varies significantly (Figure 4), and is sometimes less than the rate of the 3G/EDGE network. By not discriminating based on channel conditions, WIFI-ONLY can also exhibit high energy usage.

By contrast, SALSA, which is explicitly designed for finite average delay (and, more than that, to keep instantaneous delay bounded) and which takes channel quality into account in its transmission decisions, achieves a much better performance. Compared to WIFI-ONLY, its envelope is much more compact. Its envelope is also more compactly clustered around the origin than that of MINIMUM-DELAY. As we have discussed before, some applications may want to explicitly control the delay-energy trade-off behavior: in particular, there maybe applications that would like to emulate WIFI-ONLY or MINIMUM-DELAY. We discuss below how different parameter settings for SALSA can be used to mimic these algorithms.

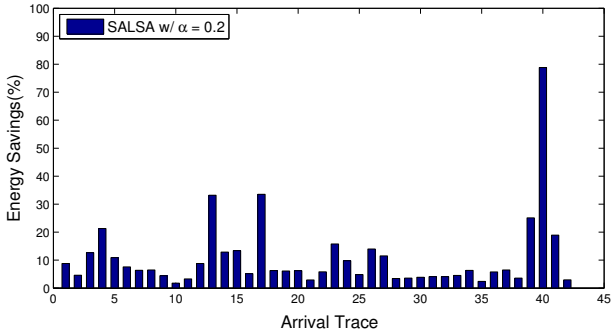

Figure 6: SALSA Energy Savings.

The rightmost sub-figure of Figure 5 depicts the dispersion of these three algorithms. Recall that the dispersion measures the average distance from an empirically-determined optimal. Before we describe these results, we briefly describe how dispersion is calculated. For calculating the distance on the *E-D* plane, we can use the absolute values of delay per byte and energy per byte, but the resulting distances then become very sensitive to the choice of units for delay and energy. To avoid this, we normalize the delay and energy by assigning a unit of 1 to the 95th-percentile values from all the simulations for each axis.

The (normalized) dispersion of MINIMUM-DELAY is about twice that of SALSA, and that of WIFI-ONLY (ignoring the runs where WIFI-ONLY had infinite delay) is about 3.4 times that of SALSA! WIFI-ONLY's performance is, of course, significantly affected by several large outliers from link traces which had very little WiFi availability. That SALSA is better than these two algorithms is not surprising, since they are relatively simple: later, we show that SALSA outperforms other, more sophisticated, algorithms as well. What is more interesting is that SALSA's absolute distance from the empirical optimal is low (0.343), leaving little room for improvement.

We consider the following question: does being a delay-tolerant actually save significant energy? From Figure 5, we can see, by considering the energy-per-byte values, that SALSA uses roughly half the energy per byte of MINIMUM-DELAY. Thus, relative to the most obvious implementation, SALSA is on average twice as energy-efficient. But, does this improvement in energy-efficiency matter in the real world, i.e., are we solving a real problem? To understand this, we measured the total energy used in Joules, for each of our arrival traces, both by SALSA and MINIMUM-DELAY. Then, we computed the ratio of the difference in energy usage to the overall battery capacity of the Nokia N95. This gives us, for each arrival trace, the *fraction of battery capacity that would have become available for use by other applications if SALSA were used instead of* MINIMUM-DELAY. Figure 6 plots this fraction for each arrival trace. For most traces, this number is in the 5-15% range, but there exist some events where users could have extended their battery life by 20-40% by using SALSA instead of MINIMUM-DELAY for uploading their videos. In one extreme case, MINIMUM-DELAY would have required more than one complete charge of the battery to upload the corresponding videos, but SALSA could have completed it without recharging.

This brings up another question: how much does SALSA pay in delay for these energy savings? Figure 7 plots the average *additional* delay incurred by a video when using SALSA over MINIMUM-DELAY, for each of our arrival traces, averaged over all link traces.

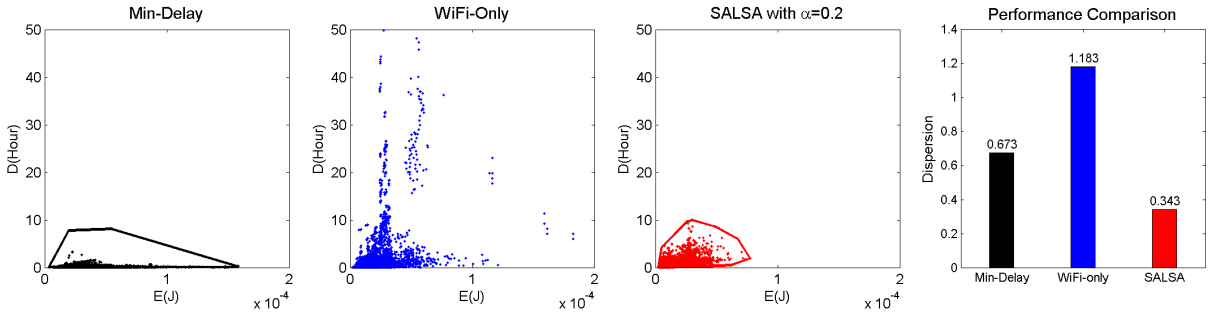

Figure 5: MINIMUM-DELAY vs WIFI-ONLY vs SALSA

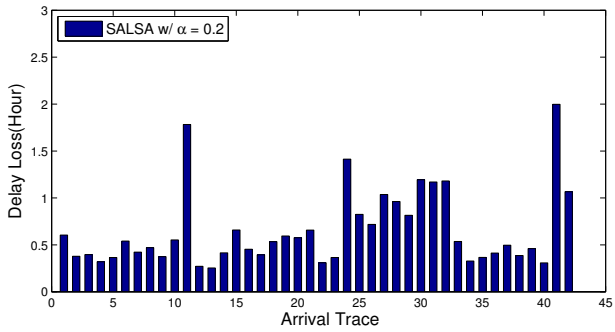

Figure 7: Additional Delay Incurred by SALSA.

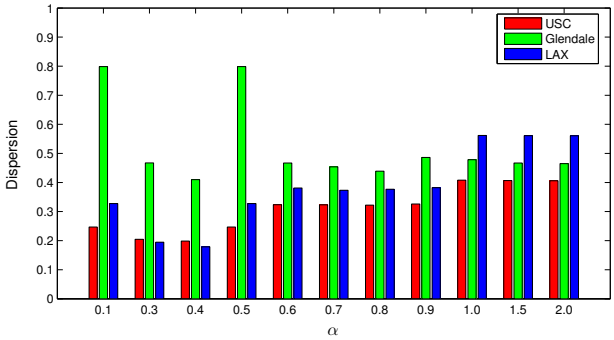

Figure 9: Performance across different environments.

For most videos, this additional delay is on the order of half an hour, and the worst-case average delay is about 1.5 hours. This tradeoff is quite encouraging: assuming that, for example, a user's smartphone lasts 12 hours, she can get, in most cases, between 30 mins to 90 minutes (5-15% of battery capacity) extra usage of her smartphone, while giving up an average delay of about 30 minutes in video upload.

SALSA Performance for different α. In §3.3, we described the design of a time-varying V parameter that would allow users to explicitly control energy-delay trade-offs. In this section, we explore the efficacy of our design by varying α from 0.1 to 2.0 with steps of 0.1. Beyond $\alpha = 2.0$, SALSA's behavior converges to that of MINIMUM-DELAY; in this range, V values approach zero and with small values of V, SALSA never defers transmissions.

Figure 8 depicts the results for a subset of the α values. By comparing with Figure 5, it is clear that SALSA can span a fairly broad range in the spectrum of energy delay trade-offs. For very small α, SALSA's envelope is qualitatively similar to that of WIFI-ONLY: for small α tending toward zero, V is high, setting a high bar (e.g., a very good WiFi AP) for SALSA's transmission decision. As α increases, the envelope becomes more compact and also flattens out until, at $alpha = 2.0$, it starts to resemble MINIMUM-DELAY. Thus, by varying α we are able to mimic both ends of the energy-delay tradeoff spectrum, and points in between. However, it is harder to intuitively understand the direct relationship between α and an application's delay tolerance. In future work, we hope to develop rules of thumb, based on deployment experience in different environments, for suggesting α values for our users.

The rightmost sub-figure of Figure 8 reveals a more interesting behavior. It plots the variation in dispersion as a function of α. From this, it appears that a value of $\alpha \approx 0.4$ is a *sweet spot* in the parameter space, having low dispersion. Recall that α serves two functions: one is to choose a good point of diminishing returns in the energy-delay tradeoff, and the other is to control the delay tolerance. The sweet spot value strikes the best balance for these objectives.

Finally, Figure 9 depicts the difference in SALSA's performance across different locations. While the sweet spot behavior is also consistent across locations, the absolute values of the dispersion are much higher in environments with sparse WiFi availability, such as the Glendale mall.

Comparison with threshold-based algorithms. We now compare SALSA's performance against that of STATIC-DELAY and KNOW-WIFI. This comparison presents a methodological difficulty: both STATIC-DELAY and KNOW-WIFI have a time threshold parameter T, but its relationship to α is not clear. Thus, it would be misleading to compare a version of SALSA with a specific α and STATIC-DELAY or KNOW-WIFI with a specific value of T.

So, we adopted a slightly different methodology. Empirically, we found, for each algorithm, the most agressive (in terms of transmission) and least aggressive parameters. We determined these ends of the parameter space by manually trying different large and small values. For SALSA, for example, $\alpha = 2$ is the most aggressive value; as we have discussed above, the system is not sensitive to choices of α beyond this. Its least aggressive parameter is a value close to zero; we chose 0.1. For STATIC-DELAY and KNOW-WIFI, the most agressive parameter was selected as 10 *mins* and the least aggressive as 16 *hours*. Between these parameter values, we selected 10 other parameters, and then executed each simulation run for these 12 parameters, for all three algorithms. For each algorithm, we plotted *all* simulation runs (across all parameter values) on the *E-D* plane. The resulting envelope captures the performance of the algorithm across a large range of the parameter setting, and, is likely a good indicator of the macroscopic performance of each algorithm.

Figure 10 plots these envelopes. SALSA's envelope is much more compact than that of the other two algorithms. It uses less energy and incurs less delay in general, and it has smaller and fewer

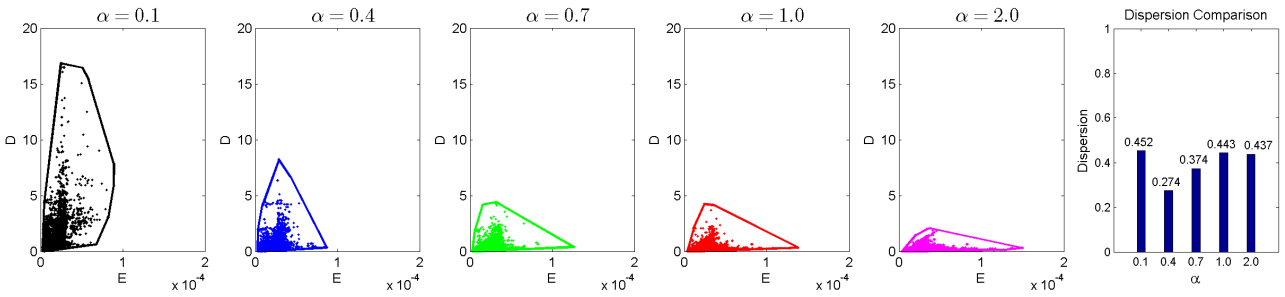

Figure 8: SALSA envelopes for different α

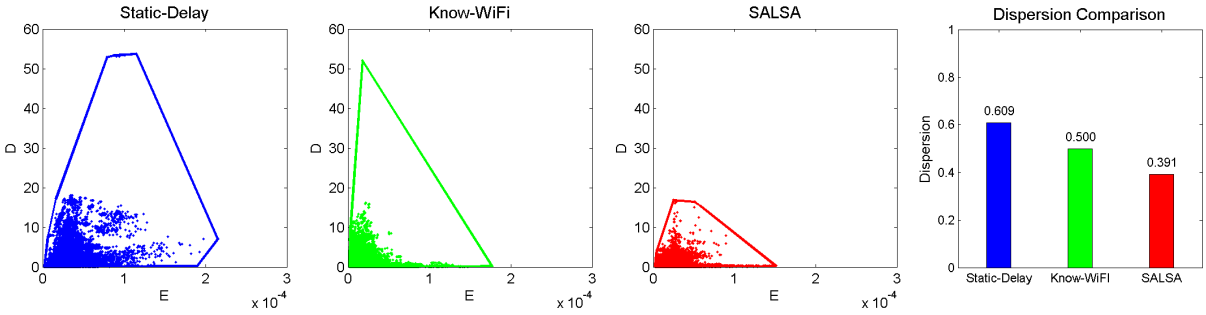

Figure 10: STATIC-DELAY VS KNOW-WIFI VS SALSA

of outliers compared to other two algorithms. STATIC-DELAY performs the worst, because it relies on a simple assumption that WiFi is more energy-efficient than 3G/EDGE. However, that is not always the case in our traces, and STATIC-DELAY sometimes pays a delay penalty waiting for WiFi only to find that the quality of the WiFi link is not significantly better than the 3G/EDGE network.

Interestingly, SALSA is also able to outperform an algorithm that has knowledge of upcoming good WiFi links. Clearly, KNOW-WIFI is careful in that it waits only for good WiFi connections, unlike STATIC-DELAY which indiscriminately uses the next WiFi link to come along. So why does KNOW-WIFI not perform as well as SALSA? The answer is that KNOW-WIFI *does not take the queue backlog into account*. Simply knowing that a good WiFi link will come along is not helpful, without knowing if that WiFi will be available long enough to be used to upload the queue backlog!

The dispersion comparison, in Figure 10, also bears this out. STATIC-DELAY has a dispersion 55% higher than SALSA, while KNOW-WIFI's dispersion is about 27% higher.

Summary of Results. In summary, our comparison with a progression of heuristics suggests the following. The comparison with MINIMUM-DELAY suggests that significant energy benefits can be obtained by judiciously delaying transmissions. However, indiscriminately delaying a transmission until a WiFi link becomes available (as WIFI-ONLY does), doesn't work well for two reasons: poor WiFi availability, and variable WiFi quality. Both these reasons are important: STATIC-DELAY is careful about waiting for a bounded amount of time for a WiFi link, but thereafter uses the first WiFi link that comes along. In our traces, there is significant WiFi variability, as a result of which STATIC-DELAY does not perform well. Finally, taking WiFi quality into account by looking ahead into the future, as KNOW-WIFI does, is also not sufficient for good performance: it fails to account for the duration of that link's availability, so in many cases the entire backlog cannot be uploaded, resulting in high delay.

SALSA, by explicitly or implicitly considering channel quality, backlog, as well as the effective transmission rate of the radio, performs the best. Of course, it may be possible to design other heuristics that take all of these factors into account, but, as we have shown, SALSA's absolute dispersion values are quite low, and leave little room for improvement.

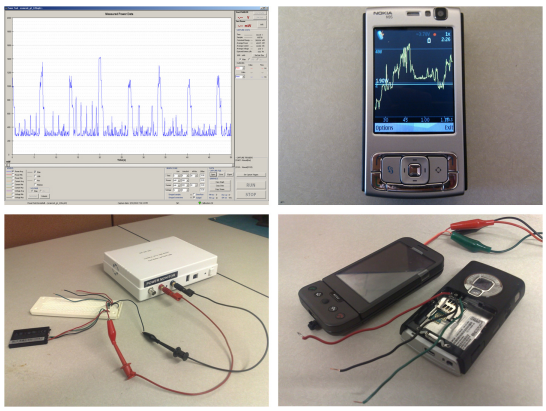

Figure 11: Energy Measurement Environment

Sensitivity to the Scanning Interval. In our trace-driven simulations, we have assumed that a WiFi scan is conducted at the beginning of every slot (i.e., every 20 seconds). Of course, WiFi scanning every 20 seconds can incur significant energy. To understand whether a larger scanning interval can be used, we explore the sensitivity of SALSA's performance to the choice of WiFi scanning frequency.

To quantify the cost of scanning, we first measured the cost of a single WiFi scan on two different platforms: the Nokia N-95 and the Android G1. For measuring the energy consumption of a WiFi scan operation, we used both a dedicated power monitor hardware [15] and a software tool (the Nokia energy profiler v1.2 [18]). Our software and hardware setup is shown in Figure 11.

Table 1 shows the result of our measurements. The N95 con-

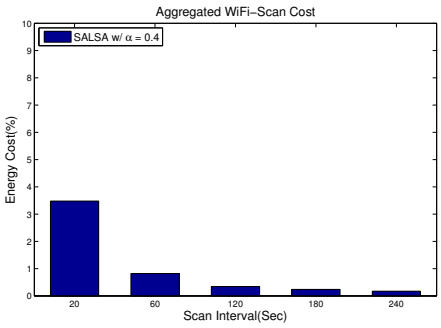

Figure 12: Scanning Energy Cost

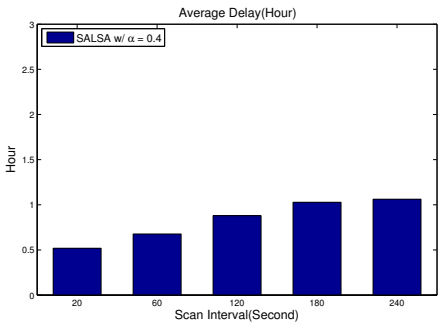

Figure 13: Average Delay

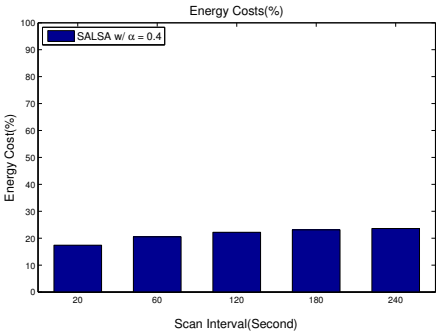

Figure 14: Average Energy Consumption

sumes 1.18J per scan, which lasts 2.03s. The G1 consumes less, about 0.63J, and it lasts 1.11s. Thus, depending on how frequently scans are invoked, scanning costs can be quite significant and require careful attention to system design.

To understand SALSA's sensitivity to the scanning frequency, we ran our trace-driven simulations for four additional scanning intervals: 60s, 120, 180s and 240s. For each scanning interval, we then counted the number of scans performed by the algorithm, and then computed the fraction of total battery capacity that can be attributed to scanning. To understand this graph, it is important to realize that SALSA (or any of our other algorithms) will only scan when the queue is nonempty. In general, one might expect SALSA to scan slightly more often than MINIMUM-DELAY, because it defers transmissions and builds up the queue.

Figure 12 plots the average scanning cost for each event in our trace, as a fraction of the total battery capacity. Clearly, at a 20s scan interval, SALSA's scanning cost is a significant 3% of the total battery capacity. For a 60s scan interval and beyond, it is much more reasonable and decreases quickly. When compared to the average energy consumption per event (Figure 14), we see that SALSA's scanning costs become a relatively small fraction for any scanning interval greater than or equal to 60s.

Interestingly, it is not possible to increase the scanning interval without a penalty. As Figure 13 shows, the average delay per interval increases fairly dramatically with scan interval, going from about 30 mins for a 20s interval to over an hour for a 240s interval. The reason for this is that a larger scan interval increases the burstiness of the departure process (relative to a smaller scan interval), and this increases the SALSA threshold V, forcing the algorithm to wait longer for better quality APs. Thus, the sweet spot for the scanning interval appears to be 60 seconds, where the cost of scanning is a small fraction of the total energy and the delay is comparable to a 20s scan interval.

5. EXPERIMENTAL RESULTS

In this section, we describe the implementation of SALSA within a video transfer application developed in Symbian C++ for the Nokia N95 smartphones. We then discuss the results of an experiment designed to verify the performance of SALSA under real-world conditions, and to validate our simulation results.

Platform	WiFi-Scan(J)	Duration(second)
Nokia N95	1.18J	2.03s
Android G1	0.63J	1.11s

Table 1: Scan Cost Measurement

Implementation Description. We have implemented the SALSA algorithm in our Urban Tomography system. The component of this system that runs on the smart phone and transmits videos to the backend server is called the **V**ideo **CAP**ture **S**ystem [25], or VCAPS. Our implementation runs on the Nokia N95 smartphone, which has a 802.11b/g WiFi interface as well as 3G/EDGE, a 2GB micro-SD card, and supports 640x480-resolution video recording capability at full frame rate.

In our implementation, VCAPS periodically scans the environment and determines the set of usable APs. These scans occur every 20 seconds (which constitutes a *timeslot*), a time period empirically detemined to work well, yet expend relatively low energy scanning for APs. At this time, it also updates all relevant statistics that are used in calculating $V[t]$.

The videos captured by a user are placed in a designated video directory, which represents the backlog queue in our system. Whenever this queue is non-empty, VCAPS attempts to transfer data to an Internet-connected server using HTTP [1]. For each transfer attempt, VCAPS invokes SALSA's decision algorithm (§3) to determine which link to use, among the ones available. The video upload process runs in the background and does not require any user intervention.

In practice, transfer attempts may fail, for several reasons, and SALSA has built-in robustness mechanisms to deal with such failures. For example, a transfer attempt may fail because current achievable rate on the current chosen link is low (either because the estimate was wrong, or because user has moved away from the AP since the last scan). If a failure happens, VCAPS waits until the beginning of next timeslot and retries. If more than 5 transfer attempts through a particular AP fail, then VCAPS blacklists that AP and waits for 20 minutes before re-using it. Re-using a blacklisted AP allows us to use a different AP that may potentially have the same SSID but provides good performance. We have observed several instances of different WiFi APs using the same SSID during our trace collection, especially on the USC campus network.

We implemented all the features of SALSA described in §3 on the smartphone including the algorithm for time-varying V and the rate estimation scheme. However, there are few minor differences between the SALSA implementation used for simulations, and our smartphone implementation. These differences are driven by real-world considerations. First, unlike the simulator which treated the

[1]Some of these videos can be viewed at `http://tomography.usc.edu`. A portion of the corpus is not publicly viewable because of privacy reasons.

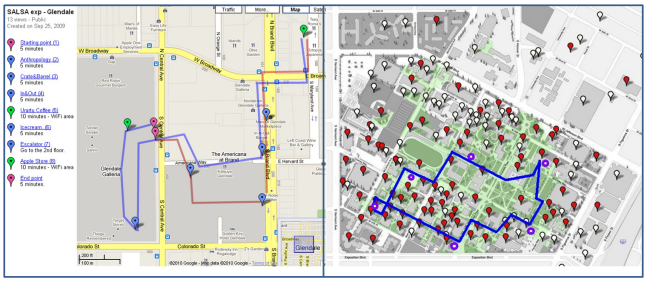

Figure 17: Exp. Walk Route

queue as a bag of bits, our implementation uses HTTP POST and attempts to transfer fixed size video chunks. These chunk transfers may fail and need to be retried. Our simulator, on the other hand, determines how many bits could have been transferred given the rate estimate, and then determines, using a weighted coin toss, whether that transfer would have resulted in a failure. This behavior was designed to mimic the theory, more than the implementation.

Second, the implementation uses the online rate estimation, but the simulator does not. Rather, the simulator simply uses the rate estimates and the failure probabilities derived from the trace. The implementation is potentially more accurate in this regard, because it learns the actual achievable rate from successful transfers.

Finally, like the simulator, our implementation also uses a fixed nominal value for the power expenditure P_C and P_W. An implementation has the potential to be more accurate if the OS were to provide fine-grained energy usage measurements, but the Symbian OS does not do this.

Results. We have conducted extensive experiments using our prototype implementation for evaluating SALSA with five different parameters. The goal of this experiment was two fold: first, to demonstrate that our Urban Tomography system with SALSA works robustly for several hours; and second, to validate that the performance of SALSA under real-world settings is consistent with our simulation results, despite the differences between the simulation and the implementation.

In our experiments, one volunteer carried five phones each configured with different values of α, and conducted 5 walks (each for approximately 3 hours) both on the USC campus and at the Glendale Galleria mall. The routes through the USC campus and Glendale Galleria are shown in Figure 17. Each phone was programmed to use the same arrival trace, obtained from one of the events recorded by users of the Urban Tomography system (i.e., we replayed, on the phone, the arrival of videos in the event). Each walk completed the upload of all videos associated with the same arrival trace. Thus, our real-world experiment corresponds to, in the terminology of §4.1, one arrival trace, and 10 link traces.

On each phone, we recorded the transfer decisions made, the average delay from the creation to transfer completion for each video, all WiFi scan results, the achieved rates, the size, and the duration of each transfer. Using this, and nominal values of the energy consumption of cellular and WiFi transfers, we were able to plot the performance of the SALSA algorithm in the *E-D* plane.

Figure 15 and Figure 16 each depict the results from the USC campus and the Glendale Galleria mall. On each figure, the 5 small black crosses each correspond to one walk. For comparison, the dots in the background on each graph depict results from our trace-driven simulations for that particular environment and α value. We say that an experiment is consistent with simulation if the experimental results fall within the envelope obtained by the simulation. Consistency implies that the differences between the simulation and implementation are not significant, and that the envelope obtained by simulation may be a reasonable indicator of performance observed in the real-world.

As Figures 15 and 16 show, all experimental data points fell within the corresponding envelopes. (In the Glendale experiment, there are a few instances for $\alpha = 0.4$ which were just on the border of the corresponding envelope; in all other cases, the experimental points were well within the envelope.) This result is encouraging: we believe that, over a wide range of parameters, SALSA will perform well in the real-world. We intend to incorporate SALSA into our VCAPS software distribution, so that our user base can obtain the performance benefits it provides.

6. RELATED WORK

Two preliminary pieces of work have inspired our own. Zaharia et al. [26] consider the same problem but assume that each network interface knows its future availability and has a fixed rate. Seth et al. [23] also consider supporting delay-tolerant applications, but focus on an approach to seamlessly manage multiple network interfaces of varying availability, relieving the programmer of this burden. In their approach, users or applications specify an overall objective, like a delay-bound, and their runtime system attempts to achieve this objective by ensuring that the progress of data transfer is at a rate that will satisfy the application objective, while having the freedom to pick the appropriate link. Our problem statement is slightly different, since we do not attempt to guarantee a fixed delay bound, and instead focus on minimizing energy.

Next closest to our work is prior work on achieving energy efficiency in smartphone applications by exploiting multiple wireless interfaces. Context-for-Wireless [22] uses the history of context information to decide whether it is beneficial, in terms of energy, to use 3G/EDGE or WiFi for data transfer. They attempt to intelligently learn and estimate WiFi network conditions without powering up the WiFi interface so as to save the energy cost of turning on the interface and re-scanning for available APs. Armstrong et al. [6] also discuss a similar problem. They report that there exist a threshold message size (30KB in their application on HP iPAQ 6325 platform) for which using WiFi is more energy efficient than 3G/EDGE, due to the wake-up cost of WiFi interfaces. However, their focus is on designing a web proxy system to reduce the size of the updated content for efficient data downloads. CoolSpots [20] aims to reduce the power consumption of wireless mobile devices with multiple radio interfaces by intelligently deciding whether and when to use WiFi and Bluetooth based on an application's bandwidth requirement. None of these pieces of work trades-off delay for reduced energy: rather, they are interested in determining the lowest energy link among a set of available links at a given instant. Moreover, our work is focused on larger data traffic (our videos in VCAPS have from a few hundreds of KBytes to a hundred MBytes) than some of these applications, and their emphasis on WiFi wake-up costs do not apply in our case, since the wake-up cost can be amortized over these larger transfers.

Other pieces of work, in slightly different contexts, have attempted to exploit multiple radio interfaces to improve energy efficiency on smartphones. Micro-Blog [10] is an application for sharing and querying content through mobile phones and social participation. Its localization component aims to save energy by adaptively changing between three different localization schemes (GPS, WiFi-based, GSM-based) considering energy cost and localization accuracy requirement. Cell2Notify [4] is an energy management architecture that leverages the cellular radio signal to wake-up the high-energy consumption WiFi radio for VoIP applications. Finally, COMBINE [5] leverages 3G/EDGE links of its wireless LAN peers to cooperatively download data. However, it focuses on throughput

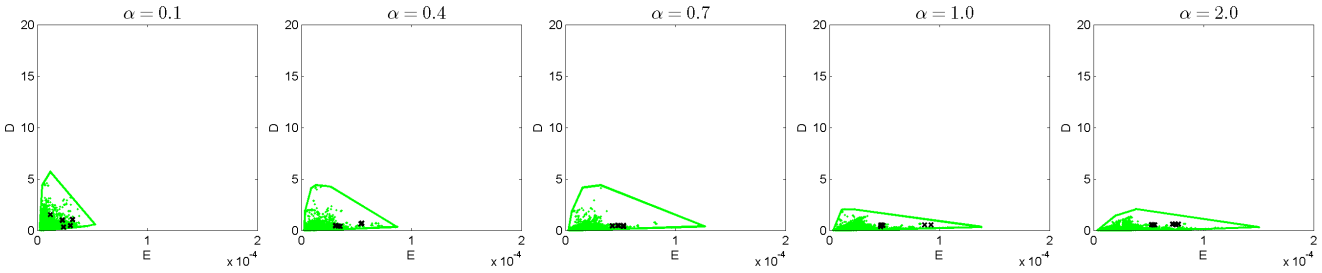

Figure 15: Experimental result at the USC Campus compared to simulation results

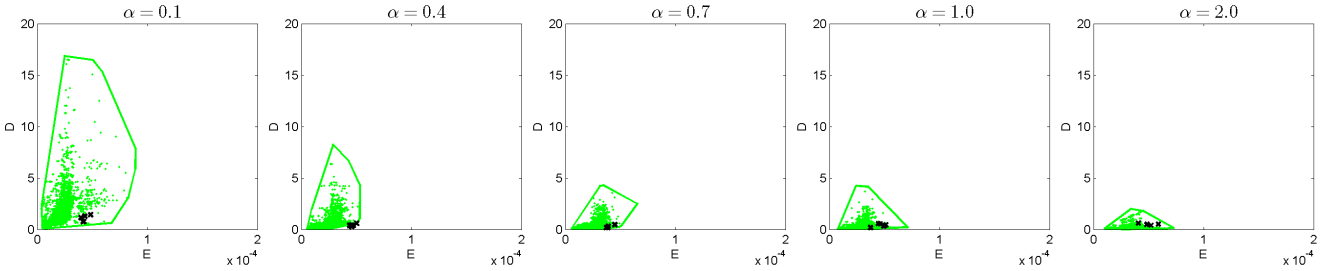

Figure 16: Experimental result at Shopping Mall compared to simulation results

enhancement rather than energy, and does not consider an intermittently connected WiFi as the download/upload link.

BreadCrumbs [17] examines WiFi connectivity changes over time and provides mobile *connectivity forecasts* by building a predictive mobility model. These forecasts can be used to more intelligently schedule network usage. This work can be complementary to ours: e.g. SALSA could benefit from this technique and determine relevant α settings. A similar benefit can be obtained from WiFi databases obtained opportunistically [19, 22].

In a different context, for a networked setting with multiple nodes transmitting data over wireless links, Neely [16] developed a joint transmit power and transmission scheduling algorithm (EECA) that minimizes the total system power consumption. There are two key differences between EECA and SALSA: (i) EECA assumes that each node has a single wireless interface whereas SALSA is designed for smartphones with multiple wireless interfaces, and (ii) EECA focuses on transmit power control while in SALSA, we assume that the transmit power on each wireless interface is fixed. Neely also discusses a variant of EECA that maximizes throughput given average power constraint. EECA has not been evaluated in implementation, and does not specify how to determine the value of the control parameter V automatically. Georgiadis et al. [11] discuss several stable control algorithms for maximizing throughput or fair rate allocation in wired and wireless networks derived using the Lyapunov optimization framework in their book [11].

Finally, there is a large literature on smartphone applications [3, 14, 8, 10, 1, 2, 25, 7] that are data-intensive but delay-tolerant applications. Some of them (e.g., [8]) implement greedy delay-tolerant strategies, like handing off data to the first available peer or access point, and do not explicitly consider the energy/delay trade-off in their designs. In that sense, they are closer to the work on delay-tolerant networks [9]. Finally, research in sensor network energy management has explicitly considered the energy-delay trade-off to increase network lifetime [21, 12], but only in the context of a single wireless interface.

7. CONCLUSIONS AND FUTURE WORK

SALSA is a near-optimal algorithm for performing the energy-delay tradeoff in bandwidth-intensive delay-tolerant smartphone applications. Its transmission decisions take several factors into account: data backlog, power cost of the wireless interface, and channel quality. Algorithms which lack even one of these factors in the transmission decisions perform significantly worse. Finally, SALSA solves a real problem: many of the users of our system have collected videos for which the total transmission cost, as well as the savings obtained by SALSA, are a noticeable fraction of the overall battery capacity. In future work, we hope to get more experience with SALSA deployments from our diverse user base.

Acknowledgements.

We would like to thank our shepherd, Robin Kravets, and the anonymous referees, for their insightful suggestions for improving the technical content and presentation of the paper.

8. REFERENCES

[1] Cyclesense. `http://urban.cens.ucla.edu/projects/cyclesense/`.

[2] Dietsense. `http://urban.cens.ucla.edu/projects/dietsense/`.

[3] Peir: Personal environmental impact report. `http://peir.cens.ucla.edu/`.

[4] Y. Agarwal, R. Chandra, A. Wolman, P. Bahl, K. Chin, and R. Gupta. "Wireless Wakeups Revisited: Energy Management for VoIP over Wi-Fi Smartphones". In *MobiSys'07*, 2007.

[5] G. Ananthanarayanan, V. N. Padmanabhan, L. Ravindranath, and C. A. Thekkath. "COMBINE: Leveraging the Power of Wireless Peers through Collaborative Downloading". In *MobiSys'07*, 2007.

[6] T. Armstrong, O. Trescases, C. Amza, and E. de Lara. "Efficient and Transparent Dynamic Content Updates for Mobile Clients". In *MobiSys'06*, 2006.

[7] X. Bao and R. R. Choudhury. "VUPoints: Collaborative Sensing and Video Recording through Mobile Phones". In *Mobiheld '09*, 2009.

[8] S. B. Eisenman, E. Miluzzo, N. D. Lane, R. A. Peterson, G.-S. Ahn, and A. T. Campbell. "The BikeNet mobile sensing system for cyclist experience mapping". In *SenSys'07*, 2007.

[9] K. Fall. "A Delay-Tolerant Network Architecture for Challenged Internets". In *SIGCOMM '03*, 2003.

[10] S. Gaonkar, J. Li, R. R. Choudhury, L. Cox, and A. Schmidt. "Micro-Blog: Sharing and Querying Content Through Mobile Phones and Social Participation". In *MobiSys'08*, 2008.

[11] L. Georgiadis, M. J. Neely, and L. Tassiulas. *"Resource Allocation and Cross-Layer Control in Wireless Networks"*. Foundations and Trends in Networking, 2006.

[12] O. Gnawali, J. Na, and R. Govindan. "Application-Informed Radio Duty-Cycling in a Re-Taskable Multi-User Sensing System". In *IPSN'09*, 2009.

[13] M. H. Krieger, R. Govindan, M.-R. Ra, and J. Paek. Commentary: Pervasive urban media documentation. *Journal of Planning Education and Research (JPER)*, 29(1):114–116, 2009.

[14] P. Mohan, V. N. Padmanabhan, and R. Ramjee. "Nericell: rich monitoring of road and traffic conditions using mobile smartphones". In *SenSys'08*, Nov. 2008.

[15] Monsoon Solutions Inc. Power Monitor. `http://www.msoon.com/LabEquipment/PowerMonitor/`.

[16] M. J. Neely. "Energy Optimal Control for Time Varying Wireless Networks". *IEEE Transactions on Information Theory*, 52(7):2915–2934, 2006.

[17] A. J. Nicholson and B. D. Noble. "BreadCrumbs: Forecasting Mobile Connectivity". In *MobiCom'08*, 2008.

[18] Nokia Corp. Nokia Energy Profiler. `http://www.forum.nokia.com/Tools_Docs_and_Code/Tools/Plug-ins/Enablers/Nokia_Energy_Profiler/`.

[19] J. Pang, B. Greenstein, M. Kaminsky, D. McCoy, and S. Seshan. "Wifi-Reports: Improving Wireless Network Selection with Collaboration". In *Mobisys '09*, 2009.

[20] T. Pering, Y. Agarwal, R. Gupta, and R. Want. "CoolSpots: reducing the power consumption of wireless mobile devices with multiple radio interfaces". In *MobiSys'06*, 2006.

[21] J. Polastre, J. Hill, and D. Culler. "Versatile Low Power Media Access for Wireless Sensor Networks". In *SenSys'04*, 2004.

[22] A. Rahmati and L. Zhong. "Context-for-Wireless: Context-Sensitive Energy-Efficient Wireless Data Transfer". In *MobiSys'07*, 2007.

[23] A. Seth, M. Zaharia, S. Keshav, and S. Bhattacharyya. A policy oriented architecture for opportunistic communication on multiple wireless networks, 2006.

[24] Skyhook Wireless. `http://www.skyhookwireless.com/`.

[25] USC/ENL. VCAPS: Urban Tomography Project. `http://tomography.usc.edu/`.

[26] M. Zaharia and S. Keshav. Fast and optimal scheduling over multiple network interfaces. Technical Report CS-2007-36, University of Waterloo, Oct. 2007.

APPENDIX

A. DERIVATION OF SALSA CONTROL DECISION

In this section, we describe the derivation of SALSA's control decision (5). Following standard practice, we define the Lyapunov function as:

$$L(U[t]) \triangleq \frac{1}{2}(U[t])^2 \tag{11}$$

and the one-step Lyapunov drift $\Delta(U[t])$ as:

$$\Delta(U[t]) \triangleq \mathbb{E}\{L(U[t+1]) - L(U[t]) \mid U[t]\} \tag{12}$$

The Lyapunov drift for the queue backlog $U[t]$ is specified by the following lemma.

Lemma 1 Suppose the data arrivals $A[t]$ and the wireless link quality $\vec{S}[t]$ are i.i.d. over timeslots. For the queue dynamics in (2) and the Lyapunov function in (11), the one-step Lyapunov drift satisfies the following constraint for all t and $U[t]$:

$$\Delta(U[t]) \leq B[t] - U[t]\,\mathbb{E}\{\mu[t] \mid U[t]\} + U[t]\,\lambda \tag{13}$$

with $B[t]$ equal to:

$$B[t] \triangleq \frac{1}{2}\left(\mathbb{E}\left\{A^2[t]\right\} + \mathbb{E}\left\{\mu^2[t] \mid U[t]\right\}\right) \tag{14}$$

PROOF. From (2) we have:

$$\begin{aligned}\frac{1}{2}U[t+1]^2 &= \frac{1}{2}\left(U[t] - \mu[t] + A[t]\right)^2 \\ &\leq \frac{1}{2}\left(U[t]^2 + \mu[t]^2 + A[t]^2\right) - U[t]\left(\mu[t] - A[t]\right)\end{aligned}$$

and we take conditional expectations given $U[t]$:

$$\Delta(U[t]) \leq \frac{1}{2}\mathbb{E}\left\{\mu[t]^2 + A[t]^2 \mid U[t]\right\} - U[t]\mathbb{E}\{\mu[t] - A[t] \mid U[t]\}$$

Now, using (14) and $\mathbb{E}\{A[t] \mid U[t]\} = \mathbb{E}\{A[t]\} = \lambda$, we obtain (13). □

Power Constraint. To include the minimum power consumption objective into the Lyapunov drift, following the Lyapunov optimization framework [11, 16] , we add a weighted cost (power consumption during slot t) to (13) to get:

$$\begin{aligned}&\Delta(U[t]) + V\mathbb{E}\{P[t] \mid U[t]\} \\ &\leq B[t] - U[t]\mathbb{E}\{\mu[t] \mid U[t]\} + U[t]\lambda + V\mathbb{E}\{P[t] \mid U[t]\} \quad (15)\\ &= B[t] - U[t]\mathbb{E}\{\mathbb{E}\{\mu[t] \mid U[t], l, S_l[t], P[t]\}\} \\ &\qquad + U[t]\lambda + V\mathbb{E}\{P[t] \mid U[t]\} \quad (16)\\ &= B[t] - \mathbb{E}\{(U[t]\mathbb{E}\{\mu[t] \mid l, S_l[t], P[t]\} - VP[t]) \mid U[t]\} \\ &\qquad + U[t]\lambda \quad (17)\end{aligned}$$

where (16) follows from (15) and (1) using iterated expectations, and (17) is derived by switching the order of expectations in (16).

We assume that the data arrival process $A[t]$, and the transmission process $\mu[t]$ have finite variance; implying that there exist constants, $\mathscr{A}^2$ and μ^2, such that $\mathbb{E}\left\{A^2[t]\right\} < \mathscr{A}^2$ and $\mathbb{E}\left\{\mu^2[t] \mid U[t]\right\} < \mu^2$. Hence, from (14), we have

$$B[t] < B \quad \forall t, \quad B = \mathscr{A}^2 + \mu^2 \tag{18}$$

Minimizing the RHS of (17) will guarantee queue stability with minimal power consumption, as per the Lyapunov framework. However, since we have no control over the application data arrival process, and hence cannot do much about the term $U[t]\lambda$. SALSA does not directly minimize the entire expression on the RHS of (17). Rather, in order to minimize the RHS of (17), it maximizes the negative term on the RHS of (17); hence explanation for the control decision in (5).

B. PROOF OF THEOREM 1

PROOF. Our proof is similar to the proof of an analogous result proved by Neely in the context of energy efficient transmission scheduling in wireless networks [16]. It builds upon the properties of *stationary randomized* policies for making control decisions. In our context, such a policy would make (randomized) link selection decisions based only on the current arrivals $A[t]$ and link quality $\vec{S}[t]$ that are i.i.d. over slots and independent of the current queue backlog $U[t]$. In practice, a stationary randomized link selection policy cannot be defined without prior knowledge of the probability distributions p_A and π_s for the arrival process $A[t]$ and the wireless link quality $S[t]$, respectively.

Since we assume that the arrival process is strictly within the network capacity region, there exists at least one stationary randomized control policy that can stabilize the queue [16], which has following features:

$$\begin{aligned} \mathbb{E}\{P[t]\} &= P^* && (19)\\ \mathbb{E}\{\mu[t] \mid U[t]\} &\geq \lambda \\ \Rightarrow \mathbb{E}\{\mu[t] \mid U[t]\} &= \lambda+\varepsilon,\ \varepsilon \geq 0 && (20) \end{aligned}$$

where we define P^* as minimum achievable power expenditure using any control policy that achieves queue stability. Then, by applying (19)-(20) to (15), we obtain:

$$\Delta(U[t]) + V\mathbb{E}\{P[t] \mid U[t]\} \leq B[t] - U[t](\lambda+\varepsilon) + U[t]\lambda + VP^* \quad (21)$$

(21) holds for all timeslot t. Taking an expection for (21) with respect to the distribution of $U[t]$ and using iterative expectation law results in:

$$\mathbb{E}\{L(U[t+1]) - L(U[t])\} + V\mathbb{E}\{P[t]\} \leq B - \varepsilon\mathbb{E}\{U[t]\} + VP^*$$

Then, summing over all timeslots $t \in \{0,1,...,T-1\}$ and dividing by T yields:

$$\frac{\mathbb{E}\{L(U[T]) - L(U[0])\}}{T} + \frac{V}{T}\sum_{\tau=0}^{T-1}\mathbb{E}\{P[\tau]\} \quad \leq \quad B - \frac{\varepsilon}{T}\sum_{\tau=0}^{T-1}\mathbb{E}\{U[\tau]\} + VP^* \quad (22)$$

Since Lyapunov function is non-negative by definition and so is $P[t]$, a simple manipulation of (22) yields:

$$\frac{1}{T}\sum_{\tau=0}^{T-1}\mathbb{E}\{U[\tau]\} \leq \frac{B + VP^* + \mathbb{E}\{L(U[0])\}/T}{\varepsilon}$$

Taking limits as $T \to \infty$ results in the time average backlog bound (6).

By similarly manipulating (22) we obtain:

$$\frac{1}{T}\sum_{\tau=0}^{T-1}\mathbb{E}\{P[\tau]\} \leq P^* + \frac{B}{V} + \frac{\mathbb{E}\{L(U[0])\}}{VT} \quad (23)$$

Again taking limits as $T \to \infty$ yields equation (7). □