SmartTransfer: Transferring Your Mobile Multimedia Contents at the "Right" Time

Yichuan Wang, Xin Liu
Dept. of Computer Science
University of California, Davis
Davis, CA 95616
yicwang@ucdavis.edu,
xinliu@ucdavis.edu

Angela Nicoara
Deutsche Telekom Innovation Laboratories
Silicon Valley Innovation Center
Mountain View CA 94043
angela.nicoara@telekom.com

Ting-An Lin, Cheng-Hsin Hsu
Dept. of Computer Science
National Tsing Hua University
Hsin Chu, Taiwan
tim19890901@gmail.com,
chsu@cs.nthu.edu.tw

ABSTRACT

Today's mobile Internet is heavily overloaded by the increasing demand and capability of mobile devices, in particular, multimedia traffic. However, not all traffic is created equal, and a large portion of multimedia contents on the mobile Internet is delay tolerant. We study the problem of capitalizing the content transfer opportunities under better network conditions via postponing the transfers without violating the user-specified deadlines. We propose a new framework called SmartTransfer, which offers a unified content transfer interface to mobile applications. We also develop two scheduling algorithms to opportunistically schedule the content transfers. Via extensive trace-driven simulations, we show that our algorithms outperform a baseline scheduling algorithm by far: up to 17 times improvement in upload throughput and/or at most 20 dBm boost in signal strength. The simulation results also reveal various tradeoff between the two proposed scheduling algorithms. We have implemented our framework and one of the scheduling algorithms on Android, to demonstrate their practicality and efficiency.

Categories and Subject Descriptors

C.2.4 [**Distributed Systems**]: Distributed Applications

General Terms

Performance

Keywords

Opportunistic transfers, scheduling, resource conservation, user profiling

1. INTRODUCTION

Recent mobile devices can generate and render multimedia contents, such as audios, images, and videos. These mobile devices, however, are resource-constrained and have to frequently transfer the multimedia contents from/to some back-end servers or clouds. Cisco reports that mobile data is expected to increase almost 40 times by 2015 and 66% of this increase is due to mobile video [1]. In fact, the service providers have moved away from unlimited dataplans to tiered services [2] and may even consider time-dependent pricing [9], which could increase the bills on the mobile users. Therefore, modern mobile devices must regulate their network resource consumption, i.e., become more *network-friendly*.

To achieve this goal, we study an innovative approach of regulating the network resource consumed by each mobile user. A key observation is that *not all traffic is created equal:* (i) Many popular mobile applications are *delay tolerant.* The delay tolerance of such traffic varies from subseconds to hours; and (ii) Data from a large cellular network shows that there exists *a significant lag between content generation and user-initiated upload time,* more than 55% multimedia contents uploaded from mobile network is at least 1 day old [17]. We call such traffic *elastic*, which can be leveraged to opportunistically schedule content transfers when the network condition is more favorable in terms of, e.g., signal strength, network throughput, energy consumption, and access price. In other words, since mobile users perceive time-varying channel condition and network load, choosing the "right" time to transfer each multimedia content is critical to system performance.

In this paper, we study how to capitalize the content transfer opportunities under better network conditions via postponing the transfers without violating the user-specified deadlines. More specifically, when delay-tolerant content is generated at a mobile device, one can opportunistically schedule the transfer to minimize network/device resource utilization, such as network air time or battery consumption. Making such a decision (throughout the paper) is challenging for two reasons. First, implementing the decision making logics in individual mobile applications is time consuming, expensive, and error-prone. Hence, such optimization techniques are unlikely to be adopted by mobile application developers; many of them are freelance programmers. We address this challenge by adding a new module called *SmartTransfer* in mobile operating systems, which provides a unified content transfer interface to applications. We present the SmartTransfer framework in Sec. 2.

NOSSDAV'12, June 7–8, 2012, Toronto, Ontario, Canada.

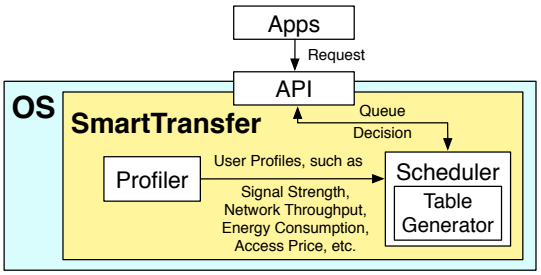

Figure 1: The proposed SmartTransfer framework.

Second, making such decisions requires prediction on future network conditions, e.g., how soon and likely a mobile device will enter a region with strong cellular network signals. A simple approach is to use the moving average of past few samples [4, 13]. While this approach may work for small time scales (e.g., in a few minutes), it is less applicable to large time scales because a mobile user may visit several different locations and experience diverse network conditions. We address this challenge by collecting longer, say 30 days, historical data on mobile devices, which are referred to as user *profiles* throughout this paper. Individual human mobility in large time scales is highly predictable [7, 16], therefore profiles allow us to make more accurate predictions because mobile user's future network conditions are time and location dependent. We present how we leverage user profiles to design content transfer scheduling algorithms in Sec. 3.

We evaluate the proposed framework and scheduling algorithms in Sec. 4 via extensive trace-driven simulations. The simulation results reveal that the proposed algorithms outperform a baseline algorithm by up to 17 times in terms of upload throughput and/or 20 dBm in terms of signal strength. One of our proposed algorithms employs a more comprehensive statistical model, and demands for more resources. The other proposed algorithm employs a lightweight model, and always terminates in 560 msecs throughout all simulations. In Sec. 5, we implement the SmartTransfer framework and OSS_L algorithm on Android and deploy a delay-tolerant application via the Android Market, which demonstrate the practicality of the proposed solution.

2. SMARTTRANSFER FRAMEWORK

Fig. 1 presents the SmartTransfer framework, which runs on mobile devices and consists of three components: *profiler*, *scheduler*, and *API* (Application Programming Interface). The profiler collects user profiles, which are essentially timestamped log files. The profiler itself can be general that collects the network conditions in various performance metrics, including signal strength, network throughput, energy consumption, and access price. Some of these metrics can be retrieved from the network interfaces, such as the signal strength and network throughput, some of them may be measured by on-board instruments, such as energy consumption, and some of them could be provided by the cellular service providers, such as the access price. The profiler monitors the profile size to avoid filling up the storage space. It also keeps the updated profile, and provides the most recent, e.g., last three-month, profile to the scheduler. The profile provides input to the scheduler that runs the scheduling algorithms to make decisions on when to start content transfer. For scheduling algorithms that cannot run in real-time, we may pre-process the user profiles while the mobile devices are charging and idling, in order to eliminate the negative impacts on the user experience. The output of the pre-processing step is typically a lookup table, and thus is referred to as *table generator* as illustrated in Fig. 1. In extreme cases, where the scheduling algorithms are too complicated for mobile devices, we may even offload the table generator to the cloud. Mobile applications connect to the framework via the API and submit content transfer requests, including the content size and delay requirement. The SmartTransfer then helps the mobile applications to schedule the user-generated multimedia contents to upload them under good network conditions; they may also help mobile applications to prefetch certain contents for later usage, perhaps chosen by a content recommender [3], under good network conditions. The SmartTransfer framework supports different optimization criteria, such as minimizing: (i) network resource consumption, (ii) transfer duration, (iii) energy consumption, and (iv) access cost. This is achieved by choosing the corresponding user profiles and scheduling algorithms.

3. SCHEDULING ALGORITHMS

3.1 Scheduling Model

In this section, we present a slotted scheduling model. We consider a transfer request from the application layer, which is available at time zero. There exists a hard deadline called *horizon* N, by which the data transfer must be completed otherwise user experience will be disrupted. N is a system parameter, which may be derived via experimental studies on user tolerance on delay, and could be application-dependent. To maximize user experience, a user can request an instantaneous data transfer, via a user interface, at any time slot. We model the time slot in which such instantaneous transfer request occurs by a random variable M called *freeze time*. The realization of M is unknown a priori. For simplicity, we consider a content transfer that can be finished in a time slot, while it is our future work to generalizing our analysis to multiple and heterogeneous content transfers. If user did not request an instantaneous data transfer at time slot t ($M > t$), the scheduler makes a *decision* $D_t \in \{\text{Wait}, \text{Transfer}\}$.

Fig. 2 summarizes the considered scheduling model. A content is transferred at time slot t if (i) $t = N$, or (ii) $M = t$, or (iii) $D_t = \text{Transfer}$. If $D_t = \text{Wait}$, the content is delayed to time slot $t + 1$.

Let X_t ($t \in [1, N]$) be the *transfer cost* at slot t. Waiting costs nothing. Let V_t be the *optimal cost* to transfer a content between time slot t and N, assuming the optimal schedule is applied. V_t can be calculated using the statistics of X_t and M, which are derived from the user profile. In Secs. 3.2 and 3.3, we develop two scheduling algorithms, which employ different statistical models for the transfer costs X_t, under different assumptions.

3.2 Optimal Stopping Scheduling (OSS)

We propose a scheduling algorithm that minimizes the expected cost. Our algorithm is inspired by earlier work on

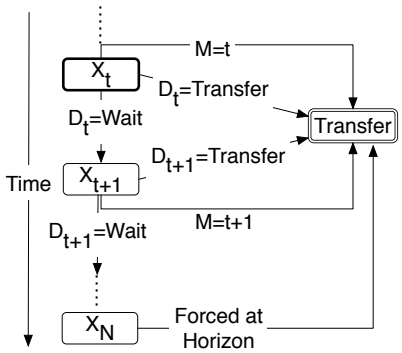

Figure 2: The considered scheduling model.

classic optimal stopping problems [6, 14] that try to maximize the probability of finding the best candidate in a job interview. To the best of our knowledge, the optimal stopping problem for minimizing the expected cost with random freeze time has not been rigorously studied. We refer to our algorithm as optimal stopping scheduling (OSS).

The OSS algorithm employs a Markovian model to capture the transfer costs. In this model, the transfer cost X_t depends on time, and previous transfer cost X_{t-1}. Statistics for M and X are derived from user profiles.

The principle of optimality is as follows. If transfer cost X_t is less or equal to the expect optimal cost $E(V_{t+1})$, transfer at the time slot t, otherwise wait until time slot $t+1$. With the above notations, the optimal schedule is written as:

$$D_t = \begin{cases} \text{Transfer}, & M = t; \\ \text{Transfer}, & M > t, X_t \leq E(V_{t+1}|X_t); \\ \text{Wait}, & M > t, X_t > E(V_{t+1}|X_t). \end{cases} \quad (1)$$

Due to its Markovian property, $E(V_t|X_{t-1})$ can be obtained by backward induction:

$$\begin{aligned} E(V_N|X_{N-1}) =& E(X_N|X_{N-1}); \\ E(V_t|X_{t-1}) =& P(M = t|M \geq t) \cdot E(X_t|X_{t-1}) \\ &+ P(M > t|M \geq t)\cdot \\ &\sum_c \{P(X_t = c|X_{t-1}) \cdot \min(c, E(V_{t+1}|X_t = c))\}. \end{aligned} \quad (2)$$

On mobile devices, at charging time, the optimal decision D_t for all time and previous cost X_{t-1} can be pre-calculated and saved as a table. At runtime, we simply look up the table using previous cost and time to find the optimal decision. We call this pre-calculated data as the *decision* table.

The model presented in Eq. (1) and (2) are general, but may come with some limitations. First, we need to accumulate enough samples to accurately derive $E(X_t|X_{t-1})$ for all t and X_{t-1}. Hence, the OSS algorithm requires a long user profile for deriving model parameters. Second, let T be the number of total time slot, and $|X|$ be the number of all possible transfer costs. The decision table has $|X| \cdot T \cdot N$ elements, which is huge in the time scale that we are interested in implementing the algorithm, e.g., one day. Computing and storing the optimal schedule may be too demanding for mobile devices. For example, consider 1 minute time slot in 1 day, 1 hour horizon, 30 different cost values, the decision table is as large as 2.6 MB, which is significant to mobile devices.

3.3 Lightweight Optimal Stopping Scheduling (OSS_L)

To alleviate the limitation of the OSS algorithm, we propose a simplified optimal stopping algorithm. The Lightweight Optimal Stopping Scheduling (OSS_L) algorithm leverages the same principle of optimality as the OSS algorithm, but has a more relaxed assumption on the statistical model of the transfer costs. In particular, OSS_L assumes that X_t are independent over time.

Following similar derivations of the OSS algorithm, we write the optimal decision as:

$$D_t = \begin{cases} \text{Transfer}, & M = t; \\ \text{Transfer}, & M > t, X_t \leq E(V_{t+1}); \\ \text{Wait}, & M > t, X_t > E(V_{t+1}). \end{cases} \quad (3)$$

where

$$\begin{aligned} E(V_N) =& E(X_N); \\ E(V_t) =& P(M = t|M \geq t) \cdot E(X_t) \\ &+ P(M > t|M \geq t) \\ &\cdot [P(X_t \leq E(V_{t+1})) \cdot E(X_t|X_t \leq E(V_{t+1})) \\ &+ P(X_t > E(V_{t+1})) \cdot E(V_{t+1})]. \end{aligned} \quad (4)$$

Note that the $E(V_t)$ $(1 \leq t \leq N)$ can also be derived using backward induction, which starts from $V_N = E(X_N)$ as the transfer must be done by time slot N regardless the cost. Then, the rest of $E(V_t)$ can be computed using the user profile, which reveals various statistics of X_t such as $P(X_t \leq E(V_{t+1}))$ and $E(X_t|X_t \leq E(V_{t+1}))$. We precompute all $E(V_t)$ $(1 \leq t \leq N)$ using Eq. (4), and refer to it as the *threshold* table. Upon optimum $E(V_t)$ $(1 \leq t \leq N)$ are derived, the proposed scheduler computes optimal schedule using Eq. (3). Compared the OSS algorithm, the OSS_L algorithm employs a simpler underneath model. Hence, OSS_L can be trained with shorter user profiles, at the expense of potentially higher inaccuracy.

4. TRACE-DRIVEN SIMULATION

4.1 Simulation Setup

For fair comparisons, we adopt a set of 3-month long traces collected from 12 North American Android users, who are heterogeneous in terms of demography, sex, and age. The traces were collected and used in another project [11], and we process the raw trace files to extract the relevant measurements: Received Signal Strength Indication (RSSI), and upload/download throughput. A higher value in RSSI or throughput reduces the transmission time of data, in turn reduces network resource and battery consumption. In the current work, we only consider data collected from the cellular network; data from WiFi is excluded. The resulting traces are used to drive the simulator.

The simulator is developed in Matlab and run on a Linux PC with a 2.8 GHz Intel CPU. The simulator implements a time slotted system. Contents arrive to the system follows a Poisson process. Four algorithms are implemented to schedule the content transfers. The first one is a baseline algorithm Instant transfer (INS). INS schedules a content to

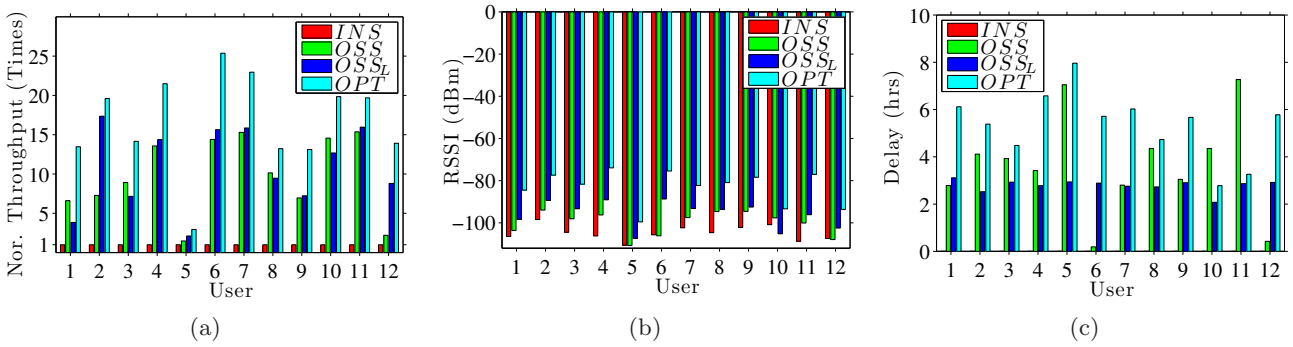

Figure 3: (a) Normalized throughput achieved by the scheduling algorithms, (b) signal strength achieved by the scheduling algorithms, and (c) delay due to the scheduling algorithms, INS leads to no delay.

transfer right after it becomes available. The second algorithm is the OSS algorithm introduced in Section 3.2. Part of the traces are passed to OSS algorithm as user profile. With user profile, OSS generates a decision table for each user. After a content becomes available, each time slot, the current optimization criteria, either RSSI or upload throughput, and current time is used to look up the decision table and find the current action (Transfer or Wait). The third algorithm is the OSS_L algorithm presented in Section 3.3. OSS_L also employs traces as user profile, however OSS_L generates a threshold for each time slot using a simplified model. Each time slot, OSS_L compares the current optimization criteria with the corresponding threshold. OSS_L transfer the content if the current value is better than the threshold. Finally, an offline algorithm OPT is implemented as an upper bound for scheduling algorithms. We assume that OPT knows all the future RSSI or throughput at the beginning. After each content becomes available, OPT will choose the time slot with the highest RSSI or throughput to transfer the content, which is of course not realistic.

If not otherwise specified, we let the user profile be half of the trace length, time horizon be 8 hours, use the upload throughput as the optimization criterion, and schedule on average 32 content transfers per day. We also study the implications of varying user profile length and time horizon. Each content is scheduled by all four algorithms, the resulting RSSI or throughput, along with incurred delay are collected. The decision table or threshold table size and the time taken to generate these tables are also collected.

4.2 Simulation Results

We first report the simulation results with default parameters. We then study the implications of the parameters. We also present computation time and memory consumption of the proposed algorithms.

Throughput optimized scheduling. We use the results from INS as the baseline, and compute the relative upload throughput of the proposed algorithms. We plot the throughput results in Fig. 3(a). This figure shows that the proposed OSS and OSS_L algorithms always outperform the INS algorithm, and in the extreme case by up to 17 times. This demonstrate the benefits of the proposed algorithms.

Signal strength optimized scheduling. We plot the RSSI values in Fig. 3(b). This figure reveals that the SmartTransfer framework supports various optimization criteria. Moreover, the proposed OSS and OSS_L algorithms outperform the INS algorithm for most users, except for users 5, 10, and 12, in which the RSSI values resulting from the OSS and OSS_L may be slightly worse than that of the INS algorithm. We took a closer look at the traces, and found that these users tend to spend a very long time, e.g., a day, at a single location. Therefore, their RSSI values over each day are rather static, and thus our proposed algorithms may not result in too much improvement. We also plot the delay of content transfers in Fig. 3(c).

Implications of profile length. For individual users, we plot the average upload throughput achieved by each algorithm under various profile lengths, between 1 and 32 days. The figures are not shown due to the space limitations. We found that the OSS and OSS_L significantly outperform the INS algorithm. Moreover, the resulting throughput is generally higher with longer profile length. Next, we report the overhead of the proposed algorithms. The decision table of the OSS algorithm contains 15256 elements, while the threshold table of the OSS_L algorithm only has 4920 elements. As a consequence, the OSS_L algorithm runs much faster than OSS: less than 200 msecs versus up to 44 secs. This shows that OSS_L is preferable when the computational resources are scarce.

Implications of time horizon. Fig. 4 reports the sample results from user 3 with various time horizons. Fig. 4(a) indicates that the OSS algorithm outperforms the OSS_L algorithm when the time horizon is shorter than 12.5 hrs, while the OSS_L performs better when time horizon is longer. This means that OSS_L is more suitable to content transfers that are tolerant to longer delays. Fig. 4(b) plots the running time of the two algorithms. We make two observations. First, the OSS algorithm does not scale well with the time horizon, the running time is 100 secs when time horizon is 32 hrs. Second, the OSS_L algorithm runs efficiently, at most 560 msecs running time is observed (not visible in this figure due to the Y-axis scale). Figs. 4(a) and 4(b) show that OSS_L is preferable when time horizon is large.

5. REAL IMPLEMENTATION AND EXPERIMENTS

5.1 SmartTransfer on Android

SmartTransfer can be implemented entirely as a user-space library. However we choose to integrate it into Android framework, so all Android applications can readily use

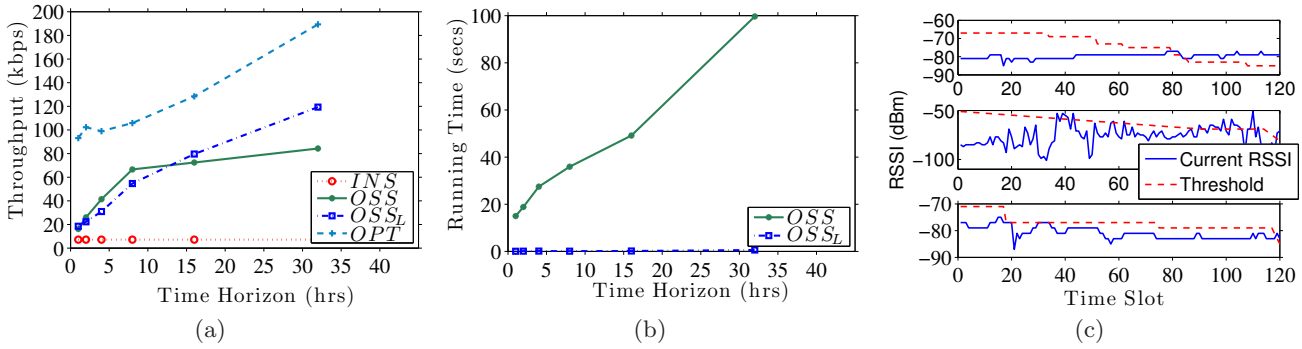

Figure 4: (a) Implications of the time horizon on throughput, (b) OSS does not scale to longer time horizon, and (c) sample scheduling decisions from the real experiments.

our content transfer service without reinventing the wheel. We have patched the Android 2.2 source tree of the Android Open Source Project (AOSP) with the proposed SmartTransfer framework. The SmartTransfer framework is realized as a service on Android. As a proof-of-concept, our current implementation supports the OSS_L algorithm; we are integrating the OSS algorithm into it. We collect user profiles in the background with a time slot of 30 seconds. The user profiles, up to 60 days, are used to generate the threshold table whenever the phone is being charged. Once there exists a pending content transfer request, we use the pre-computed threshold table to pick the best time for data transfer.

5.2 Application: VideoBlogger

We have implemented a sample Android application, called VideoBlogger, which emulates a smartphone user who regularly records videos and uploads them to a web site using the SmartTransfer API over the 3G network. The VideoBlogger generates realistic video uploads as follows. We develop a crawler to retrieve the metadata of YouTube videos. We use the crawler to get the sizes of 500 random videos, and built an empirical Cumulative Distribution Function (CDF). The VideoBlogger periodically generates video uploads with the sizes following this CDF, and employs a Poisson process with a mean of 10 minutes to determine the inter-arrival time of video uploads. Poisson process is often used to describe nature events in the real world. After each video update, the VideoBlogger sends the measurement results, including the start/end time, video size, and network throughput, to a backend ftp server. We run both the web and ftp servers on a Windows PC connected to the Internet over a Gigabit Ethernet link.

The VideoBlogger is implemented for large-scale experiments. It is implemented as a background service on Android, and has been deployed to experimental subjects over the Android Market since late February 2012. The recurring video uploads allow us to collect statistically-meaningful results without any subject intervention. We name all the log files using an one-way hash function of the mobile device IDs (IMEI numbers) to maintain subjects' privacy. We deployed VideoBlogger to subjects of diverse occupations, age, sex, and even in different time-zone. By choosing heterogeneous subjects, we show that the SmartTransfer automatically adapts to different users.

5.3 Preliminary Experimental Results

Due to the time and space limitations, we only report preliminary results in this section. Fig. 4(c) shows sample transfers from three subjects chosen from a total of 12 subjects. The solid lines show the RSSI values measured during the experiment, and the dashed lines show the thresholds generated by OSS_L. The first time RSSI falls below the current threshold, OSS_L will transfer the content. Using the experiments, we demonstrate the practicality of the proposed solution.

6. RELATED WORK

The idea of leveraging delay tolerant content transfer has been studied in the literature. For example, Hao et al. [8] proposed to selectively postpone less critical user-generated multimedia uploads to save energy on mobile devices. Their smartphone application first uploads the geographical metadata of each video at the capture time, and only uploads the video itself when there are explicit interests from other users. Their work in [8] is complementary to ours, as they consider on-demand user requests, while we concentrate on content transfer scheduling. In [17], the authors proposed to add drop zones for efficiently offloading upload traffic. In [4], the authors used WiFi network whenever possible to offload data from 3G connections, which is achieved by analyzing recent WiFi availability to predict the future availability. Schulman et al. [15] used location service to derive the user paths, which are then leveraged to predict future network condition. In comparison to [17], our solution does not require additional network infrastructure. On the other hand, in [4, 15], short history (e.g., last several measurements in a few minutes) is used to predict future network condition, which is valid for a short time scale, say minutes. In comparison, we have used much larger time-scale history profile, which enables us to provide longer time scale prediction, say in the time horizon of one hour. Because of the large time scale, we cannot use simple average to estimate future time condition, this "non-stationarity" introduces technical challenges which are addressed in this paper.

A different approach of capitalizing delay tolerant content transfers is to batch several requests together. Balasubramanian et al. [5] proposed a heuristic batching algorithm, based on the measurements of tail energy in the various networks. Kononen and Paakkonen [10] tackled the tail behavior of networks using a timer alignment technique. Qian

et al. [12] proposed to optimize the UMTS radio resource management state machine in order to balance energy saving and performance. We are working on including batching into the proposed opportunistic scheduling algorithms.

7. CONCLUSION AND FUTURE WORK

We propose SmartTransfer framework and two scheduling algorithms to intelligently schedule delay tolerant data to more favorable network conditions in order to potentially reduce network resource consumption, alleviate network congestion, and improve battery lifetime. In this preliminary work, we show that user network profile is an indispensable component to accurately predict future network condition, which is crucial for efficiently leveraging delay tolerance to save network resource. Our proposed solution is simple yet effective, and can be implemented on mobile platforms. We have discussed in details the implementation of SmartTransfer, as well as its practical applications. Using both real traces and experimental study, we have demonstrated the desirability of the proposed SmartTransfer.

We are extending the current study in several directions. First, we will consider the scenario where multiple transmissions with various sizes. Some of the transmissions could be combined/batched. The main benefit of batching is to reduce overhead induced by network setup, link setup, and tail effect of network interfaces. Second, we would like to further investigate machine learning technique to achieve a good balance between model complexity and profile data availability. We would like to also study the benefit of non-parametric learning. Furthermore, our proposed solution, in principle, can be used to alleviate network congestion, i.e., using the congestion level as the cost. The main challenge in studying network congestion alleviation is to achieve a good estimate of network congestion without imposing heavy probing overhead on both the device and the network.

8. ACKNOWLEDGEMENTS

This work is supported by HTC Magic Labs. C. Hsu and T. Lin are partially supported by National Science Council (NSC) of Taiwan (#100-2218-E-007-015-MY2).

9. REFERENCES

[1] Cisco visual networking index: Forecast and methodology, 2010–2015. `http://www.cisco.com/en/US/solutions/collateral/ns341/ns525/ns537/ns705/ns827/white_paper_c11-481360.pdf`, 2011.

[2] Why Verizon dropped its unlimited data plan (and what you can do about it). `http://moneyland.time.com/2011/06/23/why-verizon-dropped-its-unlimited-data-plan/`, 2011.

[3] M. Albanese, A. d'Acierno, V. Moscato, F. Persia, and A. Picariello. A ranking method for multimedia recommenders. In *Proc. of ACM International Conference on Image and Video Retrieval (CIVR'10)*, pages 311–318, Xian, China, 2010.

[4] A. Balasubramanian, R. Mahajan, and A. Venkataramani. Augmenting mobile 3G using WiFi. In *Proc. of ACM International Conference on Mobile Systems, Applications, and Services (MobiSys'10)*, pages 209–222, San Francisco, CA, 2010.

[5] N. Balasubramanian, A. Balasubramanian, and A. Venkataramani. Energy consumption in mobile phones: A measurement study and implications for network applications. In *Proc. of ACM SIGCOMM Conference on Internet Measurement (IMC'09)*, pages 280–293, Chicago, IL, 2009.

[6] J. P. Gilbert and F. Mosteller. Recognizing the Maximum of a Sequence. *Journal of the American Statistical Association*, 61(313):35–73, 1966.

[7] M. C. Gonzalez, C. A. Hidalgo, and A.-L. Barabasi. Understanding individual human mobility patterns. *Nature*, 453(7196):779–782, 2008.

[8] J. Hao, S. Kim, S. Ay, and R. Zimmermann. Energy-efficient mobile video management using smartphones. In *Proc. of ACM Conference on Multimedia Systems (MMSys'11)*, pages 11–22, San Jose, CA, 2011.

[9] C. Joe-Wong, S. Ha, and M. Chiang. Time-dependent broadband pricing: Feasibility and benefits. In *Proc. of IEEE International Conference on Distributed Computing Systems (ICDCS'11)*, pages 288–298, Minneapolis, MN, 2011.

[10] V. KoÌĹnoÌĹnen and P. Paakkonen. Optimizing power consumption of always-on applications based on timer alignment. In *Proc. of International Conference on Communication Systems and Networks (COMSNETS'11)*, pages 1–8, Bangalore, India, 2011.

[11] S. Nirjon, A. Nicoara, C. Hsu, J. Singh, and J. Stankovic. MultiNets: Policy oriented real-time switching of wireless interfaces on mobile devices. In *Proc. of IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS'12)*, Beijing, China, 2012.

[12] F. Qian, Z. Wang, A. Gerber, and Z. Mao. Characterizing radio resource allocation for 3G networks. pages 137–150, 2010.

[13] M. Ra, J. Paek, A. Sharma, R. Govindan, M. Krieger, and M. Neely. Energy-delay tradeoffs in smartphone applications. In *Proc. of ACM International Conference on Mobile Systems, Applications, and Services (MobiSys'10)*, pages 255–270, San Francisco, CA, 2010.

[14] E. Samuel-Cahn. Optimal Stopping With Random Horizon With Application to the Full-Information Best-Choice Problem With Random Freeze. *Journal of the American Statistical Association*, 91(433):357–364, 1996.

[15] A. Schulman, V. Navda, R. Ramjee, N. Spring, P. Deshpande, C. Grunewald, K. Jain, and V. Padmanabhan. Bartendr: A practical approach to energy-aware cellular data scheduling. In *Proc. of ACM Annual International Conference on Mobile Computing and Networking (MobiCom'10)*, pages 85–96, Chicago, IL, 2010.

[16] C. Song, Z. Qu, N. Blumm, and A.-L. BarabÃąsi. Limits of predictability in human mobility. *Science*, 327(5968):1018–1021, 2010.

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