Cut the Tail: Mobile Energy Saving Using Radio Tail Prediction

Zhe Zhao
Department of EECS
University of Michigan
zhezhao@umich.edu

Jiachen Zhao
Department of EECS
University of Michigan
zhaojiac@umich.edu

ABSTRACT

There are growing number of people using mobile computing devices such as Smart phones and tablets, to run different types of network applications. The cellular radio interface for communication in those network applications can cause significant energy drains. This is mainly because of the RaidoTails when the cellular radio remains in high energy state after communication has ended. To solve this problem, many existing works have discussed to employ Fast Dormancy, which is a feature that can force client radio to quickly go into a low energy state. The key problem in this line of research is to predict the proper time to invoke Fast Dormancy. A proper time means that when we invoke Fast Dormancy we would save as much idle time in high energy state as possible for radio cellular and also be able to avoid high overhead caused by frequent state transitions.

There are some existing works studying this problem and use data mining algorithms such as decision tree to predict the right time of End of Session(EOS) event in packet transmission. However, they only focus on applications running in background and cannot learn a general model for different types of network applications with all possible user actions. Thus our aim is to design a simple and general algorithm that can predict the proper time for Fast Dormancy in any applications with all possible user actions. We propose our effective and efficient algorithm to cut the RadioTail in this paper. We first analyze the characteristics of existing work. Then we propose a framework to reduce the dimensionality of the features used in prediction. After that, we apply the reduced features in different prediction algorithms. Lastly, we evaluate our proposed method on a data set tracked from Android OS in 18 days. Results show that our proposed algorithm can achieve 84% accuracy, which is more than 10% higher than the baseline.

Categories and Subject Descriptors

C.2.1 [**Computer Communications Networks**]: Network Architecture and Design – Wireless Communication; C.4 [**Performance of Systems**]: Design Studies, Performance Attributes

General Terms

Algorithms, Design, Measurement, Experimentation, Performance

Keywords

Fast Dormancy, EOS Prediction, Clustering

1. INTRODUCTION

Nowadays, mobile computing devices such as Smartphones tablets and laptops are playing an increasingly indispensable role in our daily life. And the most dreaded thing for every smartphone user is the battery running out of juice. Unfortunately, while smartphones are becoming ever more powerful and power-hungry with faster CPUs, more available RAM and larger bandwidth, the battery capacity simply cannot keep up. As shown in Figure 1[1], from the year 1990 to 2002, CPU speed, RAM size and Disk Capacity have grown for more than 100-fold. At the same time, battery capacity has merely doubled. Though the graph is outdated, the truth has remained the same — Battery capacity increase simply does not follow the Moore's Law like CPU frequency does. It used to be that you could charge your phone once and use it for days. Now, for heavy users, smartphone battery sometimes cannot even last for a single day. The need for energy saving is becoming more pressing with each generation of new smartphones.

Naturally, one of the major energy-consuming units in a Smartphone is the communication module. And as it turns out, it is also one of the components of a Smartphone where energy waste is severe. A dominant energy waste source is the RadioTail which is the period of time when radio remains in high-power state after a communication session has ended, as shown in Figure 2. RadioTail was originally intended to prevent radio from switching states too frequently and draining energy too quickly. However, it now has become a major energy drain itself as the radio is simply consuming power while doing nothing during this period of time. The situation is particularly bad for short communication bursts, which are very common in Smartphones. In these situations, up to 60% of all energy consumed in communication could be wasted in RadioTails [2]. Trying to save the energy

[1]Introduction to Vibration Energy Harvesting, NiPS Energy Harvesting Summer School August 1-5, 2011

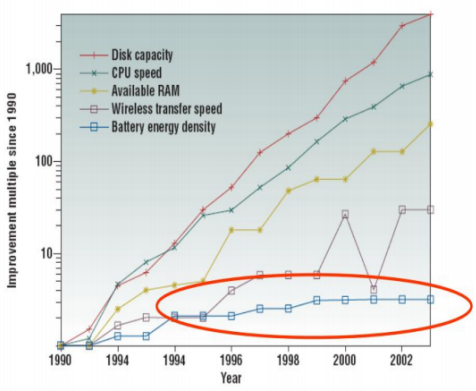


Figure 1: Improvement of Battery Energy Density

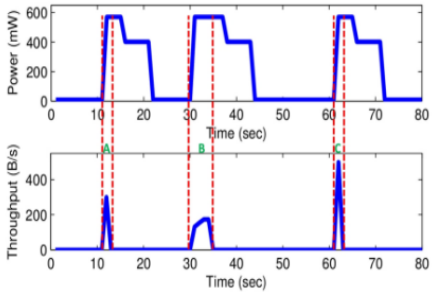


Figure 2: Energy Cost by Radio Tail[9]

wasted in Radio Tails, earlier works focused on measuring and modeling the Radio Tail[3, 5] in order to save the energy by tail sharing[17] or scheduling[11]. These works aimed at reducing or making use of the energy cost by Radio Tails.

Instead of studying how to save energy cost by Radio Tails, a popular technology called Fast Dormancy directly cuts the tail. This technology was introduced to 3G Universal Mobile Telecommunications System(UMTS) to save the current consumption of the device by switching between different mobile device activities states on the air interface. It can force the radio to quickly transition from high energy state(DCH) to a low energy state (Idle or PCH), which directly eliminates the radio tail. Apparently invoking Fast Dormancy right after transferring packet can save energy cost by Radio Tails. However, if we invoke Fast Dormancy frequently, we will suffer from high overhead casued by frequent state transitions. Therefore, the key problem of using Fast Dormancy is to decide a proper time to switch radio cellular state from DCH to PCH or Idle. Traditional timer based approach[9, 13, 16], which set a fixed inactive time such as 5s or 10s before invoking Fast Dormancy, is ineffective because it is inaccurate for different types of applications and it wastes a fixed amount of time waiting in high-energy state whenever it invokes Fast Dormancy. A recent paper [1] proposed Radio Jockey, which applied data mining algorithm to mine features of different applications and train a classification algorithm for each background application to predict the End of Session(EOS) event for packet transmission which is used as the optimal time to invoke Fast Dormancy. However, this approach trains different models for different applications, which is not general enough since there are a huge number of network applications and training one model for each one of them is clearly impractical. Meanwhile, to the best of our knowledge, suffering from the low accuracy on predicting the best time for Fast Dormancy, no approaches or implementations studied applications with user actions, i.e., applications not running in background. Because there are more packet transmission and radio cellular usage in applications with user actions, invoking Fast Dormancy in background applications only cannot eliminate energy waste effectively.

Based on the shortcomings of existing approaches, the goal of our work is two-fold: (1) A general algorithm that can predict EOS for invoking Fast Dormancy in different applications, with just one general training process. This means our proposed method can predict the proper time to invoke Fast Dormancy by predicting the EOS event for different types of applications. We don't need to train different models for multiple applications multiple times. Our proposed algorithm only needs one training process using the examples from any applications. (2) An algorithm with high accuracy and robustness for both applications running in background and with user activities. Our algorithm needs to be accurate enough for any application, not only for those running in background, but also for those with user actions. Therefore we can cut the Radio Tails in all situations.

Aiming at a general EOS prediction algorithm for any types of application, we propose our algorithm in this paper. Our algorithm mines the correlation between function call sequence of applications and the EOS event of packet transmission. We try to use function call sequence as the indicative feature of EOS and learn a classifier to detect whether any given function call sequence leads to EOS or not. Our work consists of the following three steps:

- We first analyze the characteristics of function call sequence and study how it overlaps in different applications. After the analysis, we find that most function calls appear only in one application, which makes it ineffective to use the function call as the feature of classification algorithm directly.

- We propose a feature selection framework to reduce the dimensionality and group the function calls in different dimensions or clusters. We intend for the new features selected from the function calls to be representative and overlap in most applications, so that they can be further used in general classification algorithms.

- We apply two types of selected features generated from the previous step to different classification algorithms, by which we improve the overall accuracy of predicting EOS on different applications.

To the best of our knowledge, this is the first work discussing the possibility of an effective and efficient general prediction algorithm for the optimal time of invoking Fast Dormancy in any type of applications, i.e., background applications and applications with user activities. The contribution of this work can be listed as follows:

- We conduct an extensive analysis on the characteristics of using function call sequence as feature to predict

EOS event for Fast Dormancy, including the distribution of function calls over different applications, the predictive power of classification algorithms on background applications and applications with user activity.

- We propose a EOS event prediction algorithm for Fast Dormancy that is a general model which can predict EOS event for different applications both running in background and with user actions. Based on the analysis, our algorithm introduces a feature selection framework that makes use of matrix factorization and clustering methods.
- We evaluate our proposed algorithm on data set collected from Android OS emulator in three weeks, where we observe that our proposed algorithm can achieve 0.83 accuracy in general EOS prediction and 0.74 accuracy in predicting EOS event on applications never seen before in execution.

The rest of our paper is organized as follows. We discuss related work on saving energy cost for Radio Tails and using Fast Dormancy to cut the tail in section 2. In section 3, we introduce preliminaries on Fast Dormancy which is a technique that can cut the radio tail directly. In section 4, we discuss the analysis of using function calls to predict EOS event for Fast Dormancy by studying the data we gathered and replicating representative existed works. Then we introduce our proposed general EOS prediction algorithm for Fast Dormancy in section 5. We evaluate and discuss the performance of our algorithm in section 6. Section 7 concludes the paper and talks about future works. At last, as a course project for EECS589, we summarize the project in section 8.

2. RELATED WORK

Efforts to reduce RadioTail energy waste generally break down to two categories: (1) Saving the Radio Tail Energy Cost without invoking Fast Dormancy, which includes approaches that focus on modeling and measuring the RadioTail to either reduce the energy cost by shaping traffic bursts or salvage the RadioTail for pre-fetch or delayed communication. And (2) Cut Radio Tail Using Fast Dormancy, which includes approaches based on fixed timers or utilize EOS prediction algorithm to quickly cuts the tail.

2.1 Saving Energy Cost from Radio Tail

2.1.1 Radio tail measurement and modeling

A number of papers have been devoted to measuring and modeling the RadioTail. A couple of them [3, 12] make active measurements of parameters of several major 3G network and observe significant difference of parameter settings which affect RadioTails among the different networks. However, using smartphone traffic trace collected from a large number of users, Falaki [5]proposes that up to 95% of all packets could arrive within 4.5 seconds.

2.1.2 Tail Sharing

This approach focuses on salvaging the Radio Tails for useful work. TailTheft [10] utilizes the Radio Tails to transmit pre-fetch data for pending transmission sessions or delay tolerant packets from previous sessions. Similar papers include Tailender [2] and TracBackfilling [8]. Bartendr [17] further considers signal strengths when doing the pre-fetches. However, this approach has the disadvantage of trading delay performance and packet-waste for energy savings, a tradeoff that we wish to avoid.

2.1.3 Smart scheduling to generate desirable bursts

Another way to minimizes the energy waste is to shape the RadioTails to improve energy efficiency. Looga [11] delays short communication bursts in order to combine them to form a long transmission burst which eliminates the Radio Tails that would have been generated by the short bursts. Like Tail Sharing, this approach leverages delay performance for energy efficiency.

2.2 Cut the Tail Using Fast Dormancy

2.2.1 Timer based FastDormancy

Fast Dormancy is a technology that enables smartphone handset to initiate quick radio state transition. The most intuitive and straightforward way to use Fast Dormancy is to set a fixed inactivity timer for invoking Fast Dormancy that maximize the damping effect of Radio Tail on radio state transitions while still minimizing energy cost. Several papers have devoted to finding the best timer period [16]. Other papers have proposed the adoption of dynamically choosing fixed timer using network traffic characteristics [9, 13, 20]. As stated earlier the timer approach is ineffective because the time Radio spends on waiting for timer expiration to invoke FastDormancy still constitutes energy waste and despite the effort, the great variation of user events and network traffic often makes it impossible to generate an optimal fixed timer value.

2.2.2 FastDormancy with EOS prediction

An efficient employment of FastDormancy is to use accurate EOS(End of Session) prediction for determining the best opportunities of invoking FastDormancy. TOP [14] attempts to achieve this by having applications inform the network when their transmission session ends. Network could then use the information to determine when to invoke Fast Dormancy. While accurate, it requires modifications to existing applications which is not feasible. Another approach for determining EOS, proposed by RadioJockey [1] mines program execution traces to make the prediction. Execution traces are readily available as most smartphone development kits provide method profiling tools. They also provide potentially highly characteristic patterns that could be associated with EOS as applications tend to execute a certain set of functions such as freeing buffers and closing communication sockets nearing the end of a transmission session. RadioJockey uses application execution traces to train individual decision trees for each application offline which are used online for EOS prediction. Though effective to some extent, this algorithm is simplistic and rigid, requiring a single tree for each application without regard for sequential ordering.

Instead of considering to save energy cost by Radio Tail, in our work, we consider directly cutting the tail by invoking Fast Dormancy, which is more straightforward and efficient. Compared with existing works in Fast Dormancy, our proposed approach can detect EOS from any types of application both from background and with user actions,

using just one training process for all applications. These advantages make our approach more efficient and effectiveness than those existing works in invoking Fast Dormancy at the proper time.

3. FAST DORMANCY

Fast Dormancy is a technique employed by User Equipment (UE) such as smartphones in UMTS standard to quickly turn radio from Cell_DCH state(the high energy consuming transmission state) to Cell_IDLE state without the need of waiting for the radio to first move into Cell_FACH state(an intermediate state) and then Cell_IDLE state(the lower power state that the radio maintains no connection with the network)[6]. This is achieved by sending the RRC(Radio Resource Control) Signalling Connection Release Indication which is originally intended for situations where the UE has powered off or there is a NAS level error. However, this could cause network to lose control of the RRC and incurs large signaling overhead when the UE switches back from Cell_IDLE to Cell_DCH causing the communication system capacity to quickly become limited by the amount of signalling rathern than bandwidth it could accommodate.To remedy this, 3GPP standardized a Fast Dormancy feature in Release 8 which considers system aspect and allows network to retain control of the UE. This is the so-called Release 8 Fast Dormancy which is supported by both Android OS and Iphone OS. It requires the UE to include a cause value indicating end of session in the RRC Signalling Connection Release Indication. This is used by the network to detect that a UE has no more packet to send and could move it to Cell_PCH which is a low-power state that has a much lower latency and signaling overhead for resuming communication than Cell_IDLE. Though power saving, Fast Dormancy is not a panacea as the radio state transition still consumes energy and signaling overhead. Hence it's necessary for a UE to accurately determine whether a EOS has been reached and Fast Dormancy is to be invoked. Currently, an inactivity timer of 3-5s is usually used for this purpose.

4. ANALYSIS ON EOS PREDICTION USING FUNCTION CALLS

In this section, We will first briefly introduce an existing work[1] that trains decision tree on function call sequence of each application to predict EOS session for Fast Dormancy. Then to analyze the characteristics of the existing work, we will show how we generate data set from different types of network applications by trace gathering in Android emulator. Then we study the distribution of function calls in different applications and discuss the possibility of making use of this type of feature to extend existing algorithm to a general prediction algorithm for different types of applications. We then study the performance of decision tree for both background and active applications in comparison with a state-of-the-art classification algorithm, Support Vector Machine(SVM)[4]. We also try to directly use function call sequences of all applications. We have to train one decision tree or SVM to test the performance of general prediction accuracy.

4.1 RadioJockey: Decision Tree to Predict EOS

In [1], the authors proposed RadioJockey, which applies decision tree trained using function calls of each applications. Decision Tree is a rule based classification algorithm that can use rules of features learned from training data set to classify new data point. The framework of their work is shown in Figure 3. Their task is to detect whether a given function call list indicates an EOS event that can last for a predefined length of time. By predicting the correct EOS event, the mobile device can force the application directly change to low power state using Fast Dormancy without waiting for a fixed period of time which would be at least of the length of the predefined fixed timer value. For each application, they got the function call lists near the EOS event, and they used the function call list of function calls that happen within a short period of time before EOS as positive examples, and used the function call lists of function calls that happen within a short period after EOS as negative examples. Then they trained a decision tree for the application and used the decision tree to detect new function call lists of this application so that whenever a function call lists is classified as EOS, the mobile device can invoke Fast Dormancy in this application to eliminate Radio Tail cost.

The shortcomings of this work are obvious. First, it needs separate training process for each different application, which requires much effort for every new application. The decision tree used in this work learns very strict rules at Function Call level, which is either complicated or suffers from loss of generality, since some function calls may be calling similar functions but with different name. Lastly, this work treats the function calls as a function call list but not a sequence of function calls, which ignores the meaningful sequential pattern. In the next several paragraphs, we will evaluate this work and compare it with Support Vector Machine, which is an effective algorithm widely used in general classification tasks.

4.2 Data Set Description

4.2.1 Data Gathering

Based on our project design, we collected two types of data: program execution trace and network traffic trace. The data gathering platform is an Android emulator running Android platform 4.1.2. The program execution trace is gathered using the Android function profiler[7] that's included in the Android SDK. The trace includes three categories of applications: IM, Web Browsing and Web Service Portal(such as Google Reader), which encompasses the majority of the web applications used on smart phones. Network trace is gathered using the AT&T ARO[15] which is a cross-layer mobile application resource profiling tool. It captures all network packets arriving and departing the emulator and maps each packet to its corresponding application, enabling us to correlate the two types of data.

4.2.2 Data Set Generated

Using the data gathering method, we tracked a list of applications shown in Figure 4. Based on the 18 days of continuous tracking, we obtain 21 hours of data set where there is internet access. We use the same method as in [1] to extract EOS event after which there is a predefined period (5s in our experiment) of inactive time. We obtain 1343 EOS events that are suitable for invoking Fast Dormancy to eliminate Radio Tails. For these 1343 EOS events, we select function calls within 2s before the events as positive examples and

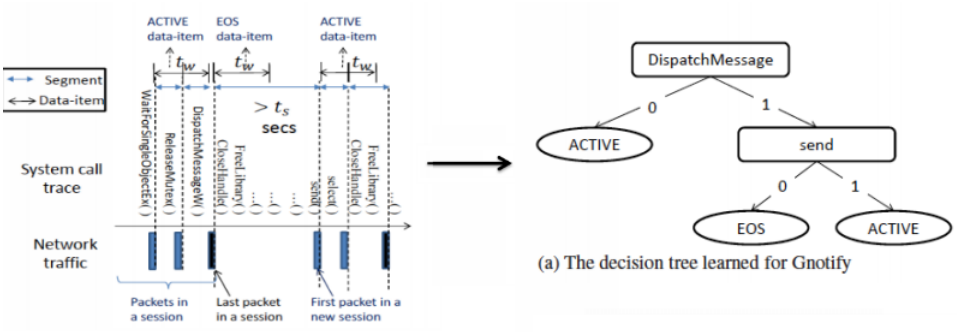


Figure 3: Learning Decision Tree from Function Call Sequence[1]

Application	Description
QQ	An IM application similar to AIM or Yahoo Messenger
Native Browser	The native Android browser
OperMini	The Android version of Opera Browser
QQBrowser	A mobile browser developed by Tencent
Tmall	A online shopping portal similar to Amazon
Tianya	A portal application to a popular Chinese online forum Tianya.
GoogleReader	A portal application for reading your GoogleReader RSS feeds.
Weibo	A Twitter-like application developed by Sina
Baidunews	A news reader application similar to Google News.

Figure 4: Applications Tracked in this Paper

those wthin 2s after the events as negative examples. We separate our tracked data set into 14 different cases. The first four cases are function calls and EOS event extracted from applications running in background, while the latter 10 cases are from applications with user activities. In the data gathering, we perform different types of user activity on different applications, such as surfing the internet, post micro-blogs or chatting using IM, etc..

4.3 Analysis on Function Call Distribution

Using function calls from application as feature is a simple and straightforward way since packet transmission should always be correlated with function calls. Figure 5 shows the distribution of function calls in the 18 cases we tracked. We can see that the most of the function calls just appear in only one case/application. Therefore although function call can be indicative feature in predicting EOS session, it is necessary to train different Decision Trees for different applications since most function call appear only in one application. Thus, we cannot use function call directly as feature in a general classification algorithm that aims at predicting EOS for any application.

4.4 Analysis on Different Applications

We implement the Decision Tree on our data set and then show its performance on different applications we tracked.Then we compare the performance with SVM. SVM is a general classification algorithm that aims at maximizing the margin between two sets of points with different labels in feature space. Compared with decision tree, it is not rule based and directly aims at a clear classification boundary. SVM is tested to be the most effective classifier in majority of tasks. Detailed information of SVM can be found in [4]. In all experiments in this paper, we use average precision in classification as the metric in evaluation, i.e., $Pr = \frac{1}{n}\sum_{t=1}^{n} \|sgn(L'_t - L_t)\|$ where L_t and L'_t are the original label and predicted label respectively, sgn is the sign function and n is the size of test data set.

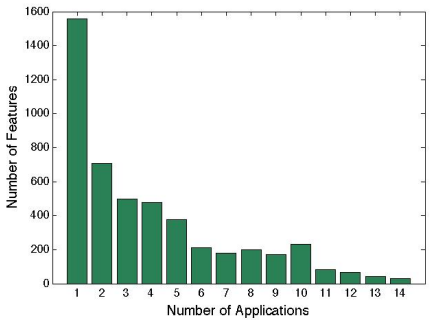
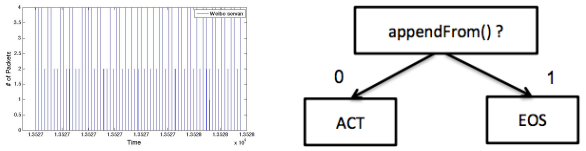


Figure 5: Function Call Distributions among Different Applications



(a) Example of Weibo Servant's Packet Flow (b) Decision Tree Learned from Weibo Servant

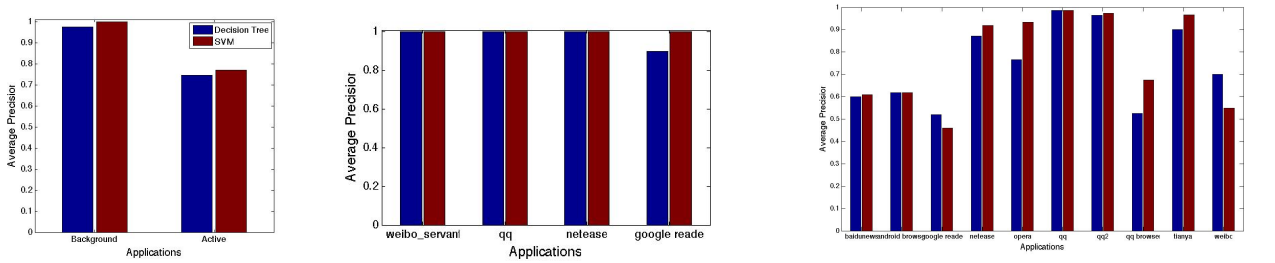
Figure 6: Example of Decision Tree for Background Application

4.4.1 *Training for Each Application*

We first replicate the existing work, where a decision tree is trained for each unique application. We also train SVM using each application to compare the average precision with Decision Tree(DT). We run the training and testing on the 14 different types of applications/cases for the two algorithms. We use 10-fold cross validation to calculate the average precision. The average accuracy on two types of applications, i.e., background applications and applications with user actions is shown in Figure 7(a). We can see the results for both algorithm on background applications is very high, while on active applications is relatively low.

Background Applications.

Figure 7(b) shows the results of DT and SVM on four background applications. We can see that the precision for



(a) Average Precision on All Applications (b) Average Precision on Background Applications (c) Average Precision on Active Applications

Figure 7: Predicting EOS using Decision Tree trained from Different Applications

SVM is 1 and for DT is very close to 1. This very high precision is because background applications usually have simple patterns of accessing Internet and packet transmission. Figure 6 shows an example decision tree learned from one of the four background applications, we can see that for this background application, the decision tree structure is as simple as just one rule. Background applications like this example act in very simple way, thus using decision tree or SVM even if with limited number of training examples can achieve a high average accuracy.

Application with User Actions.

Figure 7(c) shows the results of DT and SVM on 10 application with user actions. The average precision drop to 70%. For half of the applications, the average precision is around or less than 50%, which is similar as random guessing the EOS event. The results suggest that directly using Function call as feature to train classification algorithms such as Decision Tree or SVM on applications with user actions can not be effective.

4.4.2 Training for General Prediction

We further examine whether the existing algorithm can work well on general prediction, which means we don't train different model for each application, instead, we just train one model to deal with different applications.

Training from all Applications.

We train Decision Tree and SVM using function calls generated in 14 different types of applications. We randomly separate the examples into 10 groups and do a 10-fold cross validation. The left side of Figure 8 shows the result. We can see that the average precision of DT is below 0.7 and SVM is above 0.8. This is because rule based classification method may suffer from the diversity of function calls in different applications and the structure of the tree may become complicated. On the other side, because SVM is aiming at maximizing the margin of positive and negative data points, it can still find all possible function calls related to EOS event as features in feature space even if most function call appear only in one applications.

Training to Predict New Application.

We also want to test how the prediction algorithm will perform when they are used to predict EOS event from some application they never see before. For each application, we use model trained by examples from the other applications.

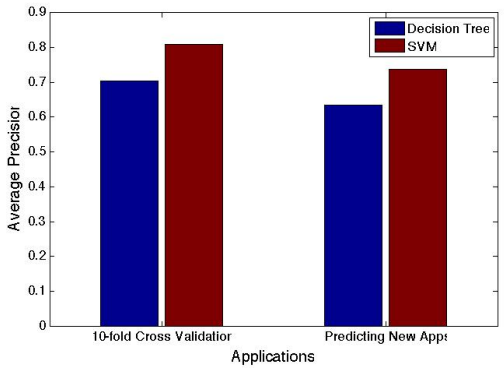


Figure 8: General Model for EOS Prediction

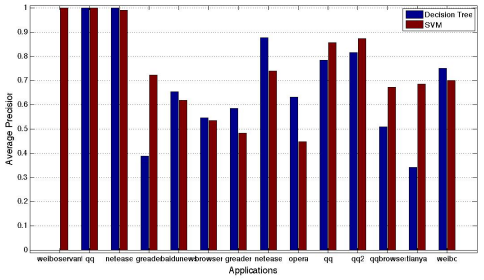


Figure 9: Predicting EOS for Unseen Applications

Then for each application, we can have the average precision of how accurate it would be if the model never see the examples of this application in training. We show the average results of the 14 applications in the right side of Figure 8 and the result of predicting each application in Figure 9. We can see that when predicting new applications, the average precision further drops about 10%. And for rule based Decision Tree algorithm, for some application, the strict rule learned always predict the opposite results so the average precision is 0. For both algorithms, it is easier to predict unseen background applications than to predict active applications.

5. OUR APPROACH

In last section, we analyze the method of using function call as feature for applications to predict EOS event using

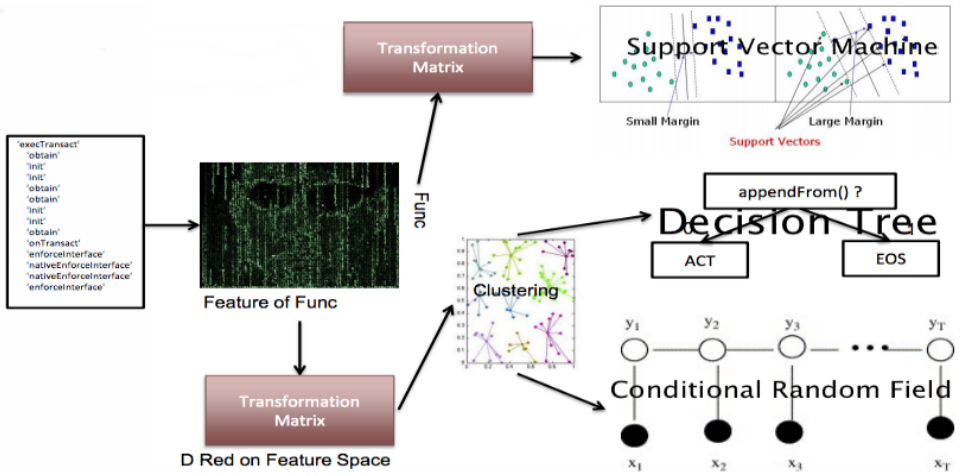


Figure 10: Framework of Our Approach

Decision Tree or SVM. We first train different models for different applications as in paper [1]. Then we train one model for all applications. We find that for background application, the mean average precision for predicting EOS event using models trained from all the applications or other applications is higher than the mean average precision for application with user actions using models trained from those applications. However, for Decision Tree, it is also possible to have 0 accuracy predicting unseen applications since the rule learned from other applications may be opposite.

Based on these observations, we propose our approach in this section, aiming at a general prediction algorithm which can achieve a higher performance for both background applications and applications with user actions. Our approach adopts a framework of feature selection that can produce a better feature for SVM, DT and other classification algorithms. Then we use the selected new features to train prediction algorithms using DT and SVM. We also try to improve the performance by considering the sequential pattern of function calls using probabilistic graphical model such as Conditional Random Field(CRF)[19]. In the rest of this section, we first briefly introduce the framework of our approach, then we discuss how to describe function calls using proper features. We propose two methods to transform function calls using the feature matrix. For SVM, we use the new feature generated from matrix factorization algorithms such as Principal Component Analysis(PCA) or Non-Negative Matrix Factorization(NMF). But for DT and CRF, we need to transform function calls into discrete features. So we adopt clustering algorithms to group function calls together. Finally, we design a linear-chain CRF to make use of the sequential pattern of function calls.

5.1 Framework

The framework of our approach is shown in Figure 10. Our aim is to achieve higher average precision for general EOS prediction. Therefore we develop a feature selection framework to select better features that are general in different types of applications. We first build feature matrix of function calls, i.e., for each unique function call, we use feature to represent it. After we have this feature matrix, we can either Map Function Call to latent space transforming each function call list to a low dimensionality feature vector, or group function calls into clusters transforming each function call list to a group list. The features selected by the first method can be used in SVM, and features selected by the second method can be used in DT or CRF.

5.2 Feature Representing Function Calls

If we want to select feature by mapping function calls to latent space or group them into different groups, we need to measure the similarity or distance of different function calls. During the data gathering process, we discover that we can obtain the library name of the function call. In our data set, there are 6208 unique function call names while there are only 2768 unique library names. Therefore we have this intuition of measuring the similarity of function calls based on their library names and the locality information, which means the similarity of function call can be measured by the libraries invoked by the function call and its adjacent function calls.

For each function call, we extract a list of libraries invoked by all the functions in a window with a predefined window size near the function call in time, where the the center of the window is the given function call. We count the frequencies of each pair of function call and library whose name shows in the window near the function call in the whole data set. Then we use the 2768 dimension feature vector to represent each function call, each dimension of which is the frequency of the occurrence of the library name in the function call window. Then the distance of two function calls measured in Euclidean distance can be interpreted as the difference of libraries invoked in the two function calls and the function calls adjacent to them.

5.3 Mapping Function Call to Latent Space

After building the feature matrix of function call, we then use matrix factorization[18] to the feature matrix to map the function call to latent space. The typical problem of matrix factorization can be written in the following equation:

$$M \approx F \times L^T, M \in R^{n \times d}, F \in R^{n \times k}, T \in R^{d \times k} \quad (1)$$

In our feature matrix M, n=6208 is the number of unique

function calls and d=2768 is the unique libraries. The k is the reduced latent space. if k is less than the rank of the matrix M, then $F \times L^T$ is a approximated result of the matrix M. Each latent dimension is represented as a linear combination of the original function call space as a row in the matrix F. Based on different objectives and constrains to achieve the approximation, different results can be learned. Principal Component Analysis(PCA) is a common statistic method that finds k principal components as the latent dimensions. In PCA, the objective shown below is to minimize the L2 norm of $M - F \times L^T$. PCA is widely used in matrix factorization, since its objective is simple and straightforward.

$$\min \|M - F \times L^T\|_2 \tag{2}$$

In our approach, we also use PCA to find the k PCs. The first k PCs are the most important latent dimensions since it can record most of the matrix information. In our problem, it is meaningless to have negative values in the transform matrix, since each entry of the transform matrix represents the importance of a function call to one latent dimension, where the least importance should be 0. So we use Non-negative matrix factorization, which adds a constrain to equation 1 shown below that each entry of the matrix F and L should be non-negative.

$$\min \|M - F \times L^T\|_2, s.t., F, L > 0 \tag{3}$$

After we have the matrix F and L, given a function call list in the positive or negative examples which is a n dimensional vector recording number of n different function calls, we use matrix F as transform matrix to transform the n-dimensional vector a to K-dimensional vector. Then each example of function calls is now represented by a k-dimensional vector. Since k is usually set to hundred level, the new features we generated can be used to train a general SVM classifier to predict EOS event. We will talk about the tuning of k, the number of latent dimension in our evaluation section.

5.4 Group Function Call to Clusters

The transformed latent vectors of function calls are vectors in continuos space, which can be directly used in SVM, but will not act well in DT or cannot be used in CRF, since in these models, the features need to be discrete as sets of items. Then here we present a clustering algorithm to group the function calls into different groups. And we transfer the function call list to group list which can be used in DT and CRF.

5.4.1 Dimensionality Reduction in Feature Space

After having the feature matrix M for function call that is a 6208×2768 matrix, our task is to group the 6208 function calls into tens or hundreds of groups. The problem of directly adopting any clustering algorithm on this matrix is that, the dimensionality of the feature space, i.e., 2768 is relatively high and the number of samples, i.e., 6208 function calls is relatively small. That means that we can either get too many clusters or too few clusters, neither of which can yield good performance. Therefore, we use PCA to reduce the dimensionality first. We reserve only top t PCs and transform the matrix to $6208 \times t$ entries. We will talk about the tuning of t, the reduced dimensionality for clustering, in our evaluation section.

5.4.2 Function Call Clustering

After we reduce the dimensionality of 2768 to t, we adopt K-means clustering algorithm to group the function calls into g groups. After that we have the mapping of which function call belong to which group. Then each example of EOS event represented by function calls can be transformed to be represented by the group lists, which can be more general in different applications. We use the group lists as features to train Decision Tree and try to use them in modeling the sequential pattern of function calls. We will discuss the tuning of g, the number of groups/clusters in our evaluation section.

5.5 Labeling with Conditional Random Field

We also try to make use of the sequential pattern of the function call list to detect EOS event. Since function calls are being invoked in an order, and such order can be highly indicative of EOS event. Since all proposed works treated function calls as a set of items and didn't model the sequential pattern, we are the first to discuss this possibility of making use of function call sequence. Because most function calls are localized and only occur in just one applications and changing orders between two functions doesn't usually matters in some occasions, we use the function call groups we generated by clustering algorithm in this part of the work.

Linear-Chain Conditional Random Field as shown in Figure 10 is a special form of Markov Random Field that is widely adopted in sequential pattern mining such as Part-of-Speech(POS) detection in Natural Language Processing[19]. Here we adopt this model in EOS event prediction for Fast Dormancy. Since there are only two labels, i.e., active and EOS, and EOS can only occur in the end of sequence, which will simplify the CRF to simple frequency matching, we add a pre-EOS label at the two groups before the last function. Then we use the CRF to predict whether the end of the sequence is EOS or not. Due to the limit of pages, we don't include the inference for EOS state in this paper. It is a similar and simplified version of Linear-Chain CRF in POS detection.

6. EVALUATION

In this section, we design and conduct extensive experiments on the data set we gathered using Android emulator. We first introduce the experiment setup, then discuss the tuning of the three parameters used in our approach, and at last show the overall performance and improvement.

6.1 Experiment Setup

Similar to section 4, we run our experiment on the data set tracked from Android OS emulator, which contains 1343 EOS event which will make the application's radio usage inactive for at least 5 seconds. We generate 1343 positive function call lists which are all the function calls in 2 seconds before the EOS events, and 1343 negative function call lists which are function calls in 2 seconds after the EOS events. We discuss how our proposed algorithm will improve on predicting EOS event of different types of applications with only one training process on all the applications. We use average precision in 10-fold cross-validation as the metric. We also

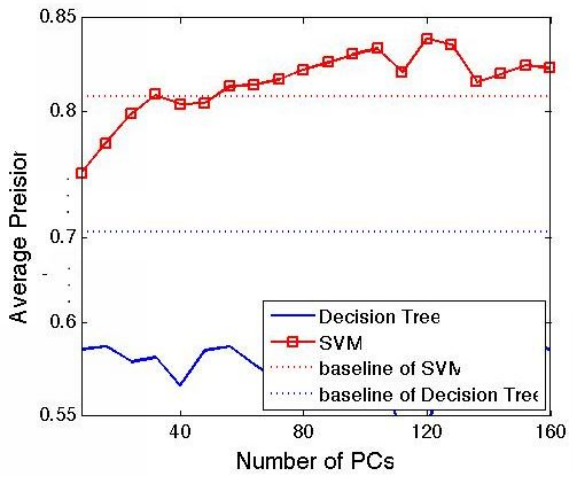


Figure 11: Tuning the Number of Principal Components

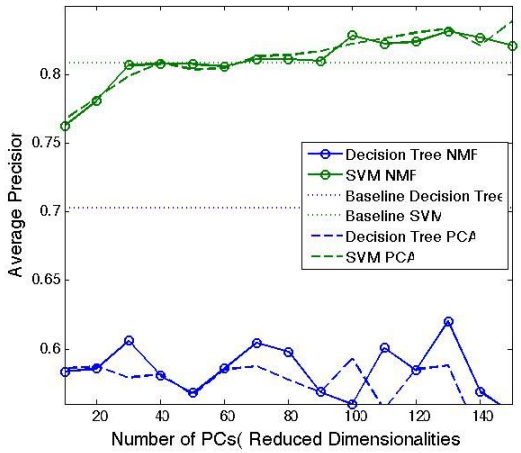


Figure 12: Tuning the Dimensionality of NMF

study the improvement on predicting EOS event of unseen applications.

6.2 Mapping Function Call to Latent Space

Here we show the experiments on tuning the parameter k, which is the dimensionality of latent space. Figure 11 shows the results of SVM and DT using the features of latent space transformed from function call feature matrix by PCA. We can see that DT doesn't work well and is even worse than the baseline which is DT before dimensionality reduction. However, SVM has a relatively significant improvement when we select a proper k, i.e., after k is larger than 40. Since the performance drops after k is larger than 120, which can be caused by introducing noise in the model. When k=120, the SVM algorithm acts the best, achieving average accuracy near 0.85 for general EOS event prediction.

Figure 12 shows the results of using NMF to reduce the dimensionality. The results is similar of using PCA shown in Figure 11. We can see that using NMF the performance improve faster when increasing the number of k.

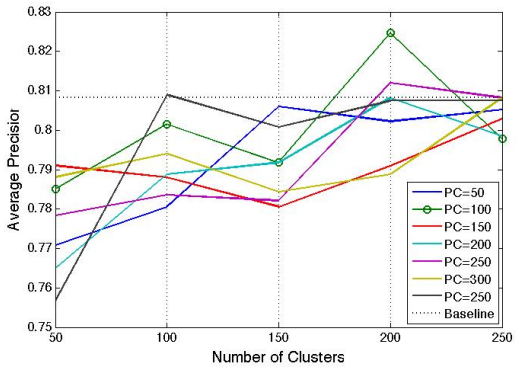


Figure 13: Tuning the Number of Clusters using SVM

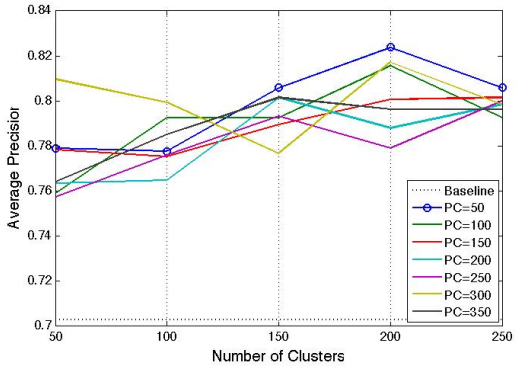


Figure 14: Tuning the Number of Clusters using Decision Tree

6.3 Group Function Call to Clusters

Here we show the experiment on tuning t, the dimensionality of latent space for clustering and g, the number of clusters. Figure 13 shows the result using the group list for EOS detection by SVM, and Figure 14 shows the result by Decision Tree. We can see that the improvement of SVM is not significant and in many of the settings of the parameters, SVM performs below the baseline, i.e., SVM directly using function call list. However, by using the function group list instead of function call list, the performance of Decision Tree improves more than 10%.

Figure 15 shows the distribution of function groups in different types of applications. We can see that compared to Figure 5, most of the function groups occur in all applications. It shows that our clustering framework significantly improves the generality of the features used in training classifiers.

6.4 Overall Performance

After tuning the parameters, we select the best parameter for SVM and Decision Tree and show the overall performance of EOS prediction for different types of applications in Figure 16. We can see that comparing to the existed work we replicated [1], simply using SVM will improve the average precision around 10%. The average precision of classifiers using our proposed feature selection framework has the best

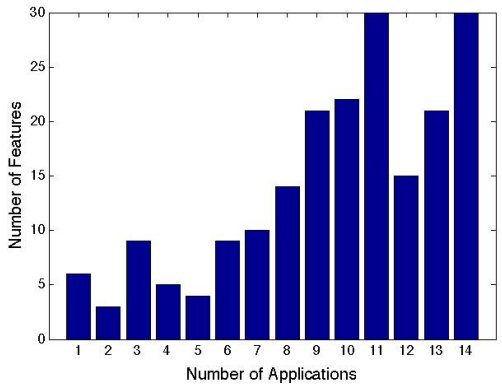


Figure 15: Cluster Distribution over Applications

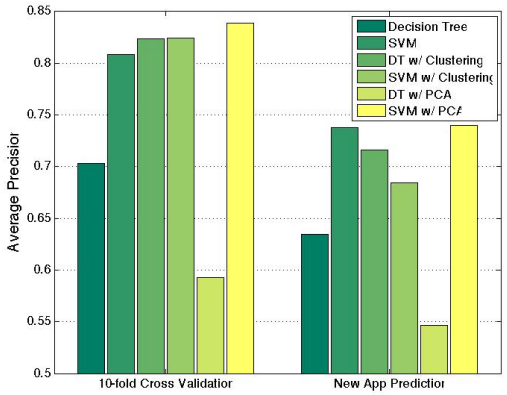


Figure 16: Performances of Proposed Algorithms Comparing with Baseline

performance, which yield a 14% improvement comparing to [1], which trains decision tree on function call list.

6.5 Performance of Linear Chain CRF

At the time of writing this report, the average precision for general EOS prediction using CRF is 69.2% and the average precision for CRF to predict EOS event on unseen applications is 61.64%. This is because the tuning of CRF's parameters, which includes the defining of potential functions and the settings of initialization, requires a large amount of time. Also, learning the CRF for each round of tuning is time consuming. Since the performance of POS detection in NLP using CRF can achieve precision higher than 90%, we believe tuning CRF on this similar task will eventually enable the model achieve better performance. We list this as one of the future works.

7. CONCLUSION & FUTURE WORK

In this paper, we study the problem of invoking Fast Dormancy to cut the Radio Tails and eliminate the tail energy cost. The key problem of this technique is to choose the proper time to invoke Fast Dormancy to force the radio usage of network application to low cost state. This is because Fast Dormancy can eliminate tail energy cost, but frequently invoking Fast Dormancy will have extra overhead due to frequent state transmission. Traditional timer based approach sets a timer of fixed length in inactive network access time which will waste the tail cost in the fixed inactive time. Some existing work use data mining algorithm such as decision tree to learn EOS event detection algorithm for each application to predict the best time of invoking fast dormancy. Based on our replication and analysis, such algorithm using function call list of applications can not be generalized without the need of learning model for each application. Meanwhile, because of the low precision in predicting the proper time for Fast Dormancy, Fast Dormancy now can only be used in background applications.

We aim at building an effective and efficient general classifier to detect EOS event for Fast Dormancy in different types of applications, i.e., not only applications running in background but also applications with various use actions. We propose a feature selection framework that use matrix factorization and clustering algorithm to reduce the dimensionality of function call space for the training of SVM, or group the function calls in clusters for the training of decision tree. We also make use of linear chain conditional random field to model the sequential pattern of function call group list. We run extensive experiments on the data we gathered from Android Emulator and results show that our proposed framework can achieve 0.84 average accuracy on EOS event prediction of different types of applications, and also a 0.74 accuracy on predicting EOS for unseen applications, which is 14% higher than baseline.

Future Work.

One of our future work as discussed in the evaluation section, is to tune CRF to make it perform better in EOS event prediction. Because the limited amount of time in finishing this project and the high time cost in learning CRF, currently the performance still needs improvement. But we believe that carefully designing the potential function and setting the initial value of weights properly can substantially improve the performance of CRF.

8. PROJECT SUMMARY

This is the project report for EECS589 Advanced Computer Network in University of Michigan, Ann Arbor. We approximately use 6 weeks to finish this project. It took us first one and a half week doing survey on related area, two weeks collecting the data and two weeks replicating, analyzing and developing of our algorithm. At last we use half a week to write this report. During the project, we find that utilizing and redesigning data mining technologies for computer network applications a very interesting area. And because of the large scale and big data, such area is full of potential to develop either a better computer network environment or data mining/ machine learning algorithms. Finally, we really appreciate the help from Junxian Huang, who is a PhD student from Prof. Morley Mao's research group, of providing us suggestions of choosing this project topic. Some of the figures from outside of this project used in this report are cited in the context, and we will change the figures into a better format in future work of this project.

9. REFERENCES

[1] P. K. Athivarapu, R. Bhagwan, S. Guha, V. Navda, R. Ramjee, D. Arora, V. N. Padmanabhan, and G. Varghese. Radiojockey: mining program execution to optimize cellular radio usage. In *Proceedings of the 18th annual international conference on Mobile computing and networking*, pages 101–112, 2012.

[2] N. Balasubramanian, A. Balasubramanian, and A. Venkataramani. Energy consumption in mobile phones: a measurement study and implications for network applications. In *Proceedings of the 9th ACM SIGCOMM conference on Internet measurement conference*, pages 280–293, 2009.

[3] A. Barbuzzi, F. Ricciato, and G. Boggia. Discovering parameter setting in 3g networks via active measurements. *Communications Letters, IEEE*, 12(10):730 –732, 2008.

[4] C. Cortes and V. Vapnik. Support-vector networks. *Machine Learning*, 20:273–297, 1995.

[5] H. Falaki, D. Lymberopoulos, R. Mahajan, S. Kandula, and D. Estrin. A first look at traffic on smartphones. In *Proceedings of the 10th ACM SIGCOMM conference on Internet measurement*, pages 281–287, 2010.

[6] http://bit.ly/Nc0US7. Behavior analysis of smartphones.

[7] http://developer.android.com/tools/help/traceview.htm. Traceview | android developers.

[8] H. A. Lagar-Cavilla, K. Joshi, A. Varshavsky, J. Bickford, and D. Parra. Traffic backfilling: subsidizing lunch for delay-tolerant applications in umts networks. *SIGOPS Oper. Syst. Rev.*, 45(3):77–81, 2012.

[9] F. Liers and A. Mitschele-Thiel. Umts data capacity improvements employing dynamic rrc timeouts. In *Personal, Indoor and Mobile Radio Communications, 2005. PIMRC 2005. IEEE 16th International Symposium on*, volume 4, pages 2186 –2190 Vol. 4, 2005.

[10] H. Liu, Y. Zhang, and Y. Zhou. Tailtheft: leveraging the wasted time for saving energy in cellular communications. In *Proceedings of the sixth international workshop on MobiArch*, pages 31–36, 2011.

[11] V. Looga, Y. Xiao, Z. Ou, and A. Yla-Jaaski. Exploiting traffic scheduling mechanisms to reduce transmission cost on mobile devices. In *Wireless Communications and Networking Conference (WCNC), 2012 IEEE*, pages 1766 –1770, 2012.

[12] P. Perala, A. Barbuzzi, G. Boggia, and K. Pentikousis. Theory and practice of rrc state transitions in umts networks. In *GLOBECOM Workshops, 2009 IEEE*, pages 1–6, 2009.

[13] I. Puustinen and J. Nurminen. The effect of unwanted internet traffic on cellular phone energy consumption. In *New Technologies, Mobility and Security (NTMS), 2011 4th IFIP International Conference on*, pages 1 –5, 2011.

[14] F. Qian, Z. Wang, A. Gerber, Z. Mao, S. Sen, and O. Spatscheck. Top: Tail optimization protocol for cellular radio resource allocation. In *Network Protocols (ICNP), 2010 18th IEEE International Conference on*, pages 285 –294, 2010.

[15] F. Qian, Z. Wang, A. Gerber, Z. Mao, S. Sen, and O. Spatscheck. Profiling resource usage for mobile applications: a cross-layer approach. In *Proceedings of the 9th international conference on Mobile systems, applications, and services*, pages 321–334, 2011.

[16] F. Qian, Z. Wang, A. Gerber, Z. M. Mao, S. Sen, and O. Spatscheck. Characterizing radio resource allocation for 3g networks. In *Proceedings of the 10th ACM SIGCOMM conference on Internet measurement*, pages 137–150, 2010.

[17] A. Schulman, V. Navda, R. Ramjee, N. Spring, P. Deshpande, C. Grunewald, K. Jain, and V. N. Padmanabhan. Bartendr: a practical approach to energy-aware cellular data scheduling. In *Proceedings of the sixteenth annual international conference on Mobile computing and networking*, pages 85–96, 2010.

[18] N. Srebro. *Learning with matrix factorizations*. PhD thesis, Cambridge, MA, USA, 2004. AAI0807530.

[19] C. Sutton and A. Mccallum. Introduction to conditional random fields for relational learning. 2006.

[20] J.-H. Yeh, J.-C. Chen, and C.-C. Lee. Comparative analysis of energy-saving techniques in 3gpp and 3gpp2 systems. *Vehicular Technology, IEEE Transactions on*, 58(1):432 –448, 2009.