Sensing Meets Mobile Social Networks: The Design, Implementation and Evaluation of the CenceMe Application

Emiliano Miluzzo[†], Nicholas D. Lane[†], Kristóf Fodor[†], Ronald Peterson[†], Hong Lu[†], Mirco Musolesi[†], Shane B. Eisenman[§], Xiao Zheng[†], Andrew T. Campbell[†]

[†]Computer Science, Dartmouth College, Hanover, NH 03755, USA

[§]Electrical Engineering, Columbia University, New York, NY 10027, USA

ABSTRACT

We present the design, implementation, evaluation, and user experiences of the CenceMe application, which represents the first system that combines the inference of the presence of individuals using off-the-shelf, sensor-enabled mobile phones with sharing of this information through social networking applications such as Facebook and MySpace. We discuss the system challenges for the development of software on the Nokia N95 mobile phone. We present the design and tradeoffs of split-level classification, whereby personal sensing presence (e.g., walking, in conversation, at the gym) is derived from classifiers which execute in part on the phones and in part on the backend servers to achieve scalable inference. We report performance measurements that characterize the computational requirements of the software and the energy consumption of the CenceMe phone client. We validate the system through a user study where twenty two people, including undergraduates, graduates and faculty, used CenceMe continuously over a three week period in a campus town. From this user study we learn how the system performs in a production environment and what uses people find for a personal sensing system.

Categories and Subject Descriptors: C.2.1 [Network Architecture and Design]: Wireless Communications; J.4 [Social and Behavioral Sciences]: Sociology.

General Terms: Design, Experimentation, Performance.

Keywords: Applications, Social Networks, Mobile Phones.

1. INTRODUCTION

One of the most common text messages people send each other today is "where r u?" followed by "what u doing?". With the advent of powerful and programmable mobile phones, most of which include a variety of sensing components (e.g., accelerometers, GPS, proximity sensors, microphone, camera, etc.) there is a new way to answer these questions. In essence, mobile phones can create mobile sensor networks capable of sensing information that is important to people, namely, where are people and what are they doing?

The sensing of people is driving a new application domain that goes beyond the sensor networks community's existing focus on environmental and infrastructure monitoring, where people are now the carriers of sensing devices, and the sources and consumers of sensed events. The expanding sensing capabilities of mobile phones (e.g., Nokia N95 and Apple iPhone) combined with the recent interest by the mobile phone vendors and cellular industry in open programming environments and platforms, typified by the recent release of the Android platform [2] and the Apple iPhone SDK [1], is accelerating the development of new people-centric sensing applications and systems [3].

In this paper, we present the design, implementation, evaluation, and user experiences of the CenceMe application [4], a new people-centric sensing application. CenceMe exploits off-the-shelf sensor-enabled mobile phones to automatically infer people's sensing presence (e.g., dancing at a party with friends) and then shares this presence through social network portals such as Facebook. We evaluate a number of important system performance issues and present the results from a user study based on an experiment conducted over a three week period in a campus town. The user study included twenty two users consisting of undergraduates, graduates and faculty at Dartmouth College.

We discuss results, experiences, and lessons learnt from the deployment of CenceMe on off-the-shelf mobile phones. These phones, while fairly powerful computers, present a number of limitations in supporting the demands of a continuous personal sensing application such as CenceMe. We implement CenceMe on the Nokia N95 phones. Although the N95 is a top-end device with a great deal of computation capability, the Symbian operating system and Java Micro Edition (JME) virtual machine which runs on top of the N95 are rather limiting due to the fact that they have both been designed to use small amounts of memory and computational resources. Additional implementation challenges arise from the fact that manufacturers and operators limit the programmability of mobile phones to preserve the closed nature of their devices and operational networks. For this reason appropriate certificates purchased from a Certificate Authority are needed, yet are not sufficient for full deployment of an application such as CenceMe. We show the tradeoffs and discuss the difficulties in implementing an always-on sensing application on the Symbian/JME platform which more generally is designed to accommodate simple applications such as gaming and calendar plugins.

The contribution of this paper is as follows:

- The design, implementation, and evaluation of a fully functional personal mobile sensor system using an unmodified mobile phone platform.
- The design of lightweight classifiers, running on mobile phones, which realize a split-level classification paradigm. We show they have a limited impact on the phone's functionality.
- Measurements of the RAM, CPU, and energy performance of the classifiers and the CenceMe software suite as a whole, showing the tradeoff between the time fidelity of the data and the latency in sharing that data.
- A validation of the CenceMe application through a user study. This is one of the first user studies that involves a large group of people using a personal sensing application running on off-the-shelf mobile phones for a continuous period of time. The study provides useful insights into how people understand and relate to personal sensing technology. The study offers some suggestions on the further development of people-centric sensing applications.

The structure of the paper is as follows. In Section 2, we present a number of design considerations when building an always-on sensing application such as CenceMe on mobile phones. The CenceMe implementation is discussed in Section 3, while in Section 4 the phone and backend classifier algorithms are presented. In Section 5, we show the performance of the CenceMe classification algorithms as well as detailed power, RAM, and CPU measurements. In Section 6, we present the results of our user study and then in Section 7 discuss related work. We conclude in Section 8 with some final remarks.

2. DESIGN CONSIDERATIONS

Before describing the implementation of the CenceMe application on the phone and backend servers, we first discuss the system development challenges encountered when implementing an application such as CenceMe on the phone. These impact several aspects of the architectural design.

2.1 Mobile Phone Limitations

OS Limitations. Although top-end mobile phones have good computational capability, often including multiple processors, they are limited in terms of the programmability and resource usage control offered to the developer. For example, the Nokia N95 is equipped with a 330 MHz ARM processor, 220 MHz DSP, and 128 MB RAM. However, when developing a non-trivial application on mobile phones a number of challenges arise. This is due in part because mobile phones are primarily designed for handling phone calls in a robust and resilient manner. As a result, third party applications running on the phone may be denied resource requests and must be designed to allow interruption at any time so as not to disrupt regular operation of the phone. This places a heavy burden on application exception handling and recovery software. While programmers may expect exception handlers to be called rarely, in Symbian they are called often and are critical to keeping an application and the phone operational. At the same time, testing exception handlers is difficult because a voice call can interrupt application code at any point in its execution; OS induced exceptions are outside the control of the programmer.

API and Operational Limitations. Additional limitations arise from the APIs provided by the phone manufacturers. JME implements a reduced set of the Java Standard Edition APIs for use on mobile phones. Because each phone model is different even from the same manufacturer, the Symbian OS and JME must be ported to each phone which typically results in missing or malfunctioning APIs for important new or existing components, such as an accelerometer or GPS. These API limitations may not be resolved by the manufacturer because new models replace old models in quick succession. As a result, the programmer is forced to come up with creative solutions to API limitations. Examples of such API limitations and operational problems encountered with the N95 include a missing JME API to access the N95 internal accelerometer and JME audio API that exhibits a memory leak, respectively.

Security Limitations. To preserve the phone's integrity and protect the cellular network from malicious attacks, phone manufacturers and cellular network operators control access to critical components, including the APIs for access to the file system, multimedia features, Bluetooth, GPS, and communications via GPRS or WiFi, through a rights management system. Properly signed keys from a Certificate Authority are needed to remove all restrictions on using these APIs.

Energy Management Limitations. An important driver for application designers on mobile phone platforms is power conservation, in particular, when radio interfaces such as Bluetooth, GPS, and GPRS are used by the application. As we show in Section 5, the phone's Bluetooth, GPS, and GPRS radios are responsible for draining most of the battery power when CenceMe is running. As application developers, we want to build applications that offer good fidelity and user experience without significantly altering the operational lifetime of the standard mobile phone. Therefore, designing efficient duty cycles for the application and its use of power hungry radios such Bluetooth and GPS radio is necessary to extend the phone's battery life. In addition to the power consumed by Bluetooth and GPS, data upload from the phone via GPRS can also draw a large amount of power, particularly when the phone is far from a cell base station. A challenge is therefore to reduce the use of these radios without significantly impacting the application experience. Currently, the Symbian version of JME does not provide APIs to power cycle (i.e., toggle on and off) the Bluetooth and GPS radios to implement an efficient radio duty-cycle strategy.

The sensing and classification algorithms that run on the phone can also consume a considerable amount of energy if left unchecked. As discussed in Section 5, sampling the phone's microphone and running a discrete Fourier transform on the sound sample uses more power than sampling the accelerometer and classifying the accelerometer data. Given this, the only way to reduce energy at the application layer is to design a sensing duty-cycle that samples sensors less frequently and avoids the use of the radios for communications or acquisition of satellite signals for location coordinates.

2.2 Architectural Design Issues

In response to the observations discussed above we design the CenceMe application using split-level classification and power aware duty-cycling. We also develop the application with software portability in mind.

Split-Level Classification. The task of classifying streams of sensor data from a large number of mobile phones is computationally intensive, potentially limiting the scalability of the system. With this in mind we propose the idea of pushing some classification to the phone and some to the backend servers. However, some classifiers require data that is only available at the server (e.g., for multiple users in the case of the social context classification discussed in Section 4.2). We call the output of the classification process on the phone *primitives*. When primitives arrive at the backend they are stored in a database and are ready to be retrieved for a second level of more complex classification. The classification operation on the backend returns *facts*, which are stored in a database from where they can be retrieved and published. With the split-level classification approach some of the classification can be done on the phone with the support of the backend, or under certain circumstances done entirely on the phone.

CenceMe's split-level design offers a number of important advantages: *i)* supports *customized tags*. A customized tag is any form of activity, gesture, or classified audio primitive that the user can bind to a personal meaning. For example, a customized tag could be created by a user by associating a certain movement or gesture of the phone (e.g., the phone being moved along an imaginary circle) with a user supplied meaning or action, e.g., going to lunch. After associating the tag "lunch" with the action, the next time the user repeats the action the user's presence state "lunch" is recognized, uploaded, and shared with their social network. This technique gives the user the freedom to build her own classified state beyond a set of defaults offered by CenceMe, hence, it provides extensibility of the application; *ii)* provides resiliency to cellular/WiFi radio dropouts. By pushing the classification of primitives to the phone, the primitives are computed and buffered when there is no or intermittent radio coverage. Primitives are stored and uploaded in batches when the radio coverage becomes available; *iii)* minimizes the sensor data the phone sends to the backend servers improving the system efficiency by only uploading classification derived-primitives rather than higher bandwidth raw sensed data; *iv)* reduces the energy consumed by the phone and therefore monetary cost for the data cellular connection by merging consecutive uploads of primitives; and finally *v)* negates the need to send raw sensor data to the backend, enhancing the user's privacy and data integrity.

As discussed in Section 4, we design the classifiers that produce the primitives to be lightweight to match the capabilities of the phone.

Power Aware Duty-Cycle. To extend the battery lifetime of the phone when running the CenceMe application we apply scheduled sleep techniques to both the data upload and the sensing components. This leads to the following question: how long can the sensors, Bluetooth, GPS, and communications upload be in a sleep mode given that the larger the sleep interval the lower the classification responsiveness of the system? Typically, a real time sensing system would supply sensor data using a high rate duty cycle. However, such an approach would conflict with energy conservation needs. Our approach is based on a duty cycle design point that minimizes sampling while maintaining the application's responsiveness, as judged by users. This design strategy allows CenceMe to operate as near to real-time as is possible; that is, some system delay is introduced before a person's sensing presence is updated on the backend servers. In the case of the current implementation the introduced delay varies according to the type of presence being inferred. The introduction of delay to improve the overall energy efficiency of the system makes good sense given the goal of CenceMe to allow buddies in social networks to casually view each other's sensing presence. For example, knowing that a buddy is in a conversation one minute after the actual conversation began seems reasonable. Other activities may allow even greater introduced latency; for example, people remain at parties for periods typically greater than five minutes or more, therefore, the delay introduced by the classifier in this case has little effect on the accuracy of the system status reports. In Section 5.2 we present the CenceMe system performance evaluation under varying upload and sensing duty cycles to best understand these trade-offs. In Section 6 we discuss results from the user study that indicate that even though users view their buddies status via the CenceMe portal infrequently they expect current information when viewed to be accurate and timely. This lends itself to a design that senses at an even lower duty cycle on average but temporarily increases the sensing rate when a buddy's page is accessed. This results in bandwidth and storage capacity improvements.

Software Portability. To design for better software portability we push as much as we can to JME. We follow this design goal to maximize software re-usability given that the majority of modern mobile phones use a Java virtual machine to support JME programs. However, because of the API limitations discussed earlier, a number of components need to be implemented directly using native Symbian APIs to support the necessary features offered by the phone but not available through JME.

3. CENCEME IMPLEMENTATION

In this section, we present the CenceMe implementation details. The CenceMe application and system support consists of a software suite running on Nokia N95 mobile phones and backend infrastructure hosted on server machines. The software installed on the phones performs the following operations: sensing, classification of the raw sensed data to produce primitives, presentation of people's presence directly on the phone, and the upload of the primitives to the backend servers. Primitives are the result of: *i)* the classification of sound samples from the phone's microphone using a discrete Fourier transform (DFT) technique and a machine learning algorithm to classify the nature of the sound; *ii)* the classification of on board accelerometer data to determine the activity, (e.g., sitting, standing, walking, running); *iii)* scanned Bluetooth MAC addresses in the phone's vicinity; *iv)* GPS readings; and finally, *v)* random photos, where a picture is taken randomly when a phone keypad key is pressed or a call is received. Classification algorithms that infer more complex forms of sensing presence (i.e., facts) run on backend machines, as discussed in Section 4.2.

3.1 Phone Software

Figure 1 shows the CenceMe software architecture for the

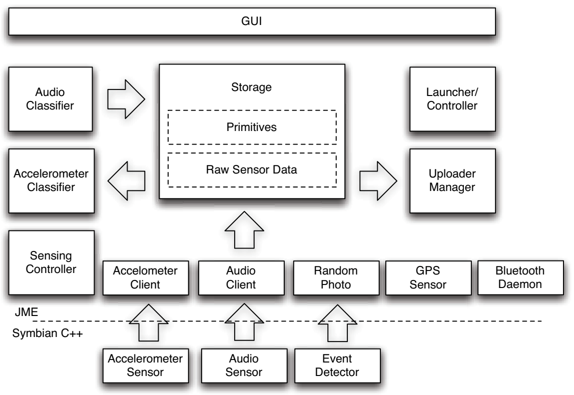
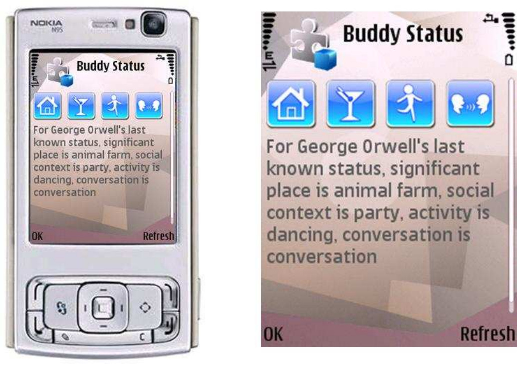


Figure 1: Architecture of the CenceMe phone software.



For George Orwell's last known status, significant place is animal farm, social context is party, activity is dancing, conversation is conversation

Figure 2: ClickStatus on the Nokia N95.

Nokia N95 phone. The phone architecture comprises the following software components:

Symbian Servers. The accelerometer sensor, audio sensor, and event detector sensor are Symbian C++ modules that act as daemons producing data for corresponding JME client methods. Their function is, respectively: polling the on board accelerometer sensor, sampling the phone's microphone, and detecting incoming/outgoing calls and keypad key presses. The sensed data is sent to the JME methods through a socket. Events detected by the event detector daemon are used by the random photo module at the JME level to generate random pictures, to trigger a photo upon an incoming phone call or to signal the application that it has to restart after a phone call for reliability reasons.

Bluetooth Daemon. This component resides at the JME level and is used to perform an inquiry over the Bluetooth radio to retrieve the MAC addresses of any neighboring Bluetooth nodes. The MAC addresses of the neighboring nodes are used to determine if there are CenceMe phones in the area at the time of the inquiry.

Accelerometer Client. This component is written in JME and connects through a socket to the accelerometer sensor to retrieve the accelerometer data byte stream. The byte stream is stored in local storage and retrieved by the activity classifier to compute the activity primitive, as discussed in Section 4.1.

Audio Client. This JME client component connects through a socket to the Symbian audio server to retrieve the audio byte stream that carries the PCM encoded representation of the sound sample. The byte stream is stored in local storage and retrieved by the audio classifier to compute the audio primitive, as discussed in Section 4.1.

Random Photo. This JME module is designed to trigger the capture of a photo upon detection of incoming calls or pressed keypad keys. The events are received through a socket from the event detector daemon. When the picture is taken it is stored locally until the next upload session.

GPS. The JME GPS implementation supplies a callback method that is periodically called by the Nokia GPS daemon to provide the geographical location of the phone. The GPS coordinates are stored locally and then uploaded to the backend servers.

Sensing Controller. This component is responsible for orchestrating the underlying JME sensing components. The sensing controller starts, stops, and monitors the sensor clients and the Bluetooth manager and GPS daemon to guarantee the proper operation of the system.

Local Storage. This component stores the raw sensed data records to be processed by the phone classifiers. As the classification of raw data records is performed, the data records are discarded, hence none of the sampled data persists on the phone. This is particularly important to address the integrity of the data and the privacy of the person carrying the phone since none of the raw sensed data is ever transferred to the backend. Primitives, GPS coordinates, and Bluetooth scanned MAC addresses are stored in local storage as well, waiting for an upload session to start.

Upload Manager. This component is responsible for establishing connections to the backend servers in an opportunistic way, depending on radio link availability which can be either cellular or WiFi. It also uploads the primitives from local storage and tears down the connection after the data is transferred. Details about how the upload manager interacts with the backend are discussed in Section 3.2.

Privacy Settings GUI. The privacy settings GUI allows the user to enable and disable the five sensing modalities supported on the phone, (viz. audio, accelerometer, Bluetooth, random photo, and GPS). Users can control the privacy policy settings from the phone and the CenceMe portal. By doing so users determine what parts of their presence to share and who they are willing to share sensing presence with or not as the case may be.

ClickStatus. To complement the full visualization of current and historical sensing presence available via the CenceMe portal (a screenshot of the portal is shown in [12]), we developed ClickStatus, a visualization client that runs on the mobile phone. The sensing presence is rendered as both icons and text on the phone GUI, as shown in Figure 2. The presence rendered by ClickStatus is subject to the same privacy policies settings as when viewed using the CenceMe portal.

After a user logs in with their CenceMe credentials, they are presented with a list of their CenceMe buddies downloaded from the CenceMe server. CenceMe buddies are Facebook friends running CenceMe on their N95. While this is always done at start up, a user has the ability to refresh their buddy list at any time via a menu command option. By highlighting and selecting a buddy from buddy list, a user triggers ClickStatus to fetch via GPRS or WiFi the latest known sensing presence for the selected buddy from the CenceMe server. This presence is displayed on a separate result screen; from there a user can either exit to return to

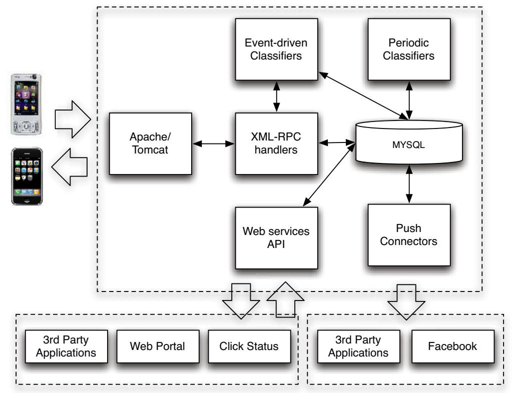


Figure 3: Software architecture of the CenceMe backend.

their buddy list or refresh the currently displayed buddy's presence.

WatchTasks. The purpose of WatchTasks is to restart any process that fails. WatchTasks also serves several other ancillary purposes including: *i)* launching CenceMe when the phone is turned on; *ii)* starting the CenceMe application software components in the correct order; *iii)* restarting the CenceMe midlet after a phone call is complete. This is detected when the event detector daemon exits, signaling the end of a call; *iv)* restarting all support daemons when CenceMe fails. Such action is necessary when we cannot reconnect to specific daemons under certain failure conditions; and finally *v)* restarting all the CenceMe software components at a preset interval to clear any malfunctioning threads.

The CenceMe phone suite uses a threaded architecture where each JME component shown in Figure 1 is designed to be a single thread. This ensures that component failure does not compromise or block other components.

3.2 Backend Software

The CenceMe backend software architecture is shown in Figure 3. All software components are written in Java and use Apache 2.2 and Tomcat 5.5 to service primitives from phones and the application requests from the CenceMe portal, ClickStatus, and Facebook. Communications between the phone and the backend uses remote procedure calls implemented by the Apache XML-RPC library on the server. Requests are handled by Java servlets in combination with a MySQL database for storage.

Phone $\Leftrightarrow$ Backend Communications. Data exchange between the phone and the backend is initiated by the phone at timed intervals whenever the phone has primitives to upload. Primitives are uploaded through XML-RPC requests. Once primitives are received at the backend they are inserted into the MySQL database.

Backend-to-phone communications such as in the significant places service described in Section 4.2 are piggybacked on both: *i)* the return message from XML-RPC requests initiated by the phone for primitive upload or periodic ping messages that the phone sends with an ad-hoc XML-RPC control message; and *ii)* the XML-RPC acknowledgment sent to the phone in response to a primitive upload.

Presence Representation and Publishing. CenceMe presence is represented through a set of icons that capture the actual presence of a person in an intuitive way. For example, if a person is driving a car they are represented by the car icon; if a person is engaged in a conversation, an icon of two people talking represents the state. CenceMe publishes presence by means of either a "pull" or "push" approach. Popular applications such as Facebook and MySpace require a push approach. This allows content to be inserted via some variant of a HTTP transported markup language (e.g., FBML, XML). Other applications such as Skype, Pidgin, and iGoogle require a pull mechanism to make content available. The CenceMe backend supports pull-based data publishing by exposing a standard web service based API. This API is also used to support the data needs of CenceMe components such as ClickStatus and the CenceMe portal. Push-based publishing is supported by the PushConnector component shown in Figure 3. This component handles the generic operation of pushing CenceMe presence based on user preferences to a number of applications. For the Facebook implementation, three Facebook widgets are offered to expose a subset of the functionality available on the portal, namely, BuddySP, Sensor Status, and Sensor Presence. Buddy SP is a buddy list replacement widget that lists CenceMe friends for user navigation. It is the same as the standard widget that lists friends within Facebook but augments this list with a mini-sensor presence icon view. Sensor Status provides automated textual status message updates such as "Joe is at work, in a conversation, standing". Finally, Sensor Presence provides a simplified version of the user's current status through an iconized representation of the user's presence.

4. CENCEME CLASSIFIERS

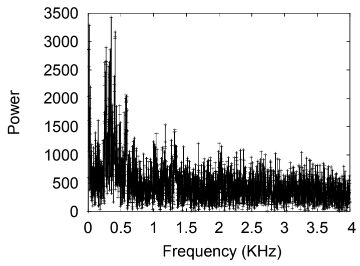
In this section, we discuss the algorithms used by the CenceMe classifiers running on the phone and the backend according to the split-level classification design discussed earlier.

4.1 Phone Classifiers

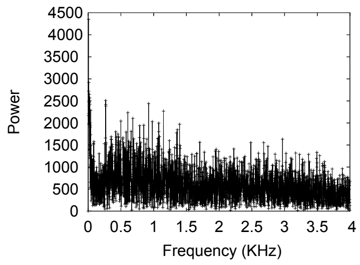
Audio classifier. The audio classifier retrieves the PCM sound byte stream from the phone's local storage and outputs the audio primitive resulting from the classification. The primitive is stored back in local storage (see Figure 1). This audio primitive indicates whether the audio sample represents human voice and is used by backend classifiers such as the conversation classifier, as discussed in Section 4.2.

The audio classification on the phone involves two steps: feature extraction from the audio sample and classification. The feature extraction is performed by running a 4096 bin size discrete Fourier transform (DFT) algorithm. A fast Fourier transform (FFT) algorithm is under development.

An extensive a-priori analysis of several sound samples from different people speaking indicated that Nokia N95 sound streams associated with human voice present most of their energy within a narrow portion of the 0-4 KHz spectrum. Figures 4(a) and 4(b) show the DFT output from two sound samples collected using the Nokia N95. The plots show the capture of a human voice, and the sound of an environment where there is not any active conversation ongoing, respectively. It is evident that in the voice case most of the power concentrates in the portion of spectrum between ~250 Hz and ~600 Hz. This observation enables us to optimize the DFT algorithm to be efficient and lightweight



(a) DFT of a human voice sample registered by a Nokia N95 phone microphone.



(b) DFT of an audio sample from a noisy environment registered by a Nokia N95 phone microphone.

Figure 4

by operating in the ∼250 Hz to ∼600 Hz frequency range. Classification follows feature extraction based on a machine learning algorithm using the supervised learning technique of discriminant analysis. As part of the training set for the learning algorithm we collected a large set of human voice samples from over twenty people, and a set of audio samples for various environmental conditions including quiet and noisy settings.

The classifier's feature vector is composed of the mean and standard deviation of the DFT power. The mean is used because the absence of talking shifts the mean lower. The standard deviation is used because the variation of the power in the spectrum under analysis is larger when talking is present, as shown in Figure 4. Figure 5 shows the clustering that results from the discriminant analysis algorithm using the mean and standard deviation of the DFT power of the sound samples collected during the training phase. The equation of the dashed line in Figure 5 is used by the audio classifier running on the phone to discern whether the sound samples comes from human voice or a noisy/quite environment with 22% mis-classification rate. Audio samples misclassified as voice are filtered out by a rolling window technique used by the conversation classifier that runs on the backend, as discussed in Section 4.2. This boosts the performance fidelity of the system for conversation recognition.

Activity classifier. The activity classifier fetches the raw accelerometer data from the phone's local storage (see Figure 1), and classifies this data in order to return the current activity, namely, sitting, standing, walking, and running. The activity classifier consists of two components: the preprocessor and the classifier itself.

The preprocessor fetches the raw data from the local storage component and extracts features (i.e., attributes). Given the computational and memory constraints of mobile phones, we use a simple features extraction technique which prove

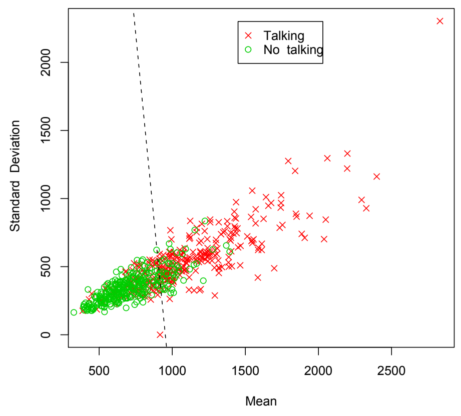


Figure 5: Discriminant analysis clustering. The dashed line is determined by the discriminant analysis algorithm and represents the threshold between talking and not talking.

to be sufficiently effective, rather than more computationally demanding operations such as FFT. The preprocessor calculates the mean, standard deviation, and number of peaks of the accelerometer readings along the three axes of the accelerometer.

Figure 6 shows the raw N95 accelerometer readings along the three axes for sitting, standing, walking, and running for one person carrying the phone. As expected, the sitting and standing traces are flatter than when the person is walking and running. When standing, the deviation from the mean is slightly larger because typically people tend to rock a bit while standing. The peaks in the walking and running traces are a good indicator of footstep frequency. When the person runs a larger number of peaks per second is registered than when people walk. The standard deviation is larger for the running case than walking. Given these observations, we find that the mean, standard deviation, and the number of peaks per unit time are accurate feature vector components, providing high classification accuracy. Because of lack of space, we do not report similar results to those shown in Figure 6 for other people. However, we observe strong similarities in the behavior of the mean, standard deviation, and the number of peaks for the accelerometer data across different individuals.

Our classification algorithm is based on a decision tree technique [32][33]. The training process of the classifier is run off-line on desktop machines because it is computationally costly. In order to maximize the positive inference of an individual's activity, prior work suggests that the best place on the body to carry a phone is the hip [34]. After interviewing the participants in our user study, we conjecture that most of people carry their phones in their pants pockets, clipped to a belt or in a bag. We collected training data from ten people that randomly placed the mobile phone inside the front and back pockets of their pants for several days. We plan to consider other usage cases in future work.

At the end of the training phase, we feed the training set to the J48 decision tree algorithm, which is part of the WEKA workbench [28]. The output of the decision tree algorithm is a small tree with depth three. Such an algorithm

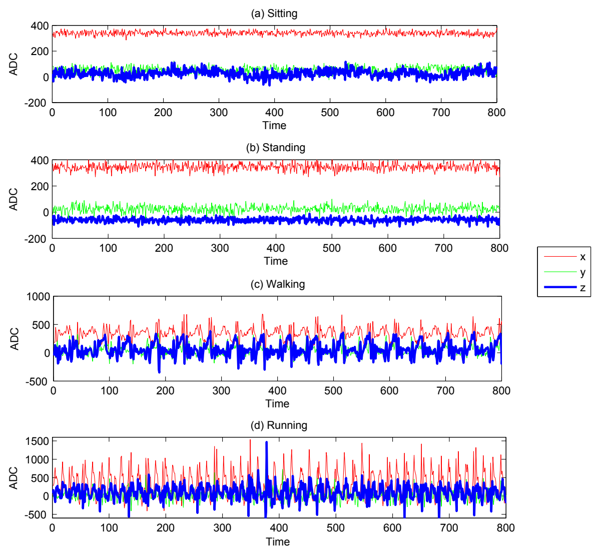


Figure 6: Accelerometer data collected by the N95 on board accelerometer when the person carrying the phone performs different activities: sitting, standing, walking, and running.

is lightweight and efficient. The time needed by the preprocessor and the classifier to complete the classification process is less than 1 second on average running on the Nokia N95.

4.2 Backend Classifiers

Backend classifiers follow the split-level classification design and generate facts based on primitives provided by the phone or facts produced by other backend classifiers. Facts represent higher level forms of classification including social context (meeting, partying, dancing), social neighborhood, significant places, and statistics over a large group of data (e.g., does a person party more than others, or, go to the gym more than others?). However, some of the classifiers (e.g., conversation and CenceMe neighborhood) will eventually be pushed down to the phone to increase the system classification responsiveness. In this case, the primitives would still be uploaded to the backend in order to make them available to other backend classifiers.

Backend classifier processing is invoked in two ways: either event triggered or periodic. An example of an event triggered classifier is the "party" classifier: it receives as input the primitives from the phone that contain the volume of an audio sample and the activity of the user and returns whether the person is at a party and dancing. Along with trigger based classifiers there is a collection of periodically executed classifiers. An example of such classifiers is the "Am I Hot" classifier that runs periodically according to the availability of data in a window of time, (i.e., day long data chunk sizes).

In what follows, we describe the backend classifiers and their implementation in more detail.

Conversation Classifier. This classifier's purpose is to determine whether a person is in a conversation or not, taking as input the audio primitives from the phone. However, given the nature of a conversation, which represents a combination of speech and silences, and the timing of sampling, the audio primitive on the phone could represent a silence during a conversation. Thus, the phone's audio primitives are not accurate enough to determine if a person is in the middle of a conversation. To address this the backend conversation classifier uses a rolling window of N phone audio primitives. The current implementation uses N=5 to achieve classification responsiveness, as discussed in Section 5.1.

The rolling window filters out pauses during a conversation to remain latched in the conversation state. The classifier triggers the "conversation" state if two out of five audio primitives indicate voice. The "no conversation" state is returned if four out of five audio primitives indicate a "no voice". We determined experimentally that fewer samples are needed to trigger the conversation state than no conversation state. We therefore design the conversation classifier following an asymmetric strategy that quickly latches into the conversation state but moves more conservatively out of that state. We made this choice because if the conversation classifier can be used as a hint to determine if a person can be interrupted (for instance with a phone call), then we only want to drop out of conversation state when the conversation has definitely ended.

The accuracy of the conversation classifier is discussed in Section 5.1.

Social Context. The output of this classifier is the social context fact, which is derived from multiple primitives and facts provided by the phone and other backend classifiers, respectively. The social context of a person consists of: *i)* neighborhood conditions, which determines if there are any CenceMe buddies in a person's surrounding area or not. The classifier checks whether the Bluetooth MAC addresses scanned by the phone, and transmitted to the backend as a primitive are from devices belonging to CenceMe buddies (i.e., the system stores the Bluetooth MAC addresses of the phones when CenceMe is installed); *ii)* social status, which builds on the output of the conversation and activity classifiers, and detected neighboring CenceMe buddies to determine if a person is gathered with CenceMe buddies, talking (for example at a meeting or restaurant), alone, or at a party. For example, by combining the output of the conversation classifier, the activity primitive, and neighboring Bluetooth MAC addresses a person might be classified as sitting in conversation with CenceMe friends. Social status also includes the classification of partying and dancing. In this case a combination of sound volume and activity is used. We use a simple approach that uses an audio volume threshold to infer that a person is at a party or not. Training for this is based on a few hours of sound clips from live parties using the N95 microphone. We also take a simple approach to the classification of dancing. We determine a person is dancing if the person is in the "party" state and the activity level is close to running, given that the accelerometer data trace for running is close to dancing. Although we realize the definition of social context is somewhat simplistic and could be improved, this is a first step toward the representation of people's status and surroundings in an automated way.

Mobility Mode Detector. We employ GPS location estimates as input to a mobility mode classifier [22][23]. This classification is currently only binary in its output, classifying the mobility pattern as being either traveling in a vehicle or not (i.e., being stationary, walking, running). We use a simple feature vector based on multiple measures of speed; that is, using multiple distance/time measurements for variable sizes of windowed GPS samples and the built-in speed estimation of the GPS device itself. The classifier is built with the JRIP rule learning algorithm, as implemented in WEKA [28], based upon manually labeled traces of GPS samples. We compensate for any inaccuracy in GPS samples by filtering based on the quality measures (i.e., horizontal dilution of precision and satellite counts) and outlier rejection relative to the estimates of previous and subsequent GPS samples.

Location Classifier. The function of this component is to classify the location estimates of users for use by other backend classifiers. GPS samples are filtered based on quality (as discussed above) to produce a final location estimate. Classification is driven based on bindings maintained between a physical location and a tuple containing: *i)* a short textual description; *ii)* an appropriate icon representation; and *iii)* a generic class of location type (i.e., restaurant, library, etc.). Bindings are sourced from GIS databases and CenceMe users. We use the Wikimapia [27] for GIS data in our implementation. Relying solely on GIS information limits the richness of shared presence. Typically, people tend to spend a larger proportion of their time in relatively few locations. This motivates the idea of user-created bindings. CenceMe allows users to insert their own bindings via either the portal or the phone. Using the phone, users can manually bind a location when they visit it. Similarly, users can use the portal to also add, edit or delete bindings manually. CenceMe also provides the ability to learn significant places in an automated manner in contrast to the manual bindings discussed above. New bindings learned by the system are based on the mobility pattern of the user. This aspect of CenceMe directly builds on the existing work in location trace analysis referred to as significant places [25] [31]. In CenceMe we perform k-means clustering using WEKA [28] where the parameters of the clustering algorithm are determined experimentally. Once a potential significant place is discovered the next time the person enters that location the phone prompts the person's mobile phone to confirm or edit the details of the location. Default labels and icons are initially based upon the most popular nearest known existing binding defined by the user or CenceMe buddies. To reduce the burden on the users to train the classifier with their own bindings we structure the classifier to initially borrow existing bindings from their CenceMe buddies [24].

Am I Hot. Making the large volumes of data collected by CenceMe easily digestible to users is a challenge. We address this challenge using a series of simple and meaningful metrics that relate historical trends in user data to either recognizable social stereotypes or desirable behavioral patterns. These metrics are calculated on a daily basis and users view patterns in their own data and compare themselves with their buddies. The metrics include the following: *i)* nerdy, which is based on individuals with behavioral trends such as being alone (from the Bluetooth activity registered by the person's phone), spending large fractions of time in certain locations (e.g., libraries) and only infrequently engaging in conversation; *ii)* party animal, which is based on the frequency and duration with which people attend parties and also takes into account the level of social interaction; *iii)* cultured, which is largely location based, being driven by the frequency and duration of visits to locations such as theaters and museums; *iv)* healthy, which is based upon physical activities of the user (e.g., walking, jogging, cycling, going to the gym); and finally, *v)* greeny, which identifies users having low environmental impact, penalizing those who drive their cars regularly while rewarding those who regularly walk, cycle or run.

Table 1: Activity classifier confusion matrix

	Sitting	*Standing*	*Walking*	*Running*
Sitting	0.6818	0.2818	0.0364	0.0000
Standing	0.2096	0.7844	0.0060	0.0000
Walking	0.0025	0.0455	0.9444	0.0076
Running	0.0084	0.0700	0.1765	0.7451

Table 2: Conversation classifier confusion matrix

	Conversation	*Non-Conversation*
Conversation	0.8382	0.1618
Non-Conversation	0.3678	0.6322

5. SYSTEM PERFORMANCE

In this section, we present an evaluation of the CenceMe application and system support. We start by discussing the performance of the CenceMe classifiers and then present a set of detailed power, memory, and CPU benchmarks. Finally, we present the results from a detailed user study.

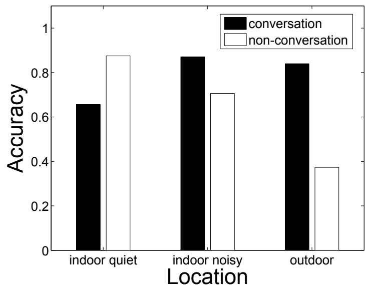
5.1 Classifiers Performance

We examine the classifiers performance based on a small-scale supervised experiments. We discuss classifier accuracy, and the impact of mobile phone placement on the body, environmental conditions, and sensing duty cycle. The results are based on eight users who annotate their actions over a one week period at intervals of approximately 15 to 30 minutes, unless otherwise stated. Annotations act as the ground truth for comparison with classifier outputs. The ground truth data is correlated to the inference made by the CenceMe classifiers. This data is collected at different locations and by carrying the mobile phone in various positions on the body. Tables 1, 2 and 3 show the confusion matrices for the activity, conversation, and mobility classifiers, respectively, over a one week period. These reported values represent good approximations; the human annotations may be inaccurate or incomplete at times.

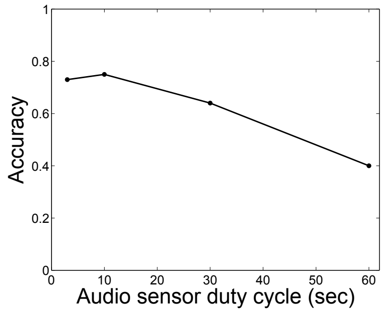
5.1.1 General Results

While the activity inference accuracy reported in Table 1 is up to 20% lower than that reported using custom hardware [14], we achieve our results using only the accelerometer on a Nokia N95 and engineering the system to be power efficient and work around the resource limitations discussed earlier. We find that our classifier has difficulty differentiating sitting and standing given the similarity in the raw accelerometer traces, as shown in Figure 6. We observe that variations in locale (e.g., office, restaurant) and people (e.g., body type, weight) do not significantly impact the activity classification performance.

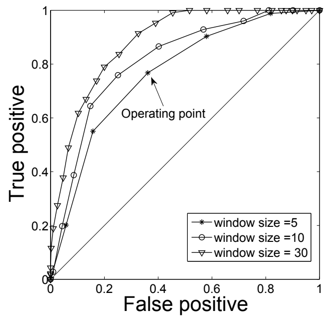
The conversation classification accuracy reported in Table 2 is high, but the classifier also suffers from a relatively high rate of false positives. This is due to a combination of classifier design and "mis-annotation" by participants. The classifier reports conversation even if the person carrying the phone is silent but someone is talking nearby. Naturally, participants often did not account for this fact. Furthermore, due to the asymmetric state latching for the conversation



(a) Conversation classifier in different locations.



(b) Conversation classifier accuracy with a variable duty cycle.



(c) ROC curves for the conversation classifier.

Figure 8

Table 3: Mobility mode classifier confusion matrix

	Vehicle	*No Vehicle*
Vehicle	0.6824	0.3176
No Vehicle	0.0327	0.9673

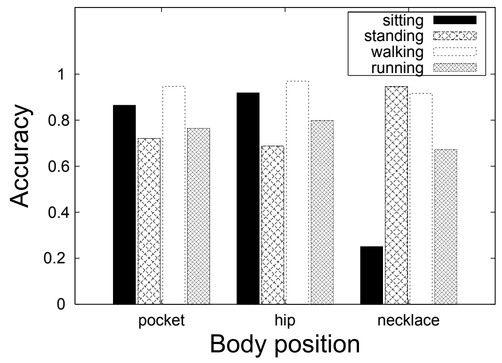


Figure 7: Activity classification vs. body position.

classifier discussed in Section 4.2, the classifier remains in the conversation state for a longer time than the real conversation duration, generating false positives.

5.1.2 Impact of Phone Placement on the Body

While mobile phone placement on the body is a personal choice, prior work has shown body placement to affect the accuracy of activity inference [34]. We assess the impact on classification when the Nokia N95 is placed at different places on the body, namely, in a pocket, on a lanyard, and clipped to a belt. Classification accuracy derived from the ground truth annotated data is shown in Figure 7. The pocket and belt positions produce similar results for all classified activities, while the lanyard position yields poor accuracy when classifying sitting, and a relatively lower accuracy for running. In follow-up laboratory experiments, we find that the length of the lanyard cord and the type of lanyard we provided to participants affect the results. If the lanyard is long the phone rests frequently on the body, particularly while walking and standing, allowing for accurate classification. However, even when seated a lanyard-mounted phone may swing from side to side with incidental torso movements, causing a mis-classification as standing or walking. Furthermore, running is sometimes classified as walking because the lanyard damps the accelerometer signatures that indicate running, compared to other body positions (e.g., belt, pocket) where the phone is more rigidly affixed to the body.

We find that conversation classification accuracy is much less sensitive to the body placement of the phone. When the phone is worn as a lanyard, conversation and no conversation are detected with 88% and 72% accuracy, respectively. The same test repeated with the phone in a pocket yields a classification accuracy of 82% for conversation and 71% for no conversation, despite the muffling effect of clothing.

5.1.3 Impact of Environment

We find activity classification accuracy to be independent of environment. Mobility classification is inherently not tied to a particular location but rather on transitions between locations. However, we do see an impact from the environment on conversation classification accuracy. Figure 8(a) shows the classification accuracy categorized by location, where the different locations are: outdoors, indoor noisy (i.e., an indoor location with background noise such as in a cafe or restaurant), and indoor quiet (i.e., with very low background noise such as at the library or office). The classifier detects conversation with more than an 85% success rate when in an indoor noisy environment. In outdoor scenarios there is an increase in false positives but the accuracy of detection of conversation, a design focus, remains high. Lower conversation detection accuracy in very quiet indoor environments occurs because the classifier is trained with the average case background noise. In a noisy environment there is an increase in power across all of the frequencies so a threshold set for this environment in mind will be larger than if a very quiet environment is assumed. As a result, in very quiet environments fewer conversations are detected since the contribution of background noise is lower. These performance characteristics are a direct result of the audio classifier design, which attempts to reduce the use of the phone's resources.

5.1.4 Impact of Duty Cycle

Applying a sleep scheduling strategy to the sensing routine is needed in order to increase the battery lifetime of the phone. Note that in Section 5.2 we discuss lifetime gains with a ten minute inter-sample time. However, this has a negative impact on the performance of the classifiers, particularly in detecting short-term (i.e., duration) events that occur between samples. For example, in Table 3, the vehicle state is only correctly detected 68% of the time. This lower accuracy is a product of shorter car journeys around town for durations less than the inter-sampling rate. This problem is aggravated by other factors such as the delay in acquiring good GPS-based positioning data. To investigate the impact of duty cycling on conversation classification, we set up an experiment with eight users that periodically reprogrammed their phones with different duty cycles while keeping a diary. Figure 8(b) shows the performance of the phone's conversation classifier as the microphone sensing duty cycle varies. Each value represents the average of five trials. We see that there is little benefit in adopting a sleeping time smaller than 10 seconds. However, longer duty cycles impact performance. We observe only a 40% accuracy using the conversation classification for a 60 second duty-cycle, which is the longest duty-cycle we considered experimentally.

A longer sensing duty cycle also implies a reduction of the conversation classifier rolling window size to maintain the high responsiveness of the classifier. A smaller conversation classifier rolling window size leads to a higher misclassification rate. This becomes apparent if we look at the Receiver Operating Characteristic (ROC) curves of the conversation classifier as shown in Figure 8(c). The ROC curves show the impact of the window size and threshold that triggers conversation (reflected in the curve shape) on the classifiers true positive and false positive rates. We use offline analysis to determine the output of the conversation classifier as we alter the window size and threshold value. We observe that the larger the window (i.e., N=10,30), the larger the true positives to false positives ratio becomes. In our current implementation, we adopt N=5 and an audio sensing rate of 30 seconds (our default operating point is labeled in the figure). With these parameters the worst-case conversation classification delay omitting communication delays is 1.5 minutes. On the other hand, if we used a window where N=30, which would give higher accuracy, we would get a delay of 9 minutes on average. This illustrates the trade off between sampling rate and classification speed. However, we choose to operate at a point in the design space that increases the true positive rate at the expense of being less accurate in the detection of non-conversation because the cost, from a user's perspective, of being wrong when detecting a conversation is larger than the cost of being wrong when detecting non-conversation.

5.2 Power Benchmarks

Power measurements of CenceMe are made using the Nokia Energy Profiler, a standard software tool provided by Nokia specifically for measuring energy use of applications running on Nokia hardware. The profiler measures battery voltage, current, and temperature approximately every third of a second, storing the results in RAM.

Figure 9 shows the typical contribution of various sensors and classifiers to the overall energy budget during a ten minute sensing cycle. Bluetooth proximity detection requires a 120 second scan period to capture neighboring MAC addresses due to the cache flushing limitations of the Bluetooth API in JME. GPS location detection is inherently power hungry and takes time to acquire "a lock" when turned

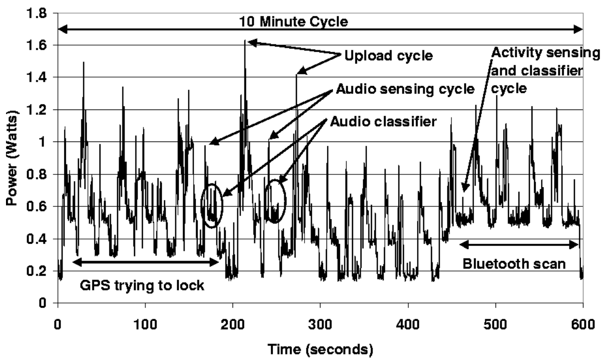


Figure 9: Details of the power consumption during a sampling/upload interval.

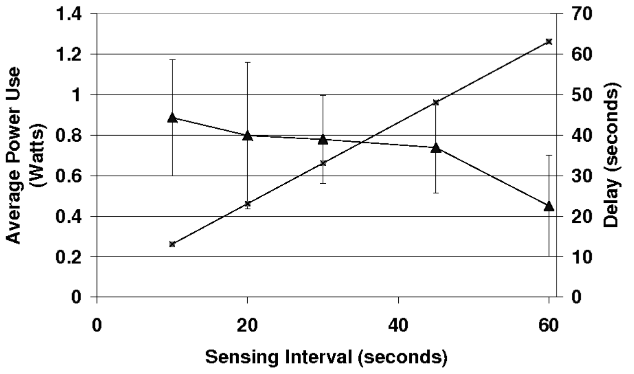


Figure 10: The tradeoff between energy consumption and data latency in CenceMe.

on. CenceMe allows 120 seconds for a lock to be acquired and then the N95 keeps the GPS activated for another 30 seconds (which is out of our control). The highest spikes shown on the plot are due to the upload of data which uses the cellular radio. The next highest spikes are due to sampling of audio data. The period of several seconds following the audio sample is where the audio classifier runs, using a relatively high amount of energy to compute a DFT. The accelerometer sampling and activity classification are fast and use little power. While this is a typical pattern of energy consumption there are other factors which can cause variations, including: distance to cell tower, environmental radio characteristics, the amount of data to upload, the number of Bluetooth neighbors, denial of resources due to the phone being in use for other purposes, network disconnections, sensor sample intervals, sample durations, upload interval, GPS lock time, and temperature.

Figure 10 shows the energy consumption measured with the profiler for sampling intervals ranging from 10 seconds to 60 seconds with power in Watts on the vertical axis. The second line and axis in the graph shows the latency in getting the facts to the backend as a function of the sample interval including the sample interval itself, classifier latency, and network delay. The audio classifier latency is actually a multiple of three times the values on this line since the classifier needs at least three facts from the phone in order to detect conversation and social setting. The horizontal axis shows the sampling interval for the accelerometer and audio. The proximity and GPS sensors are sampled at ten times the x-axis value (e.g., a 60 second interval means Bluetooth and GPS are sampled at 600 seconds, or ten minute intervals).

The combination of the two lines show the tradeoff between energy use and data latency for any particular sampling interval. There is no optimal sampling interval since users will have different requirements at different times. For example, users may want a short sample interval when they are active, a slow interval when they are inactive, and a very slow interval when their phone is running out of energy. We are currently considering several methods of automatic adaptation of the sample rate based on sensor input and battery state, combined with a user preference selector that lets the user shift the emphasis between long battery life and greater data fidelity.

Overall battery lifetime running the entire CenceMe software suite on a fully charged N95 is measured five times by running the battery to depletion under normal use conditions while using no other applications on the phone. This results in 6.22 +/- 0.59 hours of usage. The reason for the large standard deviation is that there are many factors impacting battery life such as temperature, the number of calls and duration, the number of ClickStatus queries, range from cell towers when used, and the environmental and atmospheric conditions. Without the CenceMe software running, and the phone in a completely idle state, low power state power consumption is 0.08 +/- 0.01 Watt-Hours per hour. The CenceMe suite consumes 0.9 +/- 0.3 Watt-Hours per hour when running with no user interaction. The conversation and social setting classifier consumes 0.8 +/- 0.3 Watt-Hours per hour with all other parts of the CenceMe system idle. The activity classifier consumes 0.16 +/- 0.04 Watt-Hours per hour with all other parts of the CenceMe system idle. Any use of the phone to make calls, play videos or listen to music will reduce the runtime. While the approximately 6 hour lifetime is far below the idle lifetime of the Nokia N95, we have identified several areas where we believe we can significantly reduce power usage while also decreasing data latency, as discussed in Section 2.2.

5.3 Memory and CPU Benchmarks

We also carried out benchmark experiments to quantify the RAM and CPU usage of the CenceMe software running on the N95 using the Nokia Energy Profiler tool. For all measurements we enable the screen saver to decouple the resource occupation due to the CenceMe modules from that needed to power up the N95 LCD.

We start by measuring the amount of RAM and CPU usage when the phone is idle with none of the CenceMe components running. We then repeat the measurement when either the accelerometer sampling and activity classification or audio sampling and classification are active. Then we add each of the remaining CenceMe modules until the whole software suite is running. The results are shown in Table 4. As expected, audio sampling and feature vector extraction require more computation than the other components. This is in line with the power measurements result shown in Figure 9 where audio sampling and processing are shown to use a relatively high amount of energy. We also note that the memory foot print does not grow much as components are added. Together CenceMe and ClickStatus occupy 5.48MB of RAM.

Table 4: RAM and CPU usage

	CPU	*RAM (MB)*
Phone idle	2% (+/- 0.5%)	34.08
Accel. and activity classif.	33% (+/- 3%)	34.18
Audio sampling and classif.	60% (+/- 5%)	34.59
Activity, audio, Bluetooth	60% (+/- 5%)	36.10
CenceMe	60% (+/- 5%)	36.90
CenceMe and ClickStatus	60% (+/- 5%)	39.56

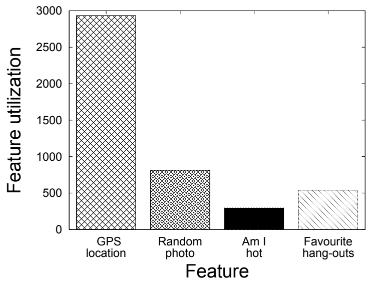
6. USER STUDY

Because CenceMe is designed to be a social network we need to go beyond simple measures of system performance to best understand the utility of people-centric sensing applications such as CenceMe. Our goal is to bring CenceMe to the attention of potential users, ask them to use CenceMe and provide detailed feedback about their user experience by means of a survey. For this reason we conducted an "operational" experiment. The experiment conducted over a three week period involved 22 people. Participants were each given a Nokia N95 with the CenceMe software (including ClickStatus) and a free voice/data plan. Users had server side accounts and access to the CenceMe portal. While some of the users were friends we placed all users in the same buddy list as a means to create some community.

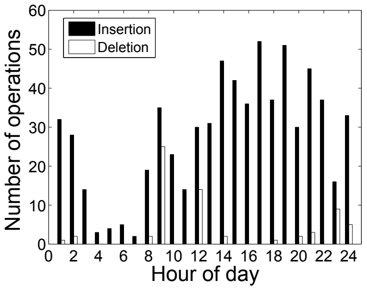
The pool of candidates picked within the population of students and staff at our university was composed of 12 undergraduate students, 1 research assistant, 1 staff engineer, 7 graduate students, and 1 professor. The research assistant and four undergraduates have little computer science background. Sixteen participants are active Facebook users. Before discussing the detailed experience of users, we summarize some results from the user study:

- Almost all of the participants liked using CenceMe and its features. One user wrote: "*it's a new way to be part of a social network*".
- Facebook users are particularly active in terms of willingness to share detailed status and presence information with their friends.
- Privacy could be a concern but users are fine with sharing their presence status as long as they have the means to easily and efficiently control their privacy settings.
- CenceMe stimulates curiosity among users. Users want to know what other people are doing while on the move.
- CenceMe can aid people in learning their own activity patterns and social status.

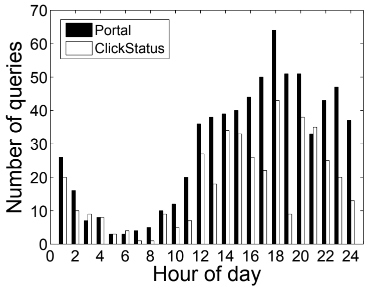
A new way to connect people. Almost all the participants find the idea of providing and viewing detailed information about people they are close to compelling, useful, and fun. In particular, location, activity/conversation, the historical log of the person's presence, random images, and social context are the features that people like the most. This pattern is confirmed in Figure 11(a), where the cumulative participants' feature utilization for different hours of the day derived from the analysis of system logs on the backend is shown. It is evident that location information which reveals where friends are is the feature most used by the



(a) Utilization distribution across the different features.



(b) Comparison between the number of random pictures inserted into the database versus the number of pictures deleted.



(c) Comparison between the CenceMe portal and ClickStatus usage.

Figure 11

participants. The random photos was also found to be of interest because it can be used as a way to tag the person's day as in a diary: "oh yeah... that chair... I was in classroom 112 at 2PM". The photos are often blurred, since they are taken outside the control of the person, but they still serve the diary tagging purpose. Some of the participants did not particularly like the fact that the system takes pictures outside their control, so they opted to turn that feature off by customizing their privacy policy on the phone.

What is the potential CenceMe demographic? We believe that people-centric sensing applications such as CenceMe could become popular among social networking application users, for whom sharing context and information is popular. For many of these users, privacy is less of a concern than for others, as shown by their interest in publicly publishing personal history in detail in blogs and on social networks. This tendency is also highlighted in Figure 11(b) which shows a comparison between the cumulative number of random photos inserted into the database versus the total number of photos deleted for different hours of the day. Once photos are uploaded users are given the opportunity to selectively delete them from the system. Few participants (4 out of 22) disabled the random photo for the entire duration of the experiment and others disabled it at different times of the day to meet their privacy needs or the needs of the people around them. In general, as shown in Figure 11(b), the number of non-deleted photos available for sharing is much greater than the number of deleted photos. Most participants did not mind having pictures taken at any time of the day and in random settings and then being shared with all the other participants. Many of them were excited by the idea of guessing what their friends were doing through the hint provided by random photos. Moreover, no CenceMe presence sharing restriction was applied by the participants, who allowed their sensing presence to be accessible by everyone in the group. Although some users stated that they could foresee wanting to apply a presence sharing restriction policy under certain conditions (e.g., if their parents had access), they felt comfortable with the idea of others seeing their presence most of the time.

Learn about yourself and your friends. *"CenceMe made me realize I'm lazier than I thought and encouraged me to exercise a bit more".* This quote is taken from one participant's survey. Other users expressed similar thoughts. Users view CenceMe as an application that potentially could tell them things that might be intuitively obvious, but are often invisible in their lives due to familiarity and repetition. Some examples are lack of physical activity and spending a lot of time in front of a computer. Near-real time presence sharing and historical presence representation are ways to capture peoples' lifestyle and trends about activity, social context (am I often alone? do I party too much?), and location.

My friends always with me. The study highlights that the participants enjoyed retrieving their friends' presence on the mobile phone with ClickStatus in addition to checking the portal. The average number of times per day they checked presence was 4 ± 3 times, where 3 is the standard deviation. Figure 11(c) shows a comparison between the total number of times presence is accessed through the portal or via ClickStatus distributed throughout the day. Although the number of times the participants access the portal is larger than their use of ClickStatus on the N95, ClickStatus is actively used. This is clear from Figure 11(c), where the use of ClickStatus rises during the time of day when people are presumably most likely on the move because they are going to class (between noon and 6PM) or hanging out with friends (between 8PM and 11PM).

Overall, the user experience is positive. Because many of them enjoyed using CenceMe, they kept the CenceMe phone for a while after the end of the experiment. We are currently working on revising some of the components and improving a few architectural elements in order to reflect some of the valuable feedback from the participants. Specifically, future revisions of the CenceMe system will include:

- An improved CenceMe software module on the phone that prolongs the battery life. Our goal is to achieve a 48 hour duration without recharging the device.

- An enhanced version of the portal to provide finer grained privacy policy settings as well as an enhanced ClickStatus user interface to provide the user with more powerful ways to browse their friend's presence.

- A shorter classification time for primitives and facts because many of the participants believe that real time access to buddies' sensing presence should be one of the features of the system. System architectural revisions are currently under consideration to meet this requirement. A burst mode for sensing may prove to be useful.

7. RELATED WORK

There is growing interest in the use of sensor-enabled mobile phones for people-centric sensing [20][21][15][26][29]. A number of diverse applications are emerging. In [5], the authors describe an application that determines pollution exposure indexes for people carrying mobile devices. A micro-blogging service is discussed in [8] that uses mobile devices to record multimedia content in-situ and shares this content in a real-time. In [9], we discuss the integration of the CenceMe application with Second Life [10]. The use of personal sensor streams in virtual worlds is a new and interesting area of research. The work presented in this paper significantly extends our initial work on CenceMe [4], where we discussed the basic idea and the results of some isolated experiments.

Cellphones have been used to learn about social connections [17][18] and provide context-aware communications using location information from cellular towers and manually configured preferences in the iCAMS system [11]. The iCAMS system allows users to pick the preferred method of communication according to a person's status and location (e.g., in person, email, home/work phone). WatchMe [13] is a similar system that aims at choosing the best way to communicate with buddies. WatchMe relies on GPS trace analysis to determine whether a person is walking or driving, and uses the phone's microphone to infer talking and silent states. CenceMe differs from iCAMS and WatchMe because of the rich context it provides about a person in an automated and transparent way. In the same way CenceMe also differs from Twitter [19], an application to publish text-based status messages generated by users.

There is a large body of work on activity inference and modeling using customized sensors worn by people [35][7] [36][6][37]. CenceMe differs from this work because it implements the activity inference algorithms on commercial mobile phones. As discussed in this paper there are a number of important design tradeoffs that need to be taken into account when implementing always-on people-centric sensing applications like CenceMe on off-the-shelf mobile phones. Systems such as SATIRE [16] also assume sensing devices with great capabilities being embedded into "smart clothing". An interactive dancing project [30] requires people to wear customized sensors mounted on shoes to track dancing activity. In [7] the authors discuss their experience building efficient classification techniques on the Intel Mobile Sensing Platform (MSP), a small form factor wearable device for embedded activity recognition. The MSP platform is quite powerful compared to many cellular devices. The CenceMe classifiers have been tailored to operate on less capable devices than the MSP while remaining effective.

8. CONCLUSION

We presented the implementation, evaluation, and user experiences of the CenceMe application, which represents one of the first applications to automatically retrieve and publish sensing presence to social networks using Nokia N95 mobile phones. We described a full system implementation of CenceMe with its performance evaluation. We discussed a number of important design decisions needed to resolve various limitations that are present when trying to deploy an always-on sensing application on a commercial mobile phone. We also presented the results from a long-lived experiment where CenceMe was used by 22 users for a three week period. We discussed the user study and lessons learnt from the deployment of the application and highlighted how we could improve the application moving forward.

Acknowledgments

This work is supported in part by Intel Corp., Nokia, NSF NCS-0631289, and the Institute for Security Technology Studies (ISTS) at Dartmouth College. We would like to thank Peter Boda and Chieh-Yih Wan for their support with this work. ISTS support is provided by the U.S. Department of Homeland Security under award 2006-CS-001-000001, and by award 60NANB6D6130 from the U.S. Department of Commerce. The views and conclusions contained in this document are those of the authors and should not be interpreted as necessarily representing the official policies, either expressed or implied, of any funding body. A special thanks goes to our shepherd Mark Corner.

9. REFERENCES

[1] Apple iPhone SDK. `http://developer.apple.com/iphone/`.

[2] Android. `http://code.google.com/android/`.

[3] A. T. Campbell, S. B. Eisenman, N. D. Lane, E. Miluzzo, R. A. Peterson, H. Lu, X. Zheng, M. Musolesi, K. Fodor, G. Ahn. The Rise of People-Centric Sensing, In *IEEE Internet Computing*, vol. 12, no. 4, pp. 12-21, Jul/Aug, 2008.

[4] E. Miluzzo, N. D. Lane, S. B. Eisenman, A. T. Campbell. CenceMe - Injecting Sensing Presence into Social Networking Applications In *Proc. of EuroSSC '07*, Lake District, UK, October 23-25, 2007.

[5] D. Estrin, et al. Seeing Our Signals: Combining location traces and web-based models for personal discovery. In *Proc. of HotMobile '08*, Napa Valley, CA, USA, Feb. 25-26, 2008.

[6] P. Zappi, et al. Activity Recognition from On-Body Sensors: Accuracy-Power Trade-Off by Dynamic Sensor Selection, In *Proc. of EWSN '08*, Bologna, Italy, Jan. 30/31 - Feb.1, 2008.

[7] T. Choudhury, et al. The Mobile Sensing Platform: an Embedded System for Capturing and Recognizing Human Activities, In *IEEE Pervasive Magazine, Spec. Issue on Activity-Based Computing*, April-June 2008.

[8] S. Gaonkar, et al. Micro-Blog: Sharing and Querying Content Through Mobile Phones and Social Participation, In *Proc. of MobiSys '08*, Breckenridge, CO, USA, June 17-20, 2008.

[9] M. Musolesi, E. Miluzzo, N. D. Lane, S. B. Eisenman, T. Choudhury, A. T. Campbell,. The Second Life of a Sensor: Integrating Real-world Experience in Virtual Worlds using Mobile Phones, In *Proc. of HotEmNets '08*, Charlottesville, Virginia, USA, June 2008.

[10] Second Life. `http://secondlife.com/`

[11] Y. Nakanishi, K. Takahashi, T. Tsuji, and K. Hakozaki. iCAMS: A Mobile Communication Tool Using Location and Schedule Information. In *IEEE Pervasive Computing*, Vol 3, Iss 1, pp. 82–88, Jan-Mar 2004.

[12] A. T. Campbell, et al. Transforming the Social Networking Experience with Sensing Presence from Mobile Phones. (Demo abstract) In *Proc. of ACM SenSys '08*, November 5-7, 2008, Raleigh, North Carolina, USA.

[13] N. Marmasse, C. Schmandt, and D. Spectre. WatchMe: Communication and Awareness Between Members of a Closely-knit Group. In *Proc. of 6th Int'l Conf. on Ubiq. Comp.*, pp. 214-231, Nottingham, Sep 2004.

[14] J. Lester, T. Choudhury, G. Borriello. A Practical Approach to Recognizing Physical Activities. In *Proc. of 4th Int'l Conf. on Perv. Comp.*, pp. 1-16, Dublin, May 2006.

[15] T. Abdelzaher, et al. Mobiscopes for Human Spaces. In *IEEE Perv. Comp.*, vol. 6, no. 2, pp. 20-29, Apr-Jun, 2007.

[16] R. Ganti, P. Jayachandran, T. Abdelzaher, J. Stankovic. SATIRE: A Software Architecture for Smart AtTIRE. In *Proc. of 4th Int'l Conf. on Mobile Systems, Applications, and Services*, Uppsala, Jun 2006.

[17] N. Eagle, A. Pentland. Eigenbehaviors: Identifying Structure in Routine. *Behavioral Ecology and Sociobiology (in submission)*, 2007.

[18] N. Eagle, A. Pentland. Reality Mining: Sensing Complex Social Systems. In *Personal Ubiq. Comp.*, pp. 255-268, May, 2006.

[19] Twitter. `http://twitter.com`

[20] J. Burke, et al. Participatory sensing. In *ACM Sensys World Sensor Web Workshop*, Boulder, CO, USA, Oct 2006.

[21] Sensorplanet. `http://www.sensorplanet.org/`

[22] D. J. Patterson et al. Inferring High-Level Behavior from Low-Level Sensors, In *Proc. of Ubicomp '03*, Seattle, USA, Oct. 2003.

[23] P. Mohan, V. N. Padmanabhan, and R. Ramjee, Nericell: Rich Monitoring of Road and Traffic Conditions using Mobile Smartphones, In *Proc. of ACM SenSys '08*, Raleigh, NC, USA, Nov 2008.

[24] N. D. Lane, H. Lu, S. B. Eisenman, A. T. Campbell. Cooperative Techniques Supporting Sensor-based People-centric Inferencing, In *Proc. of the Sixth International Conference on Pervasive Computing*, Sydney, Australia, May 2008.

[25] D. Ashbrook, T. Starner. Using GPS to learn significant locations and predict movement across multiple users, In Personal and Ubiquitous Computing Journal, vol. 7, no. 5, pp. 275-286, 2003.

[26] A. Kansal, M. Goraczko, and F. Zhao. Building a Sensor Network of Mobile Phones. In *Proc of IEEE 6th Int'l IPSN Conf.*, Cambridge, MA,USA, Apr 2007.

[27] Wikimapia. `http://wikimapia.org`.

[28] I. H. Witten and E. Frank. *Data Mining: Practical machine learning tools and techniques, 2nd Edition*, Morgan Kaufmann, San Francisco, 2005.

[29] A. T. Campbell, et al. People-Centric Urban Sensing (Invited Paper). In *Proc. of WICON '06*, Boston, MA, USA, Aug 2006.

[30] J. Paradiso, K.-Y. Hsiao, E. Hu Interactive Music for Instrumented Dancing Shoes In *Proc. of ICMC '99*, Bejing, China, Oct. 1999.

[31] C. Zhou, et al. Discovering Personally Meaningful Places: An Interactive Clustering Approach. In *ACM Trans. on Information Systems*, vol. 25, num 3, 2007.

[32] E. M. Tapia, et al., Real-time recognition of physical activities and their intensities using wireless accelerometers and a heart rate monitor, In Proc. of *International Symposium on Wearable Computers*, IEEE Press, 2007.

[33] R. O. Duda, P. E. Hart, D. G. Sork, *Pattern Classification*, Second Edition, Wiley Interscience.

[34] L. Bao, S. S. Intille. Activity Recognition from User-Annotated Acceleration Data, In *2nd International Conference, PERVASIVE '04*, Vienna, Austria, April 21-23, 2004.

[35] E. Welbourne, J. Lester, A. LaMarca and G. Borriello. Mobile Context Inference Using Low-Cost Sensors. In *Proc. of LoCA 2005*, Germany, May 2005.

[36] J. Lester, et al. Sensing and Modeling Activities to Support Physical Fitness. In *Proc. of Ubicomp Workshop: Monitoring, Measuring, and Motivating Exercise: Ubiquitous Computing to Support Fitness)*, Tokyo, Japan, Sep 2005.

[37] R. DeVaul, M. Sung, J. Gips and A. Pentland. MIThril 2003: Applications and Architecture. In *Proc. of 7th Int'l Symp. on Wearable Computers*, White Plains, Oct 2003.