PSense: Reducing Energy Consumption in Public Sensing Systems

Patrick Baier, Frank Dürr, Kurt Rothermel
Institute of Parallel and Distributed Systems
Universität Stuttgart
70569 Stuttgart, Germany
Email: {baier, duerr, rothermel}@ipvs.uni-stuttgart.de

***Abstract*—Utilizing peoples' mobile devices for gathering sensor data has attracted a lot of attention within the last few years. As a result, a great variety of systems for sensing environmental phenomena like temperature or noise have been proposed. However, most of these systems do not take into account that mobile devices have only limited energy resources. For instance, an often assumed prerequisite is that mobile devices are always aware of their position. Given the fact that a position fix is a very energy consuming operation, continuous positioning would quickly drain a device's battery. Since the owners of the mobile devices will not tolerate a significant reduction of the devices' battery lifetime, such an approach is not suitable. To address this issue we present PSense, a flexible system for efficiently gathering sensor data with mobile devices. By avoiding unnecessary position fixes, PSense reduces the energy consumption of mobile devices by up to 70% compared to existing mobile sensing approaches. This is achieved by introducing an adaptive positioning mechanism and by utilizing energy efficient short-range communication to exchange position related information.**

***Keywords*-Wireless sensor networks, Mobile Computing, Energy-aware systems**

I. Introduction

The ongoing advancements in the performance and the versatility of mobile devices, especially mobile phones, pave the way for new applications that go far beyond the primary scope of mobile communication. One particular field that already attracted a lot of attention in the research community (e.g. [1], [2]) is the opportunistically gathering of sensor data with mobile devices, also called *Public Sensing* [3], [4]. By utilizing the build-in sensing facilities of peoples' mobile devices, Public Sensing enables a cheap way of acquiring sensor data without the need for a costly sensor deployment.

Within the last few years a lot of research was done that deals with the challenges and the opportunities of Public Sensing [2]. To show its general feasibility a variety of applications were proposed that monitor particular environmental phenomena such as noise mapping [5] or traffic delay estimation [6], for instance. Besides these specialized systems, more generic views on Public Sensing were developed proposing architectural schemes for a scalable deployment of such sensing systems (e.g. [7], [8]). Although the aforementioned work focuses on the system design on a very abstract level, we argue that it is crucial to also take into account the algorithmic challenges that arise from Public Sensing when designing such a system. More precisely, as stressed in [4], most of the prevailing Public Sensing systems assume that the contributing mobile devices continuously fix their position and sample their sensors. Since it is commonly agreed that such a continuous sensing quickly drains a mobile device's battery (e.g. [2], [9]), most of these systems would not be accepted by the device's owner. As a result, the number of devices that gather sensor data would decrease. Because this would also decrease the quality of the information that the system delivers, Public Sensing systems must be designed in such a way that the amount of energy intensive operations on the mobile devices are minimized.

One of the major opportunities to reduce the devices' energy consumption is to avoid the continuous positioning of the devices [9], which is a frequently assumed prerequisite in most of the prevailing Public Sensing systems. Often GPS is used for positioning since it is widely deployed and sufficiently accurate. However, experimental measurements [10] have shown that continuously sampling the GPS sensor significantly decreases a mobile device's battery life. Circumventing this problem by simply reducing the positioning interval between two consecutive GPS fixes would have an undesired impact on the sensing result, since sensor readings are in general only useful when they are provided with the location where they were taken. Moreover, mobile devices need to be aware of their position to decide whether they should take a sensor reading at their current location.

In this paper, we present the PSense system, which reduces the energy for positioning in Public Sensing while not facing the aforementioned problems. PSense reduces the overall number of position fixes a mobile devices has to perform in a Public Sensing scenario, while not affecting the quality of the sensing process. This is achieved by adapting the time until a mobile device performs its next position fix to the distance of the nearest location at which the device could read sensor values. For instance, considering that a noise level measurement is requested at some location l in the city (referred to as *sensing location*). Naively, a mobile device m that has a microphone to record the noise level

Published in the Proceedings of the 26th IEEE International Conference on Advanced Information Networking and Applications (AINA-2012),pp. 136-143, Fukuoka, Japan, March 2012.

http://dx.doi.org/10.1109/AINA.2012.33

at l but is quite far away from it would continuously fix its position to determine whether it is close enough for recording. In a more deliberated approach, m can defer its next position fix until it is close enough for recording at l. PSense makes use of this fact by determining for each mobile device an optimal time span by which the device's next position fix can be deferred while ensuring that no sensing location is missed. We show by simulation that this reduces the energy consumption by up to 70% compared to the prevailing approaches that use continuous positioning. Moreover, PSense works on basis of a generic sensor abstraction that is utilized in many Public Sensing systems (e.g. [4], [11], [12]). This makes it deployable for a huge variety of mobile sensing scenarios.

The rest of the paper is organized as follows. In Section II we first give an overview of related work dealing with optimization approaches in Public Sensing in general and the problem of efficient positioning in particular. In Section III we introduce our system model, before we present in Section IV the basic sensor abstraction of our system. In Sections V to VII we first present a naive approach to acquire the sensor data and then consecutively introduce the optimization algorithms we have developed for PSense. In Section VIII we evaluate our systems and present the simulation results, before Section IX concludes this work.

II. Related Work

To introduce the related work, we have a look at two different research areas. First, we look at optimization techniques that aim on reducing the energy consumption of mobile devices in Public Sensing scenarios. Second, we particularly focus on work dealing with efficient positioning, which is the basic optimization concept of PSense.

A. Optimized Public Sensing

In addition to the already mentioned Public Sensing systems that do not consider energy consumption at all, there has also been some work that considers the challenge of gathering mobile sensor data in an efficient way. For instance, Eisenman et al. [13] show how the communication with mobile devices can be more efficient when adapting the size of the communication range of the communicating partners to application specific requirements. A quite different approach (e.g. [14], [15]) aims at the avoidance of cellular network communication in favor of short-range radio communication. Since short-range radio induces less energy overhead on the mobile phones [16], local forwarding and aggregation of data can be used to reduce mobile energy consumption and cellular network load. One further optimization goal that is focused on the sensing, is the prevention of redundant sensor readings. Given a predefined point-of-interest and a set of sensor equipped mobile devices in its proximity, Philipp et al. [4] show that the mobile devices can be coordinated in such a way that only a small subset of devices is needed to read the sensor data at that point in an effective and efficient way.

Although all of these approaches optimize energy consumption in Public Sensing, none of them stresses the fact of efficient positioning. Since efficient positioning in general is not a completely new research challenge, we have in the following a look on work that particular focuses this issue.

B. Efficient Positioning

To begin with, we look at work that deals with application independent techniques for efficient positioning. Taking the context of the user into account, Lu et al. [10] and Ryder et al. [17] show a way to increase the battery lifetime of a mobile device by adapting the GPS positioning to the movement mode of the user. For instance, no position fix is needed while the user stays at the same position. Although these approaches optimize positioning, they do it in a very generic way. Therefore, they cannot fully utilize the energy reduction that is possible in a dedicated sensing scenario.

In contrast, application aware approaches can utilize a more efficient positioning. One commonly used way to reduce the number of GPS fixes on a mobile device is to delay its next position fix until the earliest point in time when the device could reach a critical point for the application. For instance, Farrell et al. [18], [19] show in their work on continuous range queries that the next GPS fix can be deferred until the earliest point in time when the mobile device could reach the boundary of the range query. In our previous work [3], [11], [20] we employ this concept to coordinate the positioning of mobile devices in specialized sensing scenarios, like object discovery or map validation. More precisely, we use information about the Euclidean distance from a mobile device to the next point that is critical for the application. Upon that we calculate the time by which the next position fix of the device can be deferred.

For PSense we extend our previous work to the area of Public Sensing by including more generic assumptions that are not considered in our previous approaches. In more detail, PSense assumes an underlying road graph for the mobile devices' movement and uses graph-based distances instead of Euclidean to determine the time until the next position fix. Additionally, PSense handles frequently changing queries for sensor data. As a result, the set of critical points for positioning (in that case the set of sensing locations) is dynamically changing and a mechanism for the adaption of positioning is needed, which is not considered so far. Finally, PSense utilizes short-range communication to exchange information about a mobile phone's current position, which is not utilized by any of the existing approaches.

III. System Model

Our system model consists of one server and an arbitrary number of mobile devices (see Figure 1). The server provides a query interface for collecting and answering user

queries for sensor data, which is described in the next section in more detail. Note that for matters of abstraction we assume only one server. In a real deployment the load of this server may be distributed among a set of other servers, where each server would be responsible for a particular area. Each mobile device m is equipped with a GPS sensor for determining its position $(x_m(t), y_m(t))$ at time t. This position is inaccurate up to e_{pos} (usually up to 20 meters [21]). Furthermore, each device m comes with a set of specific sensors S_m such as a thermometer. Each sensor $s \in S_m$ has a limited reading range $range(s)$ and a type $type(s)$, which specifies the type of data that can be sensed.

Mobile devices are carried by people who move according to an underlying road graph, which is known to the server. The edges of this road graph are constituted of polygonal lines representing roads. These edges are interconnected with vertices that represent intersections and terminal points of roads. We assume that neither the speed nor the movement direction of the mobile devices can be influenced. Furthermore, we require that each mobile device is aware of its maximum speed, which is referred to as v_{max}.

The server can communicate with the mobile devices via a cellular mobile network like GPRS or UMTS. In addition, mobile devices can also use short-range radio (e.g. WLAN 802.11 or Bluetooth) to establish an inter-device ad-hoc communication that has a limited range $range(ah)$.

Since the server is part of the infrastructure, we assume that it comes with unlimited battery supply. On the contrary, a mobile devices has only limited battery supply and each sensing and communication operation drains a specific amount of energy from its battery. Moreover, we assume that cellular communication is more energy draining than short range communication [11]. For the concrete energy values that we assume for these operations we refer to Section VIII.

IV. Basic Sensor Abstraction

In this section, we introduce the basic sensor abstraction PSense is based upon. To achieve high flexibility and easy utilization of PSense, we adapt an abstraction model (see Figure 1) that is commonly used in a large variety of Public Sensing systems (e.g. [4], [11], [12]):

The server accepts sensing queries Q that contain a set of desired sensing locations. Each sensing location $l \in Q$ is specified by spatial coordinates (x_l, y_l) and a particular type of sensor $type(l)$ that indicates the kind of data which the user is interested in (e.g. the noise level). The server stores a set L, which contains all sensing locations currently requested by the users. For providing the user with the requested data, the server assigns sensing tasks to mobile devices that move within the vicinity of the queried sensing locations. This area is referred to as *queried area* and is defined as an extended bounding-box of all queried sensing locations. We call a sensing location l *compatible* with a mobile device m if it satisfies the following condition:

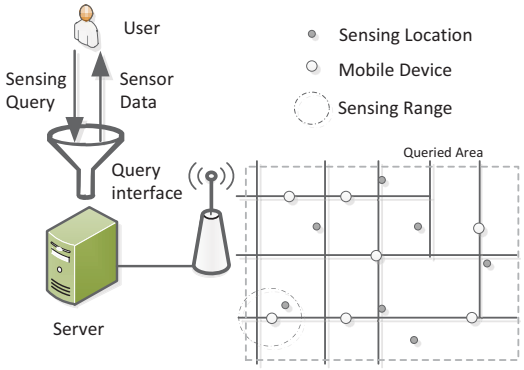

Figure 1. System Model and Sensor Abstraction

- $\exists s \in S_m : type(s) = type(l)$

Furthermore, a device can read a sensing location l at time t if the following conditions are satisfied:

1) l is compatible with m by sensor s
2) $dist((x_l, y_l), (x_m(t), y_m(t))) \leq range(s)$

Where $dist(x, y)$ depicts the Euclidean distance between point x and y. If a sensing location l was successfully read by a mobile device, the data is uploaded to the server. The server delivers the data to the user and removes l from L.

V. Naive Sensing Approach

In this section, we present a basic algorithm that realizes a naive way for providing the requested sensor data to the user. Since this approach leads to a very high energy consumption on the mobile devices, we gradually improve it in the upcoming sections to address this problem.

Upon receiving a sensing query Q, the server adds each $l \in Q$ to the set of currently requested sensing locations L. Initially, the server distributes L to all mobile devices that are located in the queried area. For sending this message, the server needs to know the set of devices that are currently located in this area. We assure this by assuming that a node registration mechanism runs on each mobile device, which notifies the server when a mobile device joins or leaves the queried area. Since this mechanism is out of the scope of PSense, we refer to dedicated work on energy efficient tracking of mobile objects (e.g. [19]). After the initial distribution of L, the server notifies the devices in the queried area every time L changes by sending an update containing only the newly inserted or removed sensing locations. Mobile devices that enter the queried area for the first time register at the server and receive the entire set L.

When a mobile device receives L from the server, it starts periodically updating its position every t_{pos} seconds via GPS. After each position fix the device checks whether the conditions for reading one (or more) sensing locations in L are fulfilled. If that is the case for l, the device reads l and uploads the data to the server. Upon receiving this

data, the server delivers it to the user and removes l from L. Subsequently, the server notifies all other mobile devices in the queried area that l no longer needs to be read.

VI. Adaptive Positioning

The naive approach presented in the previous section ensures that no mobile device passes a sensing location without noticing it (at least if t_{pos} is chosen sufficiently small) and can therefore be used as a reference for achieving optimal sensing effectiveness. Nevertheless, this approach causes a high energy consumption since the mobile devices continuously fix their position. To tackle this issue, PSense uses a mechanism that suppresses position fixes if the next sensing location is some distance away. The basic idea is to find an adaptive positioning interval of length t_a that determines the time until the next GPS fix is performed instead of fixedly scheduling the next position fix at time t_{pos}. Most beneficially, t_a is set to the maximum time span a device can suppress positioning without missing the next compatible sensing location. In the following, we first introduce a concept to calculate the adaptive positioning interval t_a. In the second step, we introduce a mechanism that ensures the effectiveness of sensing in case the set of queried sensing locations L changes, while some devices already scheduled their next position fix.

A. Adaptive Positioning Interval

For choosing an optimal positioning interval t_a, device m needs information about the distance to its closest compatible sensing location l. Knowing this distance (referred to as d_{min}), it can calculate t_a according to Equation (1) using information about its maximum speed v_{max}. The resulting value of t_a indicates the earliest point in time, when m could reach l. Since we use the maximum speed for this calculation, we ensure that no node will miss a sensor reading. A more optimistic approach would be to use the node's average speed v_{avg} instead. This would further decrease the number of GPS fixes but also reduce the effectiveness of the sensing system, since it would be possible that a sensor reading is missed in case a device moves faster then v_{avg}. Because the effectiveness of the system should be guaranteed with PSense, we use v_{max} for the calculation of t_a and defer analyses of a more optimistic approaches to our future work.

$$t_a = \frac{d_{min}}{v_{max}} \tag{1}$$

Since a device needs to know d_{min} for the calculation of t_a, it sends after each position fix its current position in a *positioning query* to the server, which is answered by the server with a *sensing update*. This sensing update includes d_{min} as well as the coordinates (x_l, y_l) and the $type(l)$ of the closest compatible sensing location l. With the help of these values the device first checks if it could read l immediately.

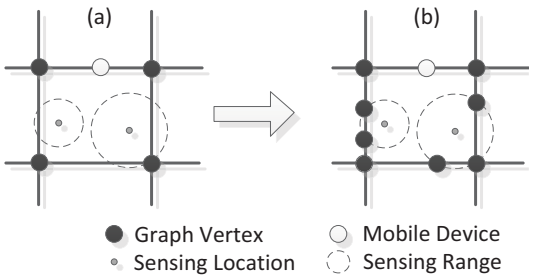

Figure 2. Restructuring the Road Graph

If that is the case it reads l and uploads the data to the server, which answers with a new sensing update (including another sensing location). If the device cannot sense l, it uses the provided d_{min} to calculate t_a and schedules its next position fix to time t_a. Although server communication is an energy draining operation, the objective of this approach is that the decrease in the number of position fixes outweighs the energy spent for sending positioning queries.

In more detail, the positioning query that mobile device m sends at time t to the server contains m's current position $(x_m(t), y_m(t))$ and m's set of sensors S_m. To accurately determine d_{min} the server utilizes the fact that the devices move according to the roads of the road graph. Therefore, instead of calculating the Euclidean distance between the device and the closest compatible sensing location it uses a shortest path algorithm (e.g. *Dijkstra* or *Bellmann-Ford*). Before this algorithm can be applied, the road graph needs to be restructured by mapping the position of the mobile device and the sensing locations to the road graph in the following way: After the server received a positioning query from mobile device m it maps m's position to the closest point on the road graph. We refer to this point as $pos(m)$. The server now checks for each $l \in L$ whether l is compatible with m. If l is compatible with m by sensor s, the server associates l with the sensing range $range(s)$ to define the area in which m can read l. This area is now projected onto the road graph by intersecting the graph and the circle with radius $range(s)$ and center (x_l, y_l) (see Figure 2-a). The obtained intersections mark the points on the road graph at which m could start reading l. If sensing location l is not compatible with m, l is skipped. Subsequently, each road edge that contains any of the intersections is split into two road edges and a connecting vertex is placed at the respective intersection point (see Figure 2-b).

To obtain d_{min}, the server now executes a shortest path algorithm on the modified road graph starting from $pos(m)$. Executing this algorithm, the server calculates the distance on the shortest path from $pos(m)$ to all vertices in the road graph. After the algorithm is finished, the server determines d_{min} that indicates the distance from $pos(m)$ to the closest vertex v that was generated out of a sensing location l.

Finally, the server subtracts e_{pos} from d_{min} and returns this value along with the coordinates and the type of sensing location l to the device. This is necessary to take the inaccuracy of the GPS fix into account that depicts the mobile devices position. This inaccuracy may cause that the device is actually located closer to v than distance d_{min}.

The reason why d_{min} is calculated on the server rather than on the mobile devices is twofold. First, executing a shortest path algorithm is a non-trivial operation and needs to be done quite often (after each position fix). Since the road graph might arbitrarily large and complex, the computational overhead on the mobile device could quickly get too energy consuming. Furthermore, this avoids the need to make any assumptions about the computational power of the mobile devices, which can be quite heterogeneous. The second reason for doing the calculation on the server is that the necessity to keep L up-to-date on all mobile devices is avoided. This is especially beneficial in scenarios in which the set of queried sensing locations changes frequently.

B. Query Adaption

Using the concept of adaptive positioning, the server does not proactively disseminate the set of current sensing locations to the mobile devices, as in the naive approach. Therefore, the individual positioning interval of some mobile devices may no longer be appropriate when a new sensing query arrives. For instance, assuming a user issues a query for data at time t from a sensing location l that is next to a mobile device m. If l is compatible with m, but m has already scheduled its next position fix to some time $t' > t$, it is possible that m misses to read l. To avoid this, the server has to recalculate the distance to the closest compatible sensing location by taking into account the newly queried sensing locations. If the result is smaller than m's formerly calculated d_{min}, the server has to notify the device. To know which mobile devices needs an update when a new query arrives, the server implements the following concept:

When calculating the distance to the closest compatible sensing location d_{min} for a mobile device m, the server remembers the device's last known position, its sensors and the calculated d_{min}. Upon receiving a new query Q the server checks for every device m if there are sensing locations $l \in Q$ that are compatible with m. If that is the case the server maps these sensing locations to the road graph and executes the shortest path algorithm from m's last known position again. If the resulting d'_{min} is smaller than the previously calculated distance d_{min} (see Figure 3), the server queries device m for its position. Knowing this position the server returns a fresh sensing update by taking into account the newly inserted sensing location. This mechanism guarantees that no mobile device misses a sensing location and only those devices receive an update that are affected by newly queried sensing locations.

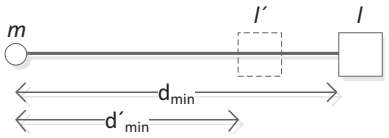

Figure 3. Arrival of new Sensing Location

Moreover, we also have to consider the case that a sensing location is successfully read by some device and gets removed from L after the results were uploaded to the server. In the naive approach the server has to inform all devices about the removed sensing location to avoid redundant sensing operations. Since the server now only sends the coordinates of the closest compatible sensing location to each device, it only has to send a sensing update to those devices that have the successfully read sensing location as their closest compatible sensing location.

VII. Ad-hoc Information Exchange

Although adaptive positioning decreases the amount of GPS fixes on the mobile devices, it increases the number of messages that are exchanged with the server, since every mobile device has to query the server after each position fix. Therefore, PSense utilizes a technique that further reduces the energy consumption by decreasing the number of messages that the mobile devices exchange with the server. This can be achieved by making use of the property that nearby mobile devices have almost the same distance to the next sensing location. If a mobile device receives a sensing update from the server it can locally forward this information via its ad-hoc interface. Nearby mobile devices can now use this information to adapt their respective positioning interval without communicating with the server. To implement this concept, the adaptive positioning algorithm is augmented by two extensions presented in the following. The first shows how the message exchange between the mobile devices is realized. The second step ensures that the introduced query adaption also works with the ad-hoc extension. This is important since the server may no longer be aware of a device's closest sensing location in case a device learns about a nearby sensing locations via an ad-hoc message.

A. Message Exchange

To introduce the message exchange, we assume that a mobile device m_s just fixed its position and received a sensing update from the server that includes sensing location l. Before m_s forwards the information about l via an ad-hoc broadcast to other nearby devices, it determines the Euclidean distance from its current position to l (referred to as $d(l)$) and then subtracts the maximum range of its ad-hoc interface $range(ah)$ from $d(l)$. The resulting value indicates the minimum distance to l for each receiver of the broadcast, since each of them maybe located up to $range(ah)$ closer to

l (see Figure 4-a). Finally, m_s sends the ad-hoc broadcast and also includes the type of the sensing location $type(l)$ and the set of its sensors S_s.

Each device m_r that receives this message from the sending device m_s does the following:

1) Check if l is compatible with m_r. If not, stop.
2) Check if $S_r \subseteq S_s$. If not, stop.
3) Compute t'_a according to the following equation:

$$t'_a = \frac{d(l) - range(s)}{v_{max}} = \frac{dist}{v_{max}}$$

   In which $range(s)$ depicts the range of sensor $s \in S_r$ that can read l (i.e. $type(s) = type(l)$).
4) If t'_a is bigger than the remaining time until the next position fix is performed, then reschedule the next position fix to t'_a. If not, stop.
5) Send an *ACK*-message back to m_r.

Besides the prerequisite that the sensing location l must be compatible with m_r (step 1), the device m_r can only use the information it received if l is also its closest compatible sensing location. To guarantee this, the broadcast sender m_s needs to have at least all the sensors that m_r has (step 2). If that is not the case, a sensor $s \in S_r$ and $s \notin S_s$ can be located closer to m_r but is not included in the sensing update from the server (since this is based on the sensors of m_s). If these requirements are fulfilled, m_r can determine time t'_a that indicates the earliest time in which m_r can reach the starting point for reading location l. To get the distance to this starting point, $range(s)$ needs to be subtracted from the received $d(l)$ (see Figure 4-a). Device m_r can now extend the time until the next GPS fix to time t'_a, if the remaining time until the next position fix is smaller than t'_a (step 4). If that is the case device m_r answers with an ACK-message to the broadcast sender (step 5), which is needed for the ad-hoc query adaption mechanism (see next section).

Although device m_s receives with the sensing update from the server also the graph-based distance d_{min} to the next sensing location l, it broadcasts the Euclidean distance to the other devices. Figure 4-b shows that broadcasting the graph-based distance may lead to a problem if the receiving device is on another road segment from which l can be reached on a shorter path. If m_r would calculate t'_a based on the graph-based distance d_{min} it received from m_s, it would be possible that t'_a is chosen too large and the reading of l is missed. Since this has to be avoided to preserve the effectiveness of the sensing system, the calculation of t'_a is based on the Euclidean distance to l.

B. Ad-hoc Query Adaption

To ensure a correct query adaption in case of the arrival of a new sensing location, the previously presented query adaption has to be extended when ad-hoc information exchange is used. More precisely, if a new sensing location is queried, the server needs to know the closest compatible sensing location of each device to execute the query adaption (see section VI-B). The problem that arises with the ad-hoc extension is that the server does not take notice of the ad-hoc messages that the devices exchange and which may update a device's closest compatible sensing location. As a result, it cannot determine which devices need an update if a query Q arrives. To avoid this, the query adaption is extended as follows. If a mobile device m_s sends an ad-hoc message it remembers the set of other devices that used the sent information for extending their respective positioning interval (i.e. all devices that returned an *ACK*-message). We refer to this set as M_o. If the server now receives a new sensing location which causes an update for m_s (according to query adaption in section VI-B), device m_s returns the set M_o to the server. The server now knows the set of devices that scheduled their next position fix according to the same sensing location as m_s. Since these devices also need an update, the server subsequently queries each device $m \in M_o$ for its position and returns a fresh sensing update. Based on this update each of theses devices adapts now its positioning interval (as described in Section VI).

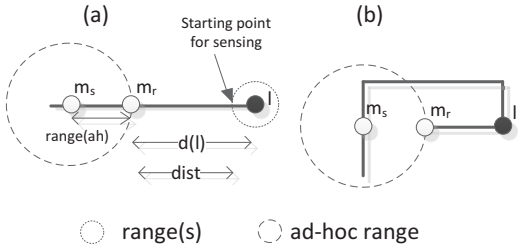

Figure 4. Ad-hoc Information Exchange

VIII. Evaluation

To evaluate PSense, we look at its efficiency and effectiveness by comparing the developed concepts to the naive approach. For this purpose, we conducted extensive simulations using the network simulator ns-2 combined with traces of pedestrian movement.

A. Simulation Setup

For the simulation we took a part of the street graph of Chicago (size 2 km x 2 km) as road graph and used the mobility trace file generator UDELModels [22] to generate realistic trace files of pedestrian movement. We simulated one hour simulation time using varying numbers of mobile devices and a standard positioning interval $t_{pos} = 5\,\text{s}$ (for the naive approach). To simulate the sensing queries we requested data from ten randomly placed sensing locations at the same time. When a sensing location was successfully read we removed it and inserted a new sensing location at some other randomly chosen place again.

To measure the amount of energy that is consumed for communication we used the commonly used energy model

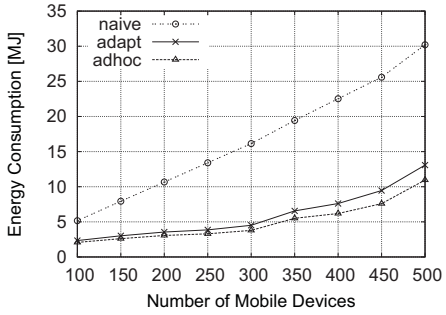

Figure 5. Total Energy consumption

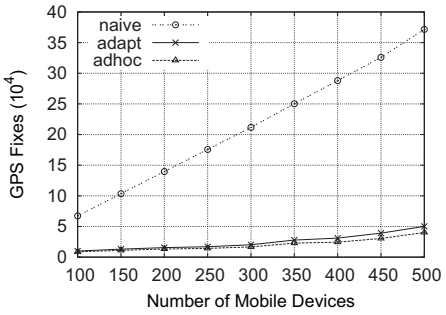

Figure 6. Positioning Operations

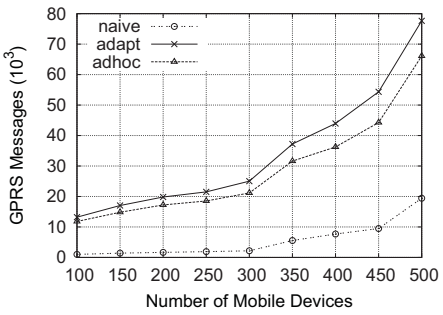

Figure 7. GPRS Communication

[3], [23] shown in Table I. For the communication between the mobile devices and the server we assumed GPRS. For the ad-hoc information exchange we assumed Wifi communication using the 802.11b standard and a maximum communication range $range(ah)$ of 100 m. As the algorithms executed on the mobile phones are quite simple and can be executed in constant time, we do not explicitly consider the energy consumption for their execution.

Operation	Energy [mJ]
GPS Position Fix	75
GPRS Send (1000 Bit)	80
GPRS Receive (1000 Bit)	40
802.11b Send (1000 Bit)	2
802.11b Receive (1000 Bit)	1

Table I
ENERGY MODEL

In the following we compare the results of the three different algorithms we presented:

- naive Refers to the results of the naive approach without including any optimizations (see Section V).
- adapt Includes the concept of the adaptive positioning interval (see Section VI).
- adhoc Includes adaptive positioning as well as the ad-hoc information exchange (see Section VII).

B. Efficiency of PSense

First, we have a look at the amount of energy that is consumed by the mobile devices to evaluate the efficiency of PSense. Figure 5 shows on the y-axis the total amount of energy that is consumed by all mobile devices. The values on the x-axis depict the number of mobile devices that were used for the respective simulation scenario. We can see that the amount of energy consumed in the ad-hoc information exchange approach is on average only 30% of the naive approach (i.e. a reduction of 70%) and that ad-hoc consumes less energy than using only adaptive positioning.

Next we investigate how these energy values are constituted. Therefore, we look at the number of GPS fixes (Figure 6) and the number of GPRS messages (Figure 7) that are used over all mobile devices. Here we can see that the naive approach requires much more GPS fixes than the other approaches. Looking at the number of GPRS messages, we see that the ad-hoc information exchange leads to a significant decrease of GPRS messages. Since cellular communication and GPS fixes are very energy consuming, we can explain from Figure 6 and 7 why the ad-hoc approach performs best in terms of overall energy consumption.

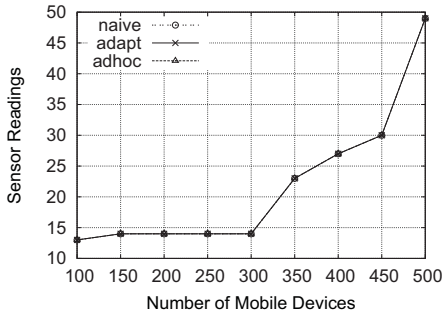

Figure 8. Sensing Effectiveness

C. Effectiveness of PSense

To determine the effectiveness of PSense, we look at the number of sensor readings that were performed in each of the three approaches. Figure 8 shows the number of sensor values that are uploaded from the mobile devices to the server. We can see from the chart that this number is the same for all approaches. As a result, we can conclude that no sensor readings are missed and all approaches deliver the same amount of sensor readings to the user.

IX. CONCLUSION

In this paper we introduced the PSense systems, a generic Public Sensing framework for efficiently gathering sensor data with mobile devices. In comparison to existing approaches using continuous positioning, PSense introduces an adaptive positioning mechanism. As shown in the evaluation, this approach helps to significantly reduce the number of required GPS fixes but does not impact the amount of acquired

sensor data. Nevertheless, the adaptive sensing approach also increases the amount of cellular messages. Therefore, we introduced in a second step an ad-hoc exchange mechanism that replaces energy intensive cellular messages to further decrease the amount of energy consumed. Finally, PSense is based on a widely used sensor abstraction model that makes it applicable in a wide spectrum of Public Sensing scenarios.

In future work we will investigate further positioning strategies that are based on a more optimistic assumption of the nodes prospective speed. This will further reduce energy consumption but may also have a negative impact on the overall effectiveness of the sensing system.

REFERENCES

[1] T. Abdelzaher, Y. Anokwa, P. Boda, J. Burke, D. Estrin, L. Guibas, A. Kansal, S. Madden, and J. Reich, "Mobiscopes for human spaces," *IEEE Pervasive Computing*, vol. 6, pp. 20–29, Apr. 2007.

[2] N. Lane, E. Miluzzo, H. Lu, D. Peebles, T. Choudhury, and A. Campbell, "A survey of mobile phone sensing," *Communications Magazine, IEEE*, vol. 48, no. 9, pp. 140–150, Sep. 2010.

[3] P. Baier, H. Weinschrott, F. Dürr, and K. Rothermel, "Map-Correct: automatic correction and validation of road maps using public sensing," in *Proc. of the 36th Conf. on Local Computer Networks*, 2011.

[4] D. Philipp, F. Dürr, and K. Rothermel, "A sensor network abstraction for flexible public sensing systems," in *Proc. of the 8th Intern. Conf. on Mobile Ad-hoc and Sensor Systems*, 2011.

[5] R. K. Rana, C. T. Chou, S. S. Kanhere, N. Bulusu, and W. Hu, "Ear-phone: an end-to-end participatory urban noise mapping system," in *Proc. of the 9th ACM/IEEE Intern. Conf. on Information Processing in Sensor Networks*, 2010.

[6] A. Thiagarajan, L. S. Ravindranath, K. LaCurts, S. Toledo, J. Eriksson, S. Madden, and H. Balakrishnan, "VTrack: Accurate, Energy-Aware Traffic Delay Estimation Using Mobile Phones," in *Proce. of the 7th ACM Conf. on Embedded Networked Sensor Systems*, 2009.

[7] T. Das, P. Mohan, V. N. Padmanabhan, R. Ramjee, and A. Sharma, "Prism: platform for remote sensing using smartphones," in *Proc. of the 8th Intern. Conf. on Mobile Systems, Applications, and Services*, 2010.

[8] A. T. Campbell, S. B. Eisenman, N. D. Lane, E. Miluzzo, and R. A. Peterson, "People-centric urban sensing," in *Proc. of the 2nd Intern. Workshop on Wireless internet*, 2006.

[9] I. Shafer and M. L. Chang, "Movement detection for power-efficient smartphone wlan localization," in *Proce. of the 13th ACM Intern. Conf. on Modeling, Analysis, and Simulation of Wireless and Mobile Systems*, 2010.

[10] H. Lu, J. Yang, Z. Liu, N. D. Lane, T. Choudhury, and A. T. Campbell, "The jigsaw continuous sensing engine for mobile phone applications," in *Proc. of the 8th ACM Conf. on Embedded Networked Sensor Systems*, 2010.

[11] H. Weinschrott, F. Durr, and K. Rothermel, "Efficient capturing of environmental data with mobile rfid readers," in *Proc. of the 10th Intern. Conf. on Mobile Data Management: Systems, Services and Middleware*, 2009.

[12] H. Lu, N. D. Lane, S. B. Eisenman, and A. T. Campbell, "Bubble-sensing: Binding sensing tasks to the physical world," in *Proc. of the Intern. Workshop on Mobile Device and Urban Sensing*, 2008.

[13] S. B. Eisenman, H. Lu, and A. T. Campbell, "Halo: Managing node rendezvous in opportunistic sensor networks," in *Proc. of the 6th IEEE Intern. Conf. Distributed Computing in Sensor Systems*, 2010.

[14] E. Jung, Y. Wang, I. Prilepov, F. Maker, X. Liu, and V. Akella, "User-profile-driven collaborative bandwidth sharing on mobile phones," in *Proc. of the 1st Workshop on Mobile Cloud Computing & Services*, 2010.

[15] B. Han, P. Hui, V. A. Kumar, M. V. Marathe, G. Pei, and A. Srinivasan, "Cellular traffic offloading through opportunistic communications: a case study," in *Proc. of the 5th ACM Workshop on Challenged Networks*, 2010.

[16] N. Balasubramanian, A. Balasubramanian, and A. Venkataramani, "Energy consumption in mobile phones: a measurement study and implications for network applications," in *Proc. of the 9th Conf. on Internet Measurement*, 2009.

[17] J. Ryder, B. Longstaff, S. Reddy, and D. Estrin, "Ambulation: A tool for monitoring mobility patterns over time using mobile phones," in *Intern. Conf. on Computational Science and Engineering*, 2009.

[18] T. Farrell, K. Rothermel, and R. Cheng, "Processing continuous range queries with spatiotemporal tolerance," *IEEE Transactions on Mobile Computing*, vol. 10, no. 3, pp. 320–334, Mar. 2011.

[19] T. Farrell, R. Lange, and K. Rothermel, "Energy-efficient Tracking of Mobile Objects with Early Distance-based Reporting," in *Proc. of the 4th Intern. Conf. on Mobile and Ubiquitous Systems: Networking and Services*, 2007.

[20] H. Weinschrott, J. Weisser, F. Durr, and K. Rothermel, "Participatory sensing algorithms for mobile object discovery in urban areas," in *Proc. of the Intern. Conf. on Pervasive Computing and Communications*, march 2011, pp. 128 –135.

[21] W. Ochieng, "Urban road transport navigation: performance of the global positioning system after selective availability," *Transportation Research Part C-emerging Technologies*, vol. 10, pp. 171–187, 2002.

[22] J. Kim, V. Sridhara, and S. Bohacek, "Realistic mobility simulation of urban mesh networks," *Ad Hoc Netw.*, vol. 7, pp. 411–430, Mar. 2009.

[23] H. Weinschrott, F. Duerr, and K. Rothermel, "Streamshaper: Coordination algorithms for participatory mobile urban sensing," in *Proc. of the 7th Intern. Conf. on Mobile Adhoc and Sensor Systems*, 2010.