Concurrent scheduling in the Active Bat location system

Oliver J. Woodman and Robert K. Harle
Computer Laboratory
University of Cambridge
Cambridge
{*ojw28,rkh23*}*@cam.ac.uk*

***Abstract*—This paper looks at the scalability problems inherent in the Active Bat system: an outside-in ultrasonic location system. Such systems are typically associated with higher positioning accuracies and longer tag battery lifetimes relative to comparable inside-out systems. However, they are often criticised for a lack of scalability since multiple tags must be addressed in series to prevent interference within the positioning medium. Multiple radio zones have previously been used to address this problem. We show that this approach is sub-optimal both in terms of system complexity and in terms of the location update rates that can be achieved.**

We propose a general solution to the scalability problem based on the computation of dynamic bounding regions within which corresponding tags are almost certainly located. These regions are used to determine when tags are sufficiently well separated to be queried concurrently without the risk of interference. We test this approach using real data, concluding that bounding regions permit more concurrency than is possible using multiple radio zones whilst at the same time reducing the complexity of the system.

***Keywords*-location; outside-in; scalability;**

I. Introduction

The Active Bat system is an outside-in ultrasonic location system that tracks small tags (known as bats) in indoor environments [1]. A bat's position is queried over an out-of-band radio channel, which is also used by bats to notify the system of their presence (this is known as registration). When a bat receives a query it responds by emitting an ultrasound pulse that is detected by ceiling-mounted receivers. If the pulse is detected by three or more ceiling-mounted receivers then the position of the bat is calculated using multi-lateration.

One property that the Active Bat system (in its simplest form) has in common with many other outside-in location systems is that it does not make any assumptions about the position of a bat prior to a query being made. As a result the system cannot make any assumptions about which receivers might hear the bat's response and so all of the receivers must listen for the signal. The system must also ensure that any ultrasound emitted by a previously queried bat has died down before another query can be made, otherwise the system has no way of telling which of the bats emitted a pulse that is detected by a receiver. To ensure this the Active Bat system is implemented as a *slotted* system, where the slot period $s = 20\,\mathrm{ms}$ is the time taken for an ultrasound pulse emitted by a bat to fall to a level low enough such that it will not be detected by any of the receivers. The system's scheduler queries one bat at the start of each slot and listens for the response using all of the receivers until the end of that slot. The main problem with this approach is that the average rate at which each bat can be queried is inversely proportional to the number of bats being tracked by the system. If 10 bats are being tracked then the average frequency at which each bat is queried is $5\,\mathrm{Hz}$. If 50 bats are being tracked then this falls to $1\,\mathrm{Hz}$. Hence the Active Bat system as described above does not scale well to support large numbers of tags. We call this the *query-rate* problem.

One solution to the query-rate problem is to deploy an inside-out system in which tags calculate their own positions based on transmissions received from static beacons installed in the environment [2]. Such systems allow an arbitrarily large number of tags to position themselves simultaneously based on the same beacon transmissions. Unfortunately such systems typically have lower accuracies and require more advanced algorithms to determine a tag's position [3]. Since tags must calculate their own positions in an inside-out system their battery lives are also shorter than those used in outside-in systems.

Another solution to the query-rate problem is to use broadband rather than narrowband signals. The use of broadband signals makes it possible for multiple tags to transmit concurrently in a way such that their individual signals can still be decoded [4], [5]. Unfortunately the hardware required is more expensive and typically consumes more power, shortening the battery life of a tag. Hence it is desirable to avoid the use of broadband signals unless absolutely necessary.

In the Active Bat system the query-rate problem is addressed by supporting the deployment of multiple radio zones, each of which is covered by its own radio for querying bats and listening for registrations as shown in Figure 1. Each zone z has a corresponding set of receivers R^z that might hear a bat queried by the zone. When a zone z queries a bat, only the receivers in R^z are used to listen for the response. Multiple zones are allowed to query bats concurrently provided that their corresponding receiver sets are disjoint. For the example in Figure 1 zones one and three would be allowed to make queries concurrently.

Although the use of multiple radio zones makes it possible

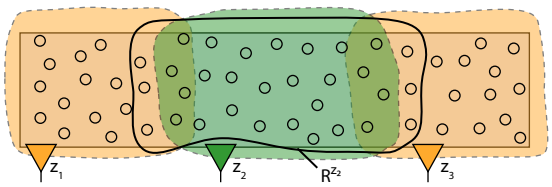

Figure 1. An Active Bat system with three separate radio zones. Circles indicate receiver locations.

to scale the Active Bat system to track large numbers of bats over a large area, there are a number of drawbacks. The first is increased hardware complexity, since multiple radio transmitters are required. The complexity of the system's software is also increased, since it must keep track of which bats are in which zones and determine when a bat moves from one zone to another. A third drawback is that radio zones do not allow as many bats to be queried concurrently as is often possible. For example it is often possible to schedule multiple bats at once within a single zone without the risk of interference (e.g. when they are at opposite ends of the zone); however this is never permitted.

In this paper we propose a more general solution to the query-rate problem, which is described in Section II. Section III outlines how this solution can be applied to the Active Bat system. We test our approach using real data obtained from the system in Section IV and compare it to the use of multiple radio zones in Section V.

II. A More General solution

In this section we present a solution to the query-rate problem based on the computation of a (time-varying) bounding region $\mathcal{C}_t^b$ for each bat b. Bounding regions are constructed such that the true position of b at time t is almost certainly contained by $\mathcal{C}_t^b$. The bounding region for a bat grows in the absence of measurements to take into account possible movement since it was last positioned. When the bat is queried the resulting measurements can be used to shrink the bounding region.

Given a bounding region, the set of receivers that *might* hear the bat's response can be calculated prior to a query being made. Only the receivers in this set need listen for the bat. Hence if multiple bats are associated with *disjoint* receiver sets then they can be queried simultaneously. Note that this approach is similar to the use of radio zones. In both approaches a set of receivers is constructed for each bat that over-approximates the set of receivers that would actually hear it should it be queried. In both cases bats can be queried concurrently provided that their corresponding receiver sets are disjoint. The key difference is that the radio zones approach uses static sets of receivers corresponding to each radio zone, where-as our solution uses dynamically generated sets based on bounding regions. Since the location of a bat can usually be bounded to a region far smaller than the size of a typical radio zone, our approach is able to construct smaller receiver sets and hence allow more concurrency.

In the remainder of this section we outline our solution in detail. In Section II-A we present a geometric algorithm for calculating bat bounding regions. In Section II-B we show how a bounding region can be used to calculate sets of receivers that might hear the response from a bat should it be queried. The use of such sets to help solve the query-rate problem is described in Section II-C.

A. Calculating Bounding Regions

Bounding regions can be calculated using a variety of algorithms in either two or three dimensions. In all cases the goal is to ensure that the bat lies within the region whilst at the same time making it as small as possible. Hence it is desirable to take environmental constraints such as walls into account, since they restrict possible movement. In this section we present a geometric algorithm that calculates a 2-dimensional bounding region for relatively little computational cost whilst taking environmental constraints defined by a floor plan into account.

We define a floor plan as an anti-clockwise polygon $\mathcal{P}$ with zero or more clockwise holes that represent obstructions (e.g. interior walls), as shown in Figure 2(a). A point is said to be inside $\mathcal{P}$ if it is inside the outer polygon but not inside any of the holes. Note that in this paper we will refer to the edges and vertices of both the outer polygon and the inner holes simply as the edges and vertices of $\mathcal{P}$.

Suppose that a bat b was last positioned at time $t-\delta t$ and that the position obtained was $\boldsymbol{x}$ in $\mathcal{P}$. Assuming a maximum bat speed $v_{\max}$, the bat could have moved a distance of

$$d_{\max} = \delta t \cdot v_{\max} + \epsilon \tag{1}$$

away from $\boldsymbol{x}$ at time t, where ϵ is an estimate of the maximum error in $\boldsymbol{x}$. Hence a point on the edge of the bounding region is either a point on the edge of $\mathcal{P}$ that is reachable from $\boldsymbol{x}$ by some path shorter than $d_{\max}$, or a point in $\mathcal{P}$ such that the shortest path to that point from $\boldsymbol{x}$ is of length $d_{\max}$. The following simple observations can be made regarding the shortest path between any two points $\boldsymbol{x}$ and $\boldsymbol{y}$ inside a floor plan [6].

- If the vector $\vec{\boldsymbol{xy}}$ does not intersect any edges of $\mathcal{P}$ then $\vec{\boldsymbol{xy}}$ is the shortest path between the two points.
- The shortest path may turn only at concave vertices of the polygon's outer edge and convex vertices of the hole edges. These vertices are known as the *possible turning points* of $\mathcal{P}$, as shown in Figure 2(a).

Given these observations it is possible to construct an algorithm that calculates bounding regions based on the computation of *visibility polygons*. Let an *expansion point*

$$e = (\boldsymbol{x}, r, \theta_1, \theta_2) \tag{2}$$

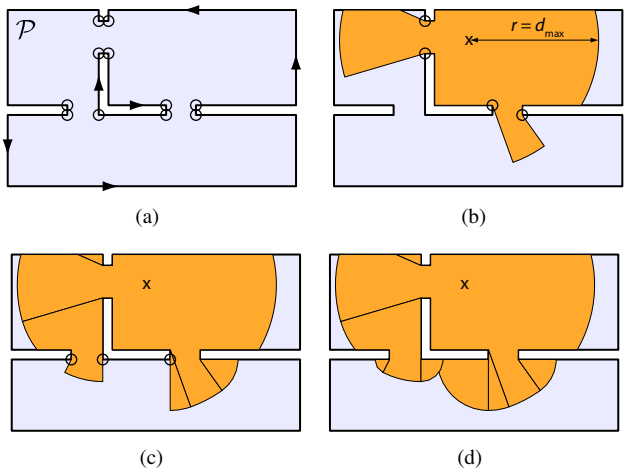

Figure 2. (a) A simple floor plan and the set of possible turning points (circles) on shortest paths through the environment. (b) The range-limited visibility polygon from a bat's previous position (marked by a cross) and the first turning points (circles) of shortest paths from that position. (c) The polygons and new turning points after the first set of expansion points has been processed. (d) The completed containment region.

consist of a point $\boldsymbol{x}$ in $\mathcal{P}$, a range r and angles $\theta_1 < \theta_2$. The visibility polygon $\mathcal{V}(e)$ is defined to contain a point $\boldsymbol{y}$ in $\mathcal{P}$ if and only if the following three conditions hold true:

1) There is a line-of-sight from $\boldsymbol{x}$ to $\boldsymbol{y}$.
2) The distance between $\boldsymbol{x}$ and $\boldsymbol{y}$ is not greater than the range r.
3) The angle from $\boldsymbol{x}$ to $\boldsymbol{y}$ falls between θ_1 and θ_2:

$$\theta_1 \leq \operatorname{atan2}(y_{\boldsymbol{y}} - y_{\boldsymbol{x}}, x_{\boldsymbol{y}} - x_{\boldsymbol{x}}) \leq \theta_2 \qquad (3)$$

where $\boldsymbol{x} = (x_{\boldsymbol{x}}, y_{\boldsymbol{x}})$ and $\boldsymbol{y} = (x_{\boldsymbol{y}}, y_{\boldsymbol{y}})$.

Due to space limitations we do not describe an algorithm for constructing visibility polygons here. Several suitable algorithms can be found in [6].

To construct the bounding region $\mathcal{C}_t^b$ for bat b at time t we first calculate the visibility polygon $\mathcal{V}(e_{\text{init}})$, where

$$e_{\text{init}} = (\boldsymbol{x}, d_{\max}, -\pi, \pi) \qquad (4)$$

in which $\boldsymbol{x}$ is the last position calculated for bat b and $d_{\max}$ is given by Equation 1. This polygon contains all positions that the bat could have reached since it was last positioned by travelling in a straight line, as shown in Figure 2(b). Next we find the set of first turning points on shortest paths from $\boldsymbol{x}$. Any such points must lie on the edge of $\mathcal{V}(e_{\text{init}})$. Hence we find the subset of the possible turning points of $\mathcal{P}$ that also lie on the edge of $\mathcal{V}(e_{\text{init}})$. For each point $\boldsymbol{p}$ in this set we determine whether it really is a first turning point by considering the vertices $\boldsymbol{v}_{-1}$ and $\boldsymbol{v}_{+1}$, which are the predecessor and successor vertices of $\boldsymbol{p}$ on the edges of $\mathcal{P}$. If the turn formed by the vertices $(\boldsymbol{x}, \boldsymbol{p}, \boldsymbol{v}_{-1})$ is anti-clockwise then a shortest path could turn at $\boldsymbol{p}$ and head in

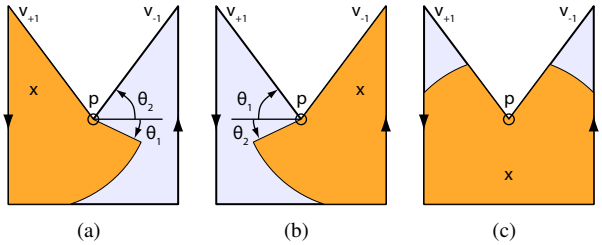

Figure 3. Conditions for first turning points. In each case the tag's previous position $\boldsymbol{x}$ is marked by a cross. (a) The turn $(\boldsymbol{x}, \boldsymbol{p}, \boldsymbol{v}_{-1})$ is anti-clockwise, so $\boldsymbol{p}$ is a first turning point. (b) The turn $(\boldsymbol{x}, \boldsymbol{p}, \boldsymbol{v}_{+1})$ is clockwise, so $\boldsymbol{p}$ is a first turning point. (c) $\boldsymbol{p}$ is not a first turning point.

any direction between the angles

$$\theta_1 = \operatorname{atan2}(y_{\boldsymbol{p}} - y_{\boldsymbol{x}}, x_{\boldsymbol{p}} - x_{\boldsymbol{x}}) \qquad (5)$$

$$\theta_2 = \operatorname{atan2}(y_{\boldsymbol{v}_{-1}} - y_{\boldsymbol{p}}, x_{\boldsymbol{v}_{-1}} - x_{\boldsymbol{p}}) \qquad (6)$$

as shown in Figure 3(a). Hence $\boldsymbol{p}$ is a first turning point in this case. Similarly $\boldsymbol{p}$ is a first turning point if the turn formed by the vertices $(\boldsymbol{x}, \boldsymbol{p}, \boldsymbol{v}_{-1})$ is clockwise, since in this case a shortest path could turn and head in any direction between the angles

$$\theta_1 = \operatorname{atan2}(y_{\boldsymbol{v}_{+1}} - y_{\boldsymbol{p}}, x_{\boldsymbol{v}_{+1}} - x_{\boldsymbol{p}}) \qquad (7)$$

$$\theta_2 = \operatorname{atan2}(y_{\boldsymbol{p}} - y_{\boldsymbol{x}}, x_{\boldsymbol{p}} - x_{\boldsymbol{x}}) \qquad (8)$$

as shown in Figure 3(b). If neither of these two conditions apply then $\boldsymbol{p}$ is not a first turning point, as shown in Figure 3(c).

For each of the identified first turning points $\boldsymbol{p}$ we construct a new expansion point

$$e_{\boldsymbol{p}} = (\boldsymbol{p}, d_{\max} - |\vec{\boldsymbol{xp}}|, \theta_1, \theta_2)\,. \qquad (9)$$

$\mathcal{V}(e_{\boldsymbol{p}})$ contains all possible points that can be reached by following a shortest path from $\boldsymbol{x}$ with a single turn at $\boldsymbol{p}$. For the example in Figure 2 there are four first turning points, as shown in Figure 2(b). Figure 2(c) shows the corresponding visibility polygons. To construct the bounding region the process of identifying first turning points and generating their visibility polygons is repeated for each newly generated polygon until no further turning points are found. The resulting set of polygons forms the bounding region within which the bat must be located, as shown in Figure 2(d). Note that the generated polygons may overlap since a path through a series of first turning points is not necessarily a globally-shortest path. The bounding region is the region covered by at least one polygon.

To avoid generating redundant polygons (i.e. polygons that are spanned entirely by other polygons in the set), turning points are processed in order of range from high to low. When an expansion point is processed the corresponding visibility polygon is generated only if another expansion point has not already been processed at the same vertex. If one has then we know that the point has already

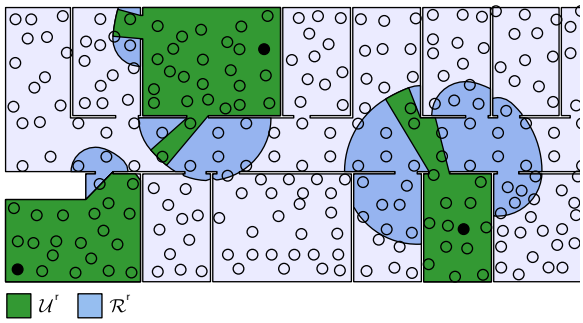

Figure 4. The receiver regions corresponding to three receivers (black dots) in the Active Bat system. Circles indicate the positions of other receivers.

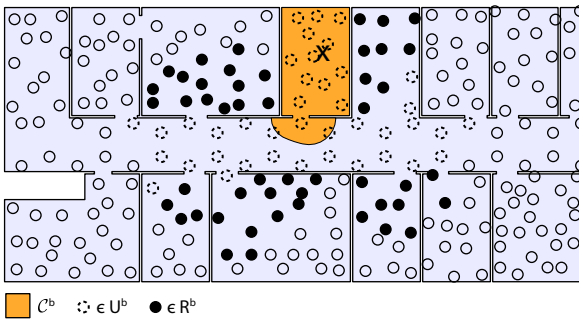

Figure 5. The containment region and receiver sets generated for a tag in the Active Bat system. The tag's previous position is marked by a cross.

been reached along a shorter path and hence the expansion point can simply be ignored. The complete algorithm for constructing a bounding region is as follows.

1) Let $\mathcal{C}_t^b := \phi$ be the containment region, let $P := \phi$ be the set of positions at which expansion points have been processed and let $E := \{e_{\text{init}}\}$ be the set of pending expansion points.
2) Remove the expansion point
$$e_{\text{next}} = (\boldsymbol{y}, r, \theta_1, \theta_2) \tag{10}$$
with the smallest range from E. If $\boldsymbol{y} \in P$ goto (5).
3) Generate the visibility polygon $\mathcal{V}(e_{\text{next}})$, add $\boldsymbol{y}$ to P and expand the containment region
$$\mathcal{C}_t^b := \mathcal{C}_t^b \cup \mathcal{V}(e_{\text{next}})\,. \tag{11}$$
4) Find the set of first turning points from $\boldsymbol{y}$. For each first turning point $\boldsymbol{p}$ add the expansion point $e_{\boldsymbol{p}}$ to E.
5) If $E \neq \phi$ goto (2), else return $\mathcal{C}_t^b$.

B. Calculating receiver sets

Given the bounding region $\mathcal{C}_t^b$ for a bat b at time t it is possible to construct the set of (potentially) useful receivers U_t^b that might receive a direct signal from the bat should it be queried. It is also possible to construct the superset $R_t^b \supseteq U_t^b$ that also contains receivers that might hear the bat's response only via a reflected or refracted path. A receiver that is in R_t^b but not U_t^b will never be useful when positioning the bat, however it must still be considered when deciding which bats can be queried concurrently. This is because the signal emitted by b may cause interference at the receiver that prevents it from being used to position another bat.

In order to calculate the receiver sets U_t^b and R_t^b we must first define coverage regions $\mathcal{U}^r$ and $\mathcal{R}^r$ corresponding to each receiver r. $\mathcal{U}^r$ is the region within which a bat can be located such that a signal emitted from it might reach r via a direct path. Since ultrasound cannot pass through walls, $\mathcal{U}^r$ is the visibility polygon from the position of r that is range limited by the maximum range of a bat, which is 6 m. $\mathcal{R}^r$ is the region from which a signal could be received via a direct or indirect path, calculated under the worst case assumption that a signal could be reflected anywhere in the environment (since the signal could bounce off a dynamic obstruction) with no loss of power (since the obstruction could be a very hard surface). Under these assumptions $\mathcal{R}^r$ can be calculated using the geometric tag bounding algorithm described in Section II-A, using

$$e_r = (\boldsymbol{x}^r, 6, -\pi, \pi) \tag{12}$$

as the initial expansion point where $\boldsymbol{x}^r$ is the position of the receiver. Figure 4 shows the coverage regions $\mathcal{U}^r$ and $\mathcal{R}^r$ for three receivers.

Given the receiver coverage regions $\mathcal{U}^r$ and $\mathcal{R}^r$, the receiver sets R_t^b and U_t^b for a bat b are constructed as follows. For each receiver r

$$r \in R_t^b \Leftrightarrow \text{Intersects}(\mathcal{C}_t^b, \mathcal{R}^r) \tag{13}$$
$$r \in U_t^b \Leftrightarrow \text{Intersects}(\mathcal{C}_t^b, \mathcal{U}^r)\,. \tag{14}$$

In other words a receiver r is included in R_t^b if and only if the bat *might* be located in a position where it *might* be heard by the receiver. The receiver is included in U_t^b if it *might* receive a signal from b via a direct path. Figure 5 shows an example of the receiver sets calculated for a single bat.

C. Using receiver sets

If the system decides to query a bat b at time t then the system need only listen for the bat's response using the receivers in U_t^b. Any receivers that are not in this set cannot receive a direct signal from b and hence cannot obtain a time-of-flight measurement that is useful for positioning a tag. Any measurements that would have been made by receivers not in U_t^b are effectively discarded. Hence measurements that are guaranteed to be of no use when calculating the bat's position are filtered out for free. Such measurements are those that are guaranteed to be of indirect signals from the bat or caused by other ultrasound sources in the environment, such as jingling keys and rustling crisp packets.

The sets U_t^b and R_t^b can also be used to determine when it is possible to query multiple bats concurrently without the

risk of interference between their emitted signals at useful receivers. Formally, let $Q_t = \{q_1, q_2, ..., q_n\}$ be the set of queries that are in progress at time t, meaning that the query has been made (i.e. transmitted over the radio channel) but the receivers are still listening for the bat's response. For each query q let

$$R^q = R^{b_q}_{t_q} \tag{15}$$

be the set of receivers that might hear the response, where b_q is the bat being queried and t_q is the time at which the query was made. Given this set, it is safe to start a new query of a bat b at time t if

$$\forall q \in Q_t \,.\, (U^b_t \cap R^q = \phi)\,. \tag{16}$$

Using this safety condition the Active Bat system's scheduler can determine when bat queries can be overlapped, hence allowing the system to track large numbers of bats over a large area without decreasing the average query-rate. Note that this condition does not permit concurrency when many bats are concentrated in a single region of space. In this case the receiver sets corresponding to different bats will always overlap and hence Equation 16 will not permit any query concurrency. For environments in which this is a regular occurrence it may be necessary to resort to using either broadband signals or an inside-out system.

III. Scheduling Bat Queries

Recall from Section I that the Active Bat system is a slotted system, meaning that bats are queried by the infrastructure component in fixed slots of length $20\,\mathrm{ms}$. The existing system maintains a list of bats L that are waiting to be queried, ordered from most urgent to least urgent. This list is updated based on events such as bat registrations, de-registrations, priority changes and query completions. For example a priority change will cause the list to be re-ordered, a query completion event will cause the queried bat to be re-inserted into the list so it is queried again in the future, and so on. We avoid this complexity by considering it the job of a pre-scheduler. The job of the scheduler itself is to remove bats from the pre-scheduler's list and perform the queries. The existing scheduler simply removes and queries the first bat in the list in each slot.

In this section we describe a greedy scheduling algorithm that makes use of the receiver sets U^b_t and R^b_t to query multiple bats concurrently whilst satisfying the safety condition given by Equation 16. The resulting system is still slotted, but can query multiple bats simultaneously in a single slot[1]. The set of bats B to be queried in a slot starting at time t is constructed as follows:

1) Let $A := R$ be the set of available receivers, which is initially equal to the set of all receivers R. Let $i := 1$ be an index into the pre-scheduler's list $L = [b_1, b_2, ..., b_n]$.
2) Consider bat $b = L[i]$. If
$$\forall r \in U^b_t \,.\, (r \in A) \tag{17}$$
then schedule bat b by adding it to B and updating the set of available receivers:
$$A := A \setminus R^b_t\,. \tag{18}$$
3) $i := i + 1$.
4) If $i \leq n$ goto (2), else return B.

Note that this algorithm satisfies the safety condition given by Equation 16 and has the nice property that the most urgent bat (i.e. $L[1]$) is always queried. When the slot is completed each bat b that was queried is positioned using the time-of-flight measurements obtained by receivers in U^b_t.

Figure 6. The floor plan of the area covered by the Active Bat system and bat traces calculated from our data-set between 11.00 a.m. and 11.30.

IV. Results

In this section we present results obtained by applying the greedy scheduling algorithm to data obtained from a real deployment of the Active Bat system. The test system is installed in one wing of our laboratory and covers a $485\,\mathrm{m}^2$ area as shown in Figure 6. It was not possible for us to modify the Active Bat system in order to test the algorithm 'for real'. To do so would have required extensive changes to the system's hardware that were not practical to implement. As an alternative we modified the system to query only a single bat in every slot. We then gave this bat to 15 different people who each carried the bat for one day whilst we logged the registration and de-registration events received by the pre-scheduler as well as the raw time-of-arrival measurements made by the receivers in each slot. By time-shifting these logs to be as though they all occurred on the same day, we obtain a data-set in which multiple bats are tracked over a single *virtual* day. For each slot in this day the data-set contains raw time-of-flight measurements corresponding to each bat being queried. Bat traces calculated from a half-hour period in this virtual day are shown in Figure 6.

We tested the greedy scheduling algorithm in simulation, using the real time-of-flight measurements from the data-set described above. The bat registration and de-registration

[1]This could be achieved over the out-of-band radio channel by notifying each bat to be queried and then sending a single *start-query* message to cause all of the notified bats to simultaneously emit an ultrasound response.

events were replayed into an emulated version of the system's standard pre-scheduler. The greedy scheduler then selected bats to query in each slot over the course of the virtual day. Bounding regions were calculated by the scheduler with the maximum position error set to $\epsilon = 0.03\,\mathrm{m}$. The maximum bat speed was assumed to be $v_{\max} = 4\,\mathrm{ms}^{-1}$, which was practically never exceeded by users of the system in a separate 24-day study [7]. For each slot the queries were emulated by returning the raw time-of-flight measurements recorded in the data-set for that slot that were obtained from the selected bats. Each queried bat b was then positioned by applying the Active Bat system's standard multi-lateration algorithm to the subset of those measurements that were obtained by receivers in U_t^b.

One advantage of our simulation is that it makes it possible to test different scheduling algorithms on the same data, which is not possible when testing a live deployment. The only disadvantage is that our data does not capture temporal events such as meetings that occur during a real day, since the users were actually tracked on separate days. We believe that this has a minimal effect on our results because most of the tracked users follow a similar daily routine. To ensure that our simulation was as realistic as possible, bounding regions were calculated using the most recently calculated bat positions but excluding those calculated during the previous slot. This models the fact that in practice the scheduler would have to decide which bats to schedule prior to these positions becoming available.

In a real deployment the scheduler would be required to schedule each slot during the previous slot. Hence the computation of bounding regions and receiver sets as well as the execution of the greedy scheduling algorithm would all have to be performed in at most $20\,\mathrm{ms}^{-1}$. Our Java implementation achieved this target for every slot during the virtual day. The maximum time required to schedule a single slot was $14\,\mathrm{ms}$. For sufficiently large numbers of bats it would be necessary to implement more efficient versions of the algorithms or upgrade the hardware on which they are executed. Bounding regions and receiver sets could be calculated more efficiently by growing them over successive slots when a bat is not positioned rather than re-computing them from scratch. These algorithms are also applied independently to each bat, making it trivial to spread the computation over multiple processors. Hence we believe that our approach is scalable to large numbers of bats.

Figure 7 shows the average queries-per-slot made by the greedy scheduler plotted as a function of the number of bats registered to the system (which is equivalent to the length of the pre-scheduler's queue), which varied naturally over the virtual day as tracked users moved in and out of the deployment area. The results show that it was often possible to schedule multiple bats simultaneously. As expected more concurrency was possible when more bats were registered. For example an average of 3.55 bats were scheduled per slot

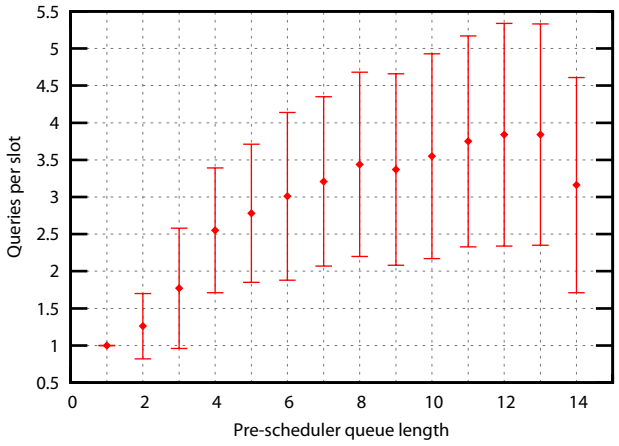

Figure 7. The mean number of bats queried in each slot plotted as a function of the number of registered tags. Error bars indicate $\pm\sigma$ values.

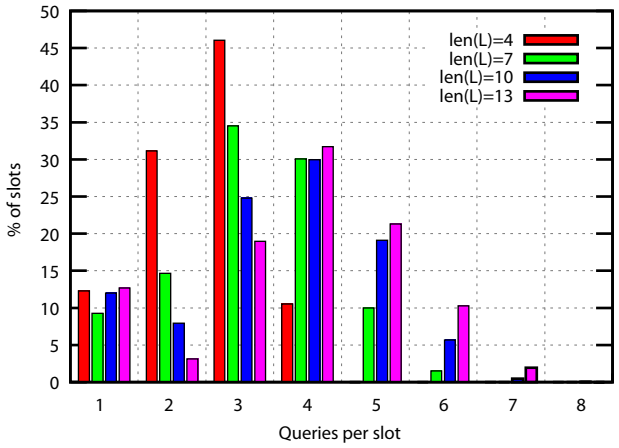

Figure 8. The queries-per-slot breakdown when 4, 7, 10 and 12 bats were registered.

during periods of the day in which 10 bats were registered to the system, compared to 2.78 when only 5 bats were registered. The maximum number of bats scheduled in a single slot was 8. Note that although Figure 7 shows a drop in the average number of queries when 14 bats were registered, we do not believe that this data-point is reliable since it is based on very few measurements (there was only a single 20 second period during which 14 bats were registered).

Figure 8 shows a breakdown of the number of queries made in slots when 4, 7, 10 and 13 bats were registered. Note that these distributions are dependent on the order in which bats are queried, and hence depend on the specifics of the pre-scheduler. In the Active Bat system the pre-scheduler lowers the priority of bats that the system has previously failed to position, since these bats are likely to be partially or fully occluded from the receivers. If this were not the case

then such bats would be scheduled more frequently. Since such bats would have relatively large bounding regions (as they will not have been recently positioned) we would expect this to increase the percentage of slots in which very few bats are queried.

V. Zones vs receiver sets

The Active Bat system currently uses multiple radio zones to address the query-rate problem. In practice the deployment described above would be divided into at most three radio zones, with the radio zones at either end of the central corridor overlapping the zone in the middle (see Figure 1). The two end zones would be able to query bats concurrently, meaning that a maximum of 2 bats could be scheduled in a single slot. If the middle zone were to query a bat in every other slot then the average query-rate would be 1.5. Hence the results shown in Figure 7 clearly illustrate that the use of bounding regions permits more concurrency than is possible using multiple radio zones.

There are several benefits to replacing multiple radio zones entirely with the algorithms presented in this paper. The hardware complexity of the system is reduced since only one radio controller is required. The software complexity is also reduced because it is not necessary to perform hand-overs as bats move between zones. Although these benefits are attractive, we believe that multiple radio zones are still necessary to achieve a truly scalable system. When a bat registers with a system consisting of multiple zones it does so only with the zone within which it is located. This effectively positions the bat within that zone before it is ever queried, meaning that other bats can be queried concurrently in different (non-overlapping) radio zones at the same time as the first query is being made to the newly registered bat. When using only bounding regions the initial bounding region of a newly registered bat covers the entire deployment area. Hence no other bats can be queried whilst a newly registered bat is being queried for the first time. When the system is scaled to cover a sufficiently large area the number of bat registrations will be such that only one bat is ever queried in each slot. Hence we see the use of bounding regions as a within-zone technique to boost the average query-rate, rather than as a replacement to the use of multiple radio zones. Although we cannot entirely remove the need for multiple zones, the use of bounding regions do make it possible to have fewer, larger radio zones, and hence partially reduce the system's complexity.

VI. Conclusions and future work

In this paper we have presented a solution to the query-rate problem and applied it to the Active Bat location system. By maintaining a bounding region within which each bat is almost certainly located, time-varying sets of receivers that might hear each of the bats should they be queried are constructed. These sets are used to determine when multiple bats can be queried concurrently without the risk of interference. We tested this approach using a simulation driven by real data obtained from the system. Our results demonstrate that bounding regions permit more query concurrency than is possible using only multiple radio zones. To achieve a truly scalable system we propose that the algorithms presented in this paper should be applied within multiple radio zones, rather than replacing them altogether.

In the future we plan to investigate ways of improving the estimation of bounding regions. For example the maximum bat speed could be set individually for each user based on observed behaviour. We also plan to investigate more aggressive scheduling techniques. For example the scheduler might choose to query multiple bats with overlapping receiver sets concurrently, provided that it is likely that they can be positioned using the remaining non-overlapping receivers.

Acknowledgements

The authors would like to thank Andrew Rice for his insightful comments. This work has been partly funded by the EPSRC.

References

[1] M. Addlesee, R. Curwen, S. Hodges, J. Newman, P. Steggles, A. Ward, and A. Hopper, "Implementing a sentient computing system," *Computer*, vol. 34, no. 8, pp. 50–56, 2001.

[2] N. B. Priyantha, "The cricket indoor location system," Ph.D. dissertation, Massachusetts Institute of Technology, Cambridge, MA, USA, 2005.

[3] A. Smith, H. Balakrishnan, M. Goraczko, and N. B. Priyantha, "Tracking moving devices with the cricket location system," in *Proceedings of the Second International Conference on Mobile Systems, Applications and Services (Mobisys 2004)*, New York, NY, USA, June 2004, pp. 190–202.

[4] M. Hazas and A. Ward, "A novel broadband ultrasonic location system," in *Proceedings of the Fourth International Conference on Ubiquitous Computing (UbiComp 2002)*, 2002, pp. 264–280.

[5] ——, "A high performance privacy-oriented location system," in *Proceedings of the First IEEE International Conference on Pervasive Computing and Communications (PerCom 2003)*, 2003, pp. 216–223.

[6] S. Ghosh, *Visibility Algorithms in the Plane*. New York, USA: Cambridge University Press, 2007.

[7] R. Harle, "Maintaining world models in context-aware environments," Ph.D. dissertation, University of Cambridge, Cambridge, UK, 2004.