PROCEEDINGS OF THE 2nd WORKSHOP ON POSITIONING, NAVIGATION AND COMMUNICATION (WPNC'05)
& 1st ULTRA-WIDEBAND EXPERT TALK (UET'05)

Efficient Proximity Detection for Location Based Services

Georg Treu and Axel Küpper
Mobile and Distributed Systems Group, Institute for Informatics
Ludwig-Maximilian University, Munich, Germany
e-mail:[georg.treu|axel.kuepper]@ifi.lmu.de

Abstract - Proactive *location-based services* (LBSs) are characterized by the fact that (spatial-related) events are automatically triggered without explicit user invocation. An example is the notification of users when entering a certain zone. Since continuous tracking of a target is not feasible due to the limitations of the air-interface, efficient position update strategies have to be conceived. For relating the position of a mobile terminal with fixed geographic entities, such as points of interest, relatively simple strategies are sufficient. In this paper an efficient but simple update strategy for detecting proximity *between* mobile terminals is presented.

1 Introduction

Proactive *location-based services* (LBSs) are characterized by the fact that they are automatically triggered if a pre-defined spatial-related event occurs, for example if a target enters a certain zone and the LBS user wants to be notified about that.

In today's networks, almost all LBSs are reactive. A prominent example is the buddy finder, where users can request the position of their buddies (assuming that each buddy agrees to that). If the user invokes this service, the positions of all targets are collected.

Proactive LBSs are scarcely available today, one reason for that certainly being the lack of an efficient position management. The proactive complement to the buddy finder is a community service, which automatically notifies a member as soon as another member approaches, for example when the distance between them falls below a pre-defined threshold. To realize this service an efficient tracking strategy needs to be applied since naively tracking all community members continuously is too excessive with regard to the signaling overhead caused at the air-interface.

The problem of detecting proximity between mobile terminals (MTs), parameterized with a given distance threshold, is a general one. Applications for which an efficient proximity detection service is fundamental range from the mentioned community service to mobile gaming, fleet management and others.

In this paper we describe and evaluate a strategy for efficient proximity detection among MTs. For this purpose, we assume terminal-based positioning like GPS, i.e., the MT performs range measurements and derives its position from the results. Furthermore, we precondition that an LBS is able to dynamically negotiate with the MT the way position updates are sent. For this purpose different update strategies are possible (compare [LeRo02], [Kuep05]). Among them are the periodic (updates are sent in fixed intervals), zone-based (an update is sent whenever an MT enters or leaves a given zone), or distance-based strategy (an update is sent whenever the distance between the previously reported update and the current update exceeds a certain threshold). An architectural sketch of how these strategies could be realized on MTs is given in [KuTr05] and [CCR03].

2 Idea

For proximity detection among MTs, the position of more than one MT must be interrelated. Therefore, various approaches are thinkable. A naive one would be to track the MTs periodically and to reconsider their distance with every position update. Unfortunately, this procedure burdens the air-interface excessively, because of the high number of position updates between the MTs and the server.

To our knowledge, the only work that addresses this issue so far is [AEM+04]. In the algorithm proposed there, so called "strips" are arranged in a way that, without entering a strip, it is not possible for two MTs to come into proximity. When an MT enters a strip, it sends a position update to the server (or, in peer-to-peer mode to all the other MTs) and the strips on all of the MTs are recalculated. Compared to a naive approach, the strips algorithm reduces the number of position updates. Nevertheless, distributing the position updates to all of the MTs as well as specifying the strips, which are represented as polygons, is associated with high costs.

Instead of sending updates on entering strips, which is equivalent to a zone-based strategy (compare last section), the proximity detection mechanism presented in this paper utilizes a distance-based update strategy. An instance of the distance-based strategy is called a *Distance-Based Update Job (short: Dist Job)*. A Dist Job carries the (spatial) distance an MT has to cover between two subsequent position updates. For example, in a Dist Job of 50 m, position updates are reported with every 50m the MT moves.

In our approach, the distances underlying the Dist Jobs are selected in way so that proximity detection among MTs can be efficiently realized. Dist Jobs are issued to and removed from an MT dynamically by the server. The idea is that two MTs cannot come into proximity without either one sending a position update. The distances selected for this are dependent on the spatial distribution and concentration of the MTs, whereby the distance selected for a particular MT is determined by the position and type of Dist Job of its most nearby neighbor. Therefore, in contrast to [AEM+04], upon a position update only a small number of MTs need to be reconfigured. This makes our approach more efficient with regard to the number of messages sent over the air-interface.

Figure 1 shows a screenshot of the algorithm during simulative evaluation. The circles around the simulated MTs represent Dist Jobs issued to them[1].

In the next section, the algorithm is described in detail. This includes formal parameters, internal variables used as well as pseudo-code. Furthermore, a possible optimization is discussed. It comes into effect when distance-based update jobs are distributed among a group of more than one MT at a time. The distances of the jobs assigned to the MTs can e.g. be chosen to reflect the recent mobility behavior of the MTs.

After that, in section 4, the algorithm is evaluated against two scenarios (configured by the number of MTs, the size of the simulated area, proximity threshold, etc.). Therefore we developed a java-based simulation framework. The proposed strategy is compared to a naive approach, where fixed Dist Jobs are issued on an MT upon registration with the server. The paper concludes in section 5 with an outlook to further work.

[1]Overlapping circles mean that the respective MTs have already been in proximity so that their spatial relation is no longer considered until re-registered with the algorithm.

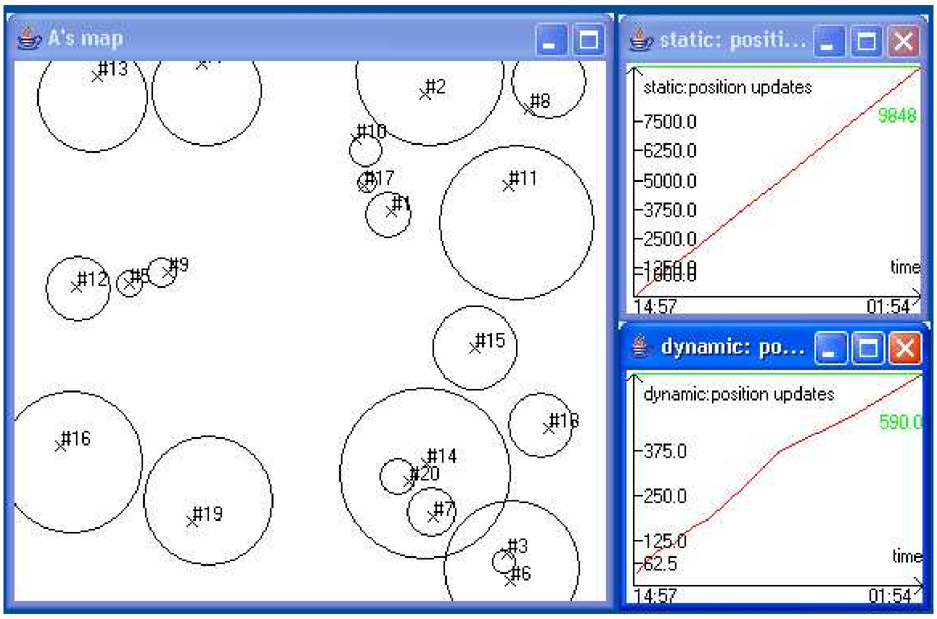

Figure 1: The proximity detection strategy during simulative evaluation

3 An Algorithm for Efficient Proximity Detection

3.1 Variables

We define C as the threshold distance where two MTs a and b are considered to be in proximity and an alert needs to be sent.

For each MT a we maintain information about its last position fix reported *Fix(a)* as well as the distance of the Dist Job issued to a, *DistJob(a)*. For simplicity, we assume the accuracy of position fixes by the MTs to be perfect. Inaccurate position fixes associated with an accuracy bound, as e.g. by GPS, can be easily integrated with the algorithm.

Upon *Fix(a)* and *DistJob(a)*, the current position of an MT a can be estimated. a must be contained within a circle around *Fix(a)* with radius *DistJob(a)*, because, according to its Dist Job, a would report a position update on leaving the circle. By relating the circles of two MTs a and b, their mutual distance can also be estimated. Figure 2 illustrates this coherence and the associated variables.

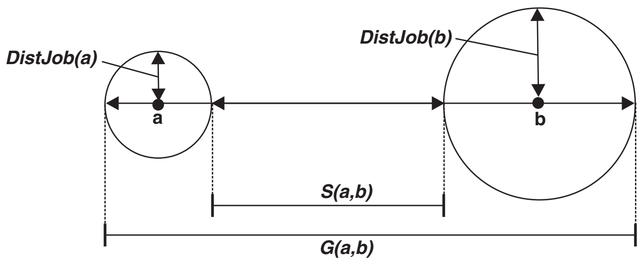

Figure 2: Estimating the distance between two MTs.

We define *S(a,b)* as the smallest distance possible between MTs a and b, both provided with Dist Jobs. Formula 1 calculates the value of *S(a,b)* upon *Fix(a)*, *DistJob(a)*, *Fix(b)* and

DistJob(b) (*distance()* is a function that calculates the spatial distance between two coordinates).

$$S(a,b) \quad = \quad distance(Fix(a), Fix(b)) - DistJob(a) - DistJob(b) \tag{1}$$

Analogous, in formula 2 *G(a,b)*, the greatest distance possible between a and b, is defined.

$$G(a,b) \quad = \quad distance(Fix(a), Fix(b)) + DistJob(a) + DistJob(b) \tag{2}$$

Furthermore, for each pair of MTs a and b, the field *rep(a,b)* records if proximity between a and b has already been detected by the algorithm. For each pair of MTs proximity is reported only once. Pairs of MTs for which *rep(a,b)==true* are not considered when issuing Dist Jobs. Nevertheless, it is possible to reset *rep(a,b)* dynamically by an external module (*a "separation detector"*), but this aspect is not considered in this work.

3.2 Algorithm Details

The proposed algorithm consists of two operations. Only one invocation can be active at a time. Concurring invocations are handled on a first-come-first-served basis. Both operations are also shown in pseudo code below.

- *NewMT* adds a new MT a to the proximity detection process. At this stage, an initial Dist Job must be placed on a. Therefore, a's current position is polled first. Then, the *PositionUpdated*-routine is called for further proceeding.
- *PositionUpdated* is invoked whenever an MT a reports a position update: *Fix(a)*. This usually happens because a Dist Job has triggered the update, but position updates out of arbitrary situations are supported. First, *Fix(a)* is related with the (estimated) positions of all other MTs b. If, according to *S(a,b)*, it is theoretically possible that a and b are within the proximity threshold, the current position of b is polled and compared to *Fix(a)*. If, after that, a and b are sure to be in proximity, an alert is generated. All MTs that have been position-updated in the method (a as well as those b that had to be polled) are provided with new Dist Jobs. The maximum Dist Job that can be issued on an MT c is calculated by *M(c)* in a way so that for all other terminals d, the distance between c and d cannot fall below C without either c or d sending a position update. As shown in the pseudo-code, *M(c)* is calculated upon *S(c,d)* and *C*.

The parameter T, introduced in the pseudo-code, is referred to as the *borderline tolerance* of the strategy. T has the following implications:

- If a and b are in distance less than C, an alert *must be* sent.
- If a and b are in distance between C and $C + T$, an alert *may be* sent.
- If a and b are in distance greater than $C + T$, an alert *must not be* sent.

T is necessary to avoid the negative effects when two MTs approach very closely to the distance threshold C, but without exceeding C. Without T, a very rapid series of changing Dist Jobs would be necessary in this case.

```
NewMT(a){
        Poll the position of a, which results in Fix(a).
        PositionUpdated ( Fix(a) ).
}

PositionUpdated(Fix(a)){

    DistJob(a):=0.
    For every pair of MTs (a,b) with rep(a,b)==false:
    {
        if S(a,b) <= C + T:         //a and b may be in reach.
        {
            if G(a,b) <= C + T:     //a and b are in reach.
                Send alert, rep(a,b):=true.
            otherwise:
            {
                Poll the position of b,
                resulting in Fix(b).
                DistJob(b):=0.
                if S(a,b) <= C + T: //a and b are in reach.
                    Send alert, rep(a,b):=true.
            }
        }
    }

    For every MT c with DistJob(c)==0:
    {
        M(c):=MAX.
        For every pair of MTs (c,d) with rep(c,d)==false:
            M(c):=Min( M(c), S(c,d) - C ).
        Issue a Dist Job on c, DistJob(c)<=M(c).
        (different optimizations possible).
    }
}
```

We illustrate our algorithm in two different situations.

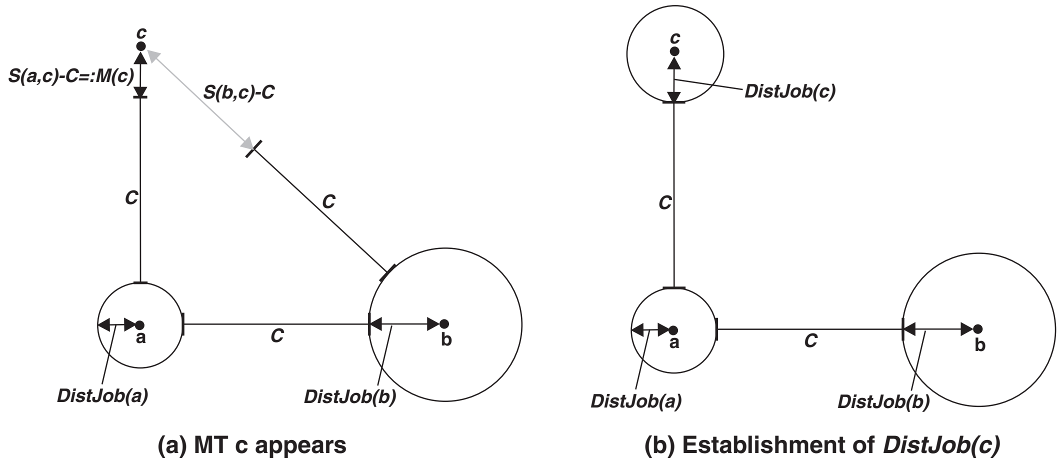

Figure 3: Position update of an MT without consequences.

Figure 3 shows a position update by an MT c that has no direct consequences for the other MTs a and b. Upon *S(a,c)*, *S(b,c)* and C the (exact) position of c is compared with the estimates for a and b. Since for both MTs, a and b, it is impossible to be within distance threshold C to c, they are not polled. c is provided with a Dist Job according to *M(c)*, which is calculated with regard to c's nearest neighbor, a.

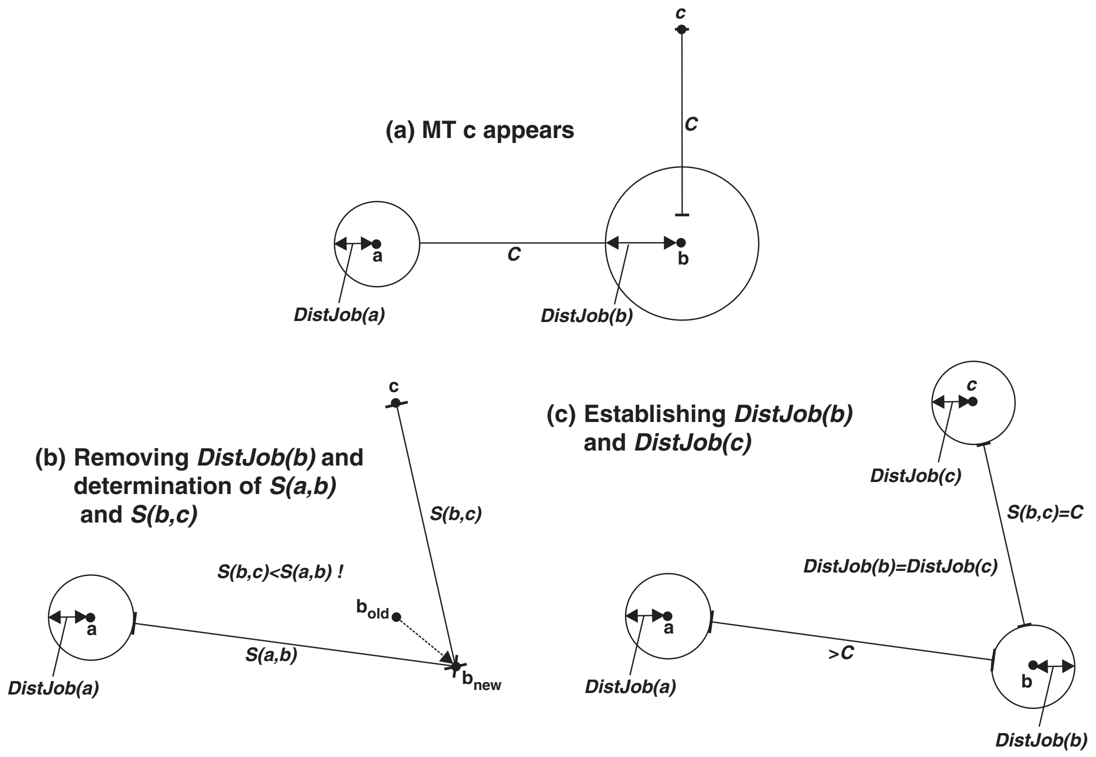

Figure 4: Position update of an MT with consequence of polling other MT.

In figure 4, the case is not as simple. This time, MT c is close enough to the imaginary circle of b so that distance threshold C is not guaranteed to be kept. b has to be polled. The polling yields *Fix(b)* being closer to *Fix(c)* than to a's circle and also *Fix(c)* being closer to *Fix(b)* than to a's circle. Therefore, for determining *DistJob(b)*, b's distance to c and for *DistJob(c)*, c's distance to b is considered (note that this symmetric relation is not necessarily given). In the figure, MTs b and c both get a Dist Job of equal size: $DistJob(b) = DistJob(c) = \frac{S(b,c)-C}{2}$. However, the rule that has to be adhered to is less restrictive: $DistJob(b) + DistJob(c) <= \frac{S(b,c)-C}{2}$. A possible optimization would be to select the distances in a way that reflects the mobility behavior of the MTs. While slowly moving or stationary MTs can be allowed small distances, fast moving MTs should have available as much "room" as possible. If it would be known to the server, that, e.g. c is currently moving quite fast, while b is stationary at the moment, the selection could be done so that $DistJob(c) > DistJob(b)$. For determining the current mobility behavior of an MT, the number of position fixes reported by it as well as the *Dist Jobs* issued to it could be recorded. An alternative would be to include e.g. the current speed of an MT in a position update. The information can be provided by GPS.

To assist the efficiency of the presented strategy for proximity detection, in the next section, it is evaluated by simulation against a naive strategy.

4 Evaluation by Simulation

For evaluating our approach we have developed a Java simulation (compare figure 1).

The proposed algorithm was compared to a strategy where each MT is configured with a fixed distance-based update job upon registration. The distance is set to *T/2* (compare *borderline tolerance* T, last section). This way, the static approach is guaranteed to work correctly, since it is not possible that two MTs reduce their distance by more than T between two subsequent position updates. Nevertheless, also for the naive strategy it is necessary to poll MTs for their position when an MT is suspected to be within distance threshold C of another MT.

4.1 Configuration

We considered two different scenarios A and B. For both scenarios a series of simulations with increasing numbers of users was conducted. Each simulation run was executed until half of all possible pairs of MTs were proximity detected (remember from above that the algorithm removes detected pairs of MTs from the list). For n simulated MTs, this corresponds to detecting $\frac{n(n-1)}{4}$ proximity events. The Smooth Random Mobility Pattern ([Bett01]), a synthetic but relatively realistic model, was implemented for simulating the movement of the MTs.

Table 1: Simulation Results for Scenario A

Approach	MTs	Position Updates	Server Msgs	Total Msgs
Dynamic	5	564	658	1222
Dynamic	10	1661	2035	3696
Dynamic	15	4344	5387	9731
Dynamic	25	8053	10169	18222
Dynamic	50	45759	58164	103923
Dynamic	75	46003	60410	106413
Dynamic	100	128713	168271	296984
Naive	5	17940	17	17957
Naive	10	30190	62	30252
Naive	15	55710	183	55893
Naive	25	85899	359	86258
Naive	50	411599	2406	414005
Naive	75	269056	3235	272291
Naive	100	777290	8884	786174

In scenario A, a metropolitan area of 10 km x 10 km was considered. The borderline tolerance T was set to 50m and the distance threshold C to 200m. The smooth random pattern was configured with a maximum speed of 60 km/h, but with average speed of far less than that, representing walkers and bikers.

In scenario B, we considered a wider area of 400 km x 200 km. T was set to 2km and C to 10km. The smooth random pattern was configured with a maximum speed of 180 km/h, but with more stationary moments than that for scenario A. The intention was to simulate traveling pattern of users within a state-like area.

Table 2: Simulation Results for Scenario B

Approach	MTs	Position Updates	Server Msgs	Total Msgs
Dynamic	5	478	600	1078
Dynamic	10	1479	1819	3298
Dynamic	15	3576	4474	8050
Dynamic	25	7476	9612	17088
Dynamic	50	25220	33297	58517
Dynamic	75	49248	65822	115070
Dynamic	100	82867	113209	196076
Naive	5	26997	27	27024
Naive	10	44286	67	44353
Naive	15	89105	151	89256
Naive	25	112348	438	112786
Naive	50	247463	1745	249208
Naive	75	405827	3982	409809
Naive	100	511560	7767	519327

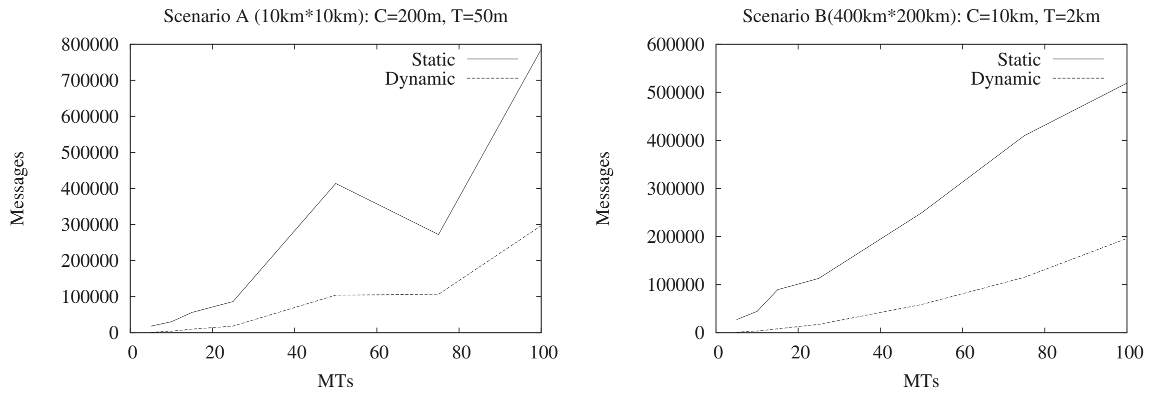

Figure 5: Graph for Scenarios A and B

4.2 Results

We estimate the message size of *position updates* and *server messages* to be relatively equal since both types of messages carry a lightweight payload. A *server message* was accounted for the placement of update jobs, as well as for requesting the position of an MT. MTs polled for their position were accounted with a *position update* and a *server message* at the same time.

For all configurations we tried, the proposed dynamic strategy clearly outperformed the naive approach with regard to total messages sent. Although the proposed algorithm produces an increased number of server messages for dynamically updating Dist Jobs, for all configurations we found that these costs were clearly outweighed by the efficiency gains associated with the reduced number of position updates.

See table 1 to overview the simulation results for scenario A, table 2 for scenario B. In the tables, generated *position updates* and *server messages* are separately listed, while in Figure 5 the number of *total messages* is compared graphically between the two approaches. The kink in the chart for scenario A is attributed to the random-driven mobility pattern.

5 Conclusion and Future Work

In this paper, a novel position update strategy for proximity detection among MTs was presented. A simulative evaluation showed that the proposed strategy is clearly superior to a naive one. Furthermore, the proposed strategy is much simpler than the only concurrent approach known to us: [AEM+04]. We speculate that our approach is more efficient with regard to the air-interface, because a position update of an MT leads to less subsequent messages. A simulation that compares both strategies quantitatively has been postponed to future work.

Apart from that, we also have plans for devising an update strategy that does the opposite of the presented one, namely detecting when two MTs that have been initially in proximity get separated. The mechanism can be used in conjunction with the one described in this work.

References

[AEM+04] Amir, A., A. Efrat, J. Myllymaki, L. Palaniappan, K. Wampler. Buddy Tracking — Efficient Proximity Detection among Mobile Friends. *Proceedings of IEEE Infocom 2004*, Hongkong, March 2004.

[KuTr05] Küpper, A.,G. Treu. From Location to Position Management: User Tracking for Location–based Services. *Proceedings of the 14. Fachtagung Kommunikation in Verteilten Systemen KIVS05*, Kaiserslautern Germany, February 2005, Springer–Verlag.

[Kuep05] Küpper, A. *LBS — Fundamentals and Operation.* To be published. John Wiley & Sons, 2005.

[LeRo02] Leonhardi, A., K. Rothermel. Protocols for Updating Highly Accurate Location Information. A. Behcet (Ed.). *Geographic Location in the Internet.* Kluwer Academic Publishers, 111–141.

[Bett01] Bettstetter, C. Smooth is better than sharp: a random mobility model for simulation of wireless networks *Proceedings of the 4th ACM international workshop on Modeling, analysis and simulation of wireless and mobile systems*, Rome, Italy, 2001, ACM Press, 19–27

[CCR03] Xiaoyan Chen and Ying Chen and Fangyan Rao An efficient spatial publish/subscribe system for intelligent location-based services *Proceedings of the 2nd international workshop on Distributed event-based systems*, San Diego, California, 2003, ACM Press, 1–6