# Maintaining Information about Nearby Processors in a Mobile Environment

Faith Ellen *, Sivaramakrishnan Subramanian **, and JenniferWelch**

**Abstract.** The problem of maintaining information about the location of nearby processors in a mobile adhoc network is considered. A new scalable, deterministic algorithm is presented, provided processors can only move along a line. Many open questions and directions for future work are discussed.

## 1 Introduction

In many algorithms designed for mobile adhoc networks, processors are assumed to know information (such as location) about the other processors that are located nearby [1, 3, 4, 7, 9]. However, as processors move, the information may change and the set of processors each particular processor needs to know about may change.

Updating information is not simply a matter of each processor broadcasting changes to its information when they occur. One problem is that nearby processors may not necessarily be within transmission range of one another. To handle this, processors have to relay some of the information they receive from their neighbours. For example, Calinescu [5] shows how processors can maintain information about the processors that are at most two hops away, assuming that broadcasts never interfere with one another.

A more significant problem is that there is interference when different processors perform concurrent broadcasts. In this case, the information contained in the messages will not be received by these processors. Furthermore, a processor that is in transmission range of two or more of these processors will also not receive any messages, even if it doesn't broadcast and the broadcasting processors are not within transmission range of one another.

One way to avoid interference is to employ time slicing. Each processor periodically gets allocated a time slot in which it can broadcast updates to its information. Unfortunately, the time between a processor's broadcasts depends on the total number of processors in the system and, hence, this solution is not scalable. In particular, if the number of processors is large, the information known about a specific processor will be out of date most of the time.

* University of Toronto, Canada, supported by the Natural Sciences and Engineering Research Council of Canada and the Scalable Synchronization Research Group of SUN Microsystems, Inc.

** Texas A&M University, USA, supported in in part by National Science Foundation grant 0500464 and Texas Advanced Research Program grant 000512-0007-2006.

Another approach to gathering and maintaining information about nearby processors and for communicating with them, popular in practice, is to settle for probabilistic guarantees on performance, by relying on random behavior of the processors. For instance, in the hello protocol [3], used to discover and maintain neighbor relationships in mobile adhoc networks, each processor periodically broadcasts a hello packet. When another processor receives such a message, it knows that the sender is currently its neighbor. It is assumed that the likelihood of missing more than a fixed number of such hello packets from a neighbor, due to collisions, is negligible. This likelihood can be reduced even further by adding some random jitter to the time when hello packets are sent.

In the IEEE 802.11 standard, the medium access control protocol resolves channel contention using randomization: a processor chooses a random "back-off interval" in some range, and waits this amount of time before performing the RTS/CTS protocol [2]. In this protocol, a few short control packets are exchanged, in which the processor requests to send (RTS). Once it has been cleared to send (CTS), the processor broadcasts the data. Since the control packets are short, the probability of collisions is assumed to be small. The use of a random backoff interval makes this probability even smaller.

The ability of processors to discover their neighbors and to communicate with them using these protocols is probabilistic. There always exists a (small) probability that a processor has incorrect information about its neighbors and that it is unable to transmit information to its neighbors. For some applications, such as real-time applications, which require stringent guarantees on system behavior, typically deterministic upper bounds on message delays, such probabilistic behavior is not acceptable.

In this paper, we consider how information about nearby processors can be maintained deterministically by processors as they move. Section 2 presents a model in which to study this problem. In Section 3, we give an algorithm for a restricted case in which processors move along a line. This case is a good model for one-dimensional environments such as highways and railroads. More importantly, the work in this section provides a good foundation for addressing various issues that arise in more general versions of the problem, discussed in Section 4. We also hope that the algorithm for the restricted case will provide insight to solutions for other versions.

## 2 Model

We consider a set of $n$ processors moving in a Euclidean space, for example, on the plane or along a line. The motion of a processor can be described by its *trajectory*, a function that specifies the location of the processor as a function of time. There is an upper bound $\sigma$ on maximum speed of a processor.

Because processors occupy space, there is also an upper bound on the density of processors. However, trajectory functions may be allowed to intersect. For example, a four lane highway can be modelled as a line on which at most four processors can occupy the same location at the same time.

Each processor is assumed to know its current location, for example, via GPS, and its future trajectory, for some period of time. We also assume the existence of a global clock, which can be read by all processors.

Processors communicate by wireless broadcast [10, 8]. There are two important parameters related to the reception of broadcast messages, the *broadcast radius*, $R$, and the *interference radius*, $R' \geq R$. If a processor $p$ is within distance $R$ of another processor $q$ during the time that $q$ is broadcasting a message, then the message *arrives* at $p$. If, in addition, no other processor within distance $R'$ of $p$ transmits at any point during this period of time, then $p$ *receives* the message broadcast by $q$. Collisions occur when a message arrives at a processor, but is not received. A processor broadcasting a message may or may not be aware of collisions that occur at processors within its broadcast radius.

Broadcasts occur during *broadcast slots*, which are disjoint unit intervals of time. They are sufficiently long so that a message which starts being broadcast at the beginning of a broadcast slot arrives at all processors within the broadcast radius of the sender by the end of the broadcast slot. Broadcast slots start every $u$ units of time and each is followed by an interval of $u - 1$ time units that can be used by other algorithms. We assume that broadcast slot 0 starts at time 0. Then broadcast slot $j$ starts at time $ju$.

## 3 An Algorithm for Maintaining Information on a Line

In this section, we present a scalable, deterministic algorithm for processors to maintain trajectory information about all nearby processors, provided processors can only move along a line. We make two simplifying assumptions. The first assumption is that each processor knows its entire trajectory function and can easily share this information with other processors. This means that trajectory functions must be representable in a relatively succinct way. The second assumption is that, at the beginning of the algorithm, each processor knows the entire trajectory of every nearby processor. This can be achieved by simply assuming that processors are initially sufficiently far apart from one another, so that no processors has any other processor nearby.

Our approach is to partition the line into *segments* of length $G$, starting at each multiple of $G$. The segments are coloured with the $m$ *colours* $0, 1, \ldots, m - 1$. Segment $i$ is the half open interval $[iG, (i + 1)G)$ and has colour $i \bmod m$. Segments with the same colour are assigned to the same broadcast slots and segments with different colours are assigned to different broadcasts slots.

Our algorithm proceeds in phases, consisting of $m - 1$ broadcast slots. The segments that are not assigned to any broadcast slot in a particular phase are assigned to the first broadcast slot in the next phase. During each broadcast slot, the only processors allowed to broadcast are those that were in a segment assigned to this broadcast slot at the beginning of the phase.

To avoid collisions between broadcasts performed by processors in the same segment, the processors in each segment at the beginning of a phase choose a *leader*. During each broadcast slot in that phase, only the leaders of segments

assigned to the broadcast slot can perform broadcasts. Provided that all processors in the same segment know the locations of the processors that are currently in the segment, they can agree on the same leader, in some predetermined way, using only local computation. (Another possibility is to choose a leader of a segment immediately before a broadcast slot to which it is assigned. The problem with this approach is that a processor moving from one segment to another segment during a phase might entirely miss its opportunity to broadcast during a phase, even if both segments are scheduled.)

If segments of successively increasing colour are assigned to successive broadcast slots, then information propagates rightwards quickly, but may be slow to propagate leftwards. Similarly, if segments of successively decreasing colour are assigned to successive broadcast slots, then information propagates leftwards quickly, but may be slow to propagate rightwards. Instead, our assignment of segments to broadcast slots interleaves sequences of segments of successively increasing and successively decreasing colours.

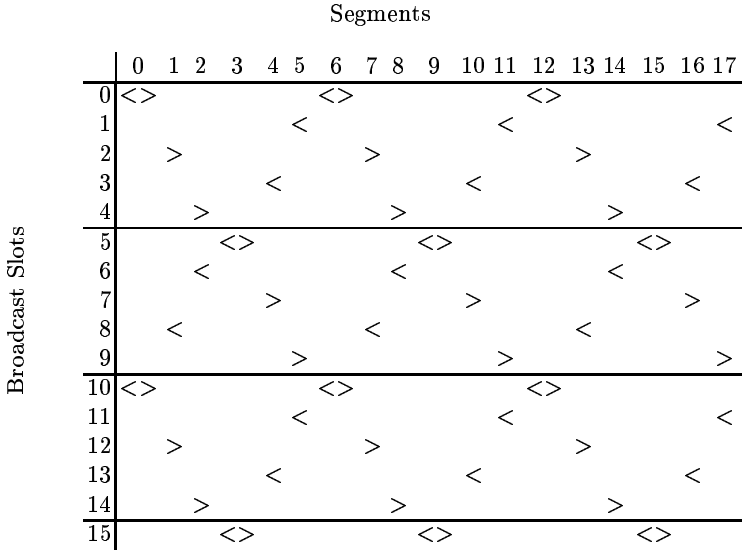


**Fig. 1.** The Broadcast Schedule for $m = 6$

The broadcast schedule for $m = 6$ is illustrated in Figure 1. A "$>$" indicates a segment that arises from the successively increasing sequence of segment colours, a "$<$" indicates a segment that arises from the successively decreasing sequence of segment colours, and a "$<>$" indicates a segment that arises from the intersection of both.

When $m$ is odd, the broadcast schedule is slightly different. Specifically, in the second broadcast slot of an odd phase, the assigned segments are chosen ac-

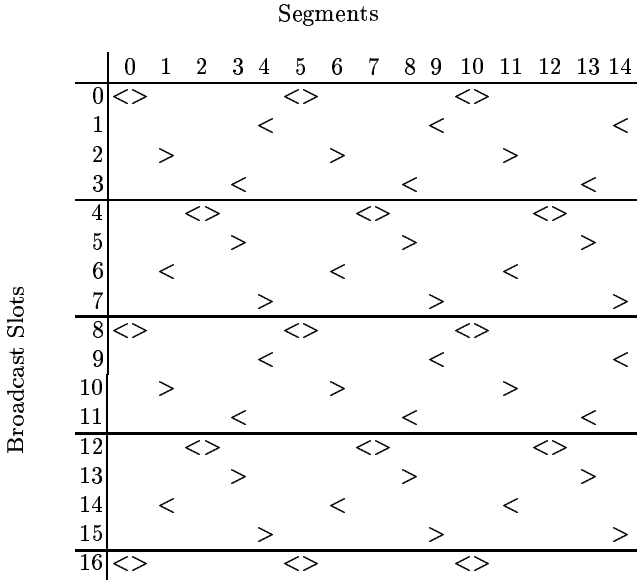


**Fig. 2.** The Broadcast Schedule for $m = 5$

cording to the increasing sequence of segment colours, rather than the decreasing sequence. For example, see the broadcast schedule for $m = 5$ that appears in Figure 2.

The complete algorithm is presented in Figure 3. The function `location()` returns the current location of the processor that called it.

There are two constraints we will impose on the parameters $G$ and $m$. The first constraint,

$$(m-1)u\sigma < G, \tag{1}$$

says that a processor cannot cross more than one segment boundary during a phase. The second constraint,

$$(m-1)G - 2[(m-2)u+1]\sigma \geq R + R', \tag{2}$$

implies that every broadcast that arrives at a process is received. This follows from the facts that the distance between the end of a segment and the beginning of the next segment of the same colour is $(m-1)G$, the time between the beginning of a phase and the end of its last broadcast slot is $(m-2)u+1$, and a processor can move at most distance $[(m-2)u+1]\sigma$ during this time.

If, in addition, there is a lower bound on the density of processors, then there is an upper bound on the time it takes for information to be propagated.

**Lemma 1.** *If there is never an interval of length* $[R - 3(m-1)u\sigma - 3G]/2$ *that contains no processors, then the speed of information propagation is at least*

```
at time π(m − 1)u, for any non-negative integer π,
%this is the beginning of phase π
    %determine the segment, i, in which processor p is located
    i ← ⌊location()/G⌋
    if p is the leader of segment i then
        %determine the colour, f, of the segments which are
        %assigned to the first broadcast slot in phase π
        f ← (π mod 2)⌊m/2⌋
        %determine to which broadcast slot, if any,
        %segment i is assigned in phase π
        j ← (i − f) mod m
        offset ← (π mod 2) AND (m mod 2)
        if j = 0 then slot ← 0
        else if j ≤ ⌊m/2⌋ − 1 then slot ← 2j− offset
        else if j > ⌊m/2⌋ then slot ← 2(m − j) − 1+ offset
        %broadcast in the assigned broadcast slot
        if j ≠ ⌊m/2⌋ then
            at time π(m − 1)u + slot, broadcast trajectory information
```

**Fig. 3.** The Information Maintenance Algorithm for Processor $p$

*$G/2u$, (i.e. information travels at least one segment for every two broadcast slots), in the worst case.*

Using this result, it is possible to prove that the algorithm enables processors to maintain trajectory information about nearby processors.

**Lemma 2.** *Suppose there is never an interval of length $[R-3(m-1)u\sigma-3G]/2$ that contains no processors, and all processors know the trajectory functions of the processors within distance $R + 2(m-1)u\sigma$ of themselves at the beginning of phase 0. Then all processors know the trajectory functions of the processors within distance $R + 2(m-1)u\sigma$ of themselves at the beginning of every phase.*

**Theorem 1.** *Suppose there is never an interval of length $[R-3(m-1)u\sigma-3G]/2$ that contains no processors, and all processors know the trajectory functions of the processors within distance $R + 2(m-1)u\sigma$ of themselves at the beginning of phase 0. Then all processors always know the trajectory functions of the processors within distance $R$ of themselves.*

For these results, it is not necessary that each processor repeatedly broadcast all the trajectory functions it knows about. It suffices that processor $p$ broadcasts the trajectory function of processor $q$ only if, at the beginning of the phase, there is another processor $q'$ that $p$ knows about, but may not know about $q$ (because $q'$ is more than distance $R + 2(m-1)u\sigma$ away from $q$) and will come within distance $R + 2(m-1)u\sigma$ of $q$ by the end of the phase.

Without the lower bound on the density of processors, but with a somewhat more stringent constraint between $G$ and $m$, there is a similar result, but with the definition of nearby being closer.

**Lemma 3.** *Suppose that $R \geq 4(m-1)u\sigma + 4G$ and all processors know the trajectory functions of the processors within distance $R - G - 5(m-1)u\sigma$ of themselves at the beginning of phase 0. Then all processors know the trajectory functions of the processors within distance $R - G - 5(m-1)u\sigma$ of themselves at the beginning of every phase.*

The proofs of these results, including details about what information to include in broadcasts, appear in [11] and will appear in the full version of the paper.

The constraints on the relative values of the parameters are easy to satisfy. For example, suppose the broadcast radius, $R$, and the interference radius, $R'$, are 250 meters and 550 meters, respectively, (which are their default values in the IEEE 802.11 standard), the length of a broadcast slot is 1 microsecond, and the time, $u$, between the beginning of successive broadcast slots is 100 microseconds. If $G = 30$ meters, $m = 31$ and $\sigma < 36,000$ km/hr, or if $G = 60$ meters, $m = 21$ and $\sigma < 4500$ km/hr, then all of the constraints are satisfied.

## 4 Directions for Further Work

There are many open questions that remain. Some of these address possible optimizations to the broadcast algorithm or the relaxation of various assumptions. Other questions are concerned with the model and various aspects of the problem.

In the previous section, a broadcast schedule was presented that avoids collisions and enables processors to quickly propagate information to other processors, provided the density of the processors never gets too small. Are there broadcast schedules that can propagate information faster? Are there broadcast schedules that can propagate information efficiently when processors can be further apart from one another? Perhaps an entirely different approach would be better. For example, allowing a small number of collisions might enable a more efficient algorithm to be obtained.

The density assumption ensures that processors moving towards one another will learn about each other's trajectory function before they are able to communicate directly, from processors located between them. However, if there is a large region containing no processors and the processors on the boundary of that region start moving towards one another, they cannot possibly get this information until the distance between them is less than the broadcast radius. Maybe the requirements for this situation should be relaxed so that processors only have to know about one another's trajectory functions until they have been close to one another for a sufficiently long period of time. However, this may have implications for the other parts of the algorithm and for applications that rely on knowledge of the locations of all nearby processors.

Perhaps processors on a boundary should broadcast more frequently so that they can be guaranteed to have exchanged information by the time they are close together. Because processors can move at different speeds, processors can

overtake one another and, consequently, which processor is on the boundary can change. In fact, if there are other processors close to the boundary, for example in the same segment as a processor on the boundary, it may not be best to have the actual processor that is on the boundary be responsible for performing these broadcasts, but instead have the leader of the segment perform them.

The assumption that each processor knows its entire future trajectory is probably the most unrealistic simplifying assumption we have made. It is more likely that a processor only knows its trajectory for some short period of time in the future, or that its trajectory function might change in response to events. Then processors would need to announce information about their trajectories either periodically or when they change.

It is also possible that a processor only has approximate information about its location. For example, a processor might not know its exact location, but only know the segment in which it is located at the beginning of the current phase, perhaps with some error in either direction.

How should processors announce movement between segments, updates to their trajectories, or other information (unrelated to their trajectories) that needs to be maintained, but can change at arbitrary times? One approach is to have the processors in a segment take turns being leader. Then they could announce any changes during their allocated broadcast slots. This could work well, provided the number of processors per segment is relatively small and care is taken to schedule processors when they change segments, so that they don't lose their opportunity to broadcast. If there are many processors in a segment, it might be better for the processors that have changes to announce to use a separate algorithm for transmitting this information to the leader of their segment, using an algorithm that adapts to the number of participating processors. For one segment, this problem is equivalent to broadcasting on a multiple access channel [6]. However, it is also necessary to avoid collisions with processors in nearby segments trying to do the same thing.

Our other simplifying assumption is that processors initially know the trajectories of all nearby processors. How can this be achieved, if processors are not initially far apart from one another? If processors have distinct identifiers in the range $\{1, \ldots, n\}$ and all processors begin at time 0, then it suffices for processor $i$ to broadcast its trajectory information in broadcast slot $i$, for $i = 1, \ldots, n$. Then the broadcast schedule can begin with broadcast slot $n + 1$. Is it possible to perform the initialization more efficiently? For example, could it help to have the time at which processors broadcast be a function of both their identifiers and location? When processors can begin at arbitrary times, the problem may be more difficult.

The broadcast schedule relies on the existence of a global clock. Common knowledge of time enables processors to determine the current locations of other locations from their trajectory functions. A global clock also allows processors to agree on the beginning of broadcast slots and the beginning of phases. A weaker assumption is the existence of a heartbeat, a beacon that transmits ticks at regular intervals, but does not provide the time. In particular, this enables

processors to construct local clocks that run at the same rate. If the ticks are sufficiently far apart, then our broadcast schedule still works. For example, when $m$ is even, all even phases are the same and all odd phases are the same. If a tick occurs once every two phases, then processors can agree, for example, to start odd phases immediately after ticks. However, if ticks occur more frequently, then processors may need to rely on a clock synchronization protocol to agree on the number of each phase, when phases begin, or when broadcast slots begin. For processors on boundaries or during initialization, ticks that occur too frequently can be especially problematic, because there has been no communication and, hence, no synchronization. In these cases, one idea is for a processor to choose the length of time between successive broadcasts as a function of its location.

Most mobile ad hoc networks consist of processors moving on a plane or in space. Can the same approach that was used, in Section 3, to maintain trajectory information be extended from one dimension to two dimensions by appropriately colouring a tiling of the plane with squares or hexagons? What about in three dimensions?

Our model assumes omnidirectional antennas, which broadcast messages both to the left and right simultaneously. With directional antennas, the problem changes significantly. It is possible to avoid some interference, for example two nearby processors that want to broadcast away from one another. It also may be more energy efficient to use a directional broadcast if a message only has to be sent in one direction. However, if a processor wants to send the same information in both directions, it will need to perform two broadcasts instead of one. In two and three dimensional environments, this issue is even more complex, because one has to consider the conical broadcast regions where messages will arrive at other processors and the larger regions surrounding them where messages may cause interference.

Finally, it would be useful to implement our algorithm for maintaining trajectory information about nearby processors, to see how it performs experimentally and to find good choices for the parameters $G$ and $m$. More generally, an experimental comparison of this algorithm with simpler approaches that rely on randomization would be interesting.

## References


1. R. Bar-Yehuda, O. Goldreich, and A. Itai, On the Time Complexity of Broadcast in Multi-Hop Radio Networks: An Exponential Gap Between Determinism and Randomization, *Journal of Computer and System Sciences*, 45(1), pp. 104-126, 1992.
2. V. Bharghavan, A. Demers, S. Shenker, and L. Zhang, MACAW: A Media Access Protocol for Wireless LANs, *Proceedings of the ACM SIGCOMM'94 Conference on Communications Architectures, Protocols, and Applications*, pp. 212-225, Aug./Sep. 1994.
3. J. Broch, D.A. Maltz, D.B. Johnson, and J. Jetcheva, A Performance Comparison of Multi-Hop Wireless Ad Hoc Network Routing Protocols, *Proc. ACM/IEEE Intl Conf. Mobile Computing and Networking*, pp. 85-97, Oct. 1998.

4. D. Bruschi, M. D. Pinto, Lower bounds for the broadcast problem in mobile radio networks, *Distributed Computing*, 10(3), pp. 129-135, Mar. 1997.
5. G. Calinescu, Computing 2-Hop Neighborhoods in Ad Hoc Wireless Networks, *Lecture Notes in Computer Science*, Vol. 2865, pp. 175-186, Jan. 2003.
6. R. Gallager, A Perspective on Multiaccess Channels, *IEEE Transactions on Information Theory*, 31(2), pp. 124-142, Mar. 1985.
7. N. Malpani, Y. Chen, N. Vaidya, and J. Welch, Distributed token circulation in mobile ad hoc networks, *IEEE Transactions on Mobile Computing*, 4(2), pp. 154-165, Mar./Apr. 2005.
8. N. Vaidya, *Medium Access Control Protocols for Wireless Networks*, manuscript, 2006.
9. R. Prakash, A. Schiper, M. Mohsin, D. Cavin, and Y. Sasson, A lower bound for broadcasting in mobile ad hoc networks, *EPFL Technical Report. IC/2004/37*, Jun. 2004.
10. J. Schiller, *Mobile Communications*, Addison-Wesley, 2000.
11. Sivaramakrishnan Subramanian, *Deterministic Knowledge about Nearby Nodes in a Mobile One Dimensional Wireless Environment*, M.Sc. Thesis, Department of Computer Science, Texas A&M University, Sep. 2006.