Using Virtual Mobile Nodes for Neighbor Detection in Proximity-Based Mobile Applications

Behnaz Bostanipour   Benoît Garbinato

Distributed Object Programming Laboratory
University of Lausanne
CH-1015 Lausanne, Switzerland
Email: {behnaz.bostanipour,benoit.garbinato}@unil.ch

***Abstract*—We introduce a time-limited neighbor detector service for mobile ad hoc networks, which enables a mobile device to detect other nearby devices in the past, present and up to some bounded time interval in the future. Our motivation lies in the emergence of a new trend of mobile applications known as proximity-based mobile applications, which enable a user to communicate with other users within some defined range and for a certain amount of time. Neighbor discovery is a fundamental requirement for these applications and is not restricted to the current neighbors but can include past or future neighbors. To implement the time-limited neighbor detector service, we apply an approach based on virtual mobile nodes. A virtual mobile node is an abstraction that is akin to a mobile node that travels in the network in a predefined trajectory. In practice it can be implemented by a set of mobile nodes based on a replicated state machine approach. In this paper, we assume that each node can accurately predict its own locations up to some bounded time interval in the future. Thus, we present a time-limited neighbor detector algorithm that uses a virtual mobile node that continuously travels in the network, collects the predicted locations of all nodes, performs the neighborhood matching between nodes and sends the list of neighbors to each node. We show that our algorithm correctly implements the time-limited neighbor detector service under a set of conditions.**

Keywords*—*Neighbor Detection; Virtual Mobile Node; Proximity-Based Broadcast; MANET; Distributed Systems; Smartphone;

I. Introduction

With the ubiquitous use of mobile devices and particularly smartphones, we face the emergence of a new type of distributed applications known as *Proximity-Based Mobile* (*PBM*) applications [5], [6]. These applications enable a user to interact with others in a defined range and for a given time duration e.g., for social networking ([26], [21], [1], [20]), gaming ([4]) and driving ([25]).

Discovering who is nearby is at the core of the PBM applications. It is the preliminary step for the further interactions between users. It also provides users with the opportunity to extend their social network from the people that they know to the people that they might not know but who are in their proximity. For instance, in a simple usage scenario of social networking applications such as WhosHere [26] or LoKast [21], a user first discovers other users in her proximity and then decides to view their profiles, start a chat with a user or a group of users or add them as friends. The discoverability, however, may not always be limited to the current neighbors. For instance, with the social networking applications such as iGroups [1] or LocoPing [20], a user can discover other users who were in her vicinity during a past event (e.g., concert, tradeshow, wedding) or simply during a past time interval (e.g., the past 24 hours). One can also think of applications that provide the user with the list of people who will be in her proximity up to some time interval in future and thus create the potential for new types of social interactions.

In this paper, we introduce a *time-limited neighbor detector* service that enables a user to discover the set of its neighbors in the past, present and up to some bounded time interval in the future in a mobile ad hoc network (MANET). We also present an algorithm that implements the time-limited neighbor detector using a virtual mobile node. A virtual mobile node is an abstraction that is already introduced in the literature and used for tasks such as routing or collecting data in MANETs [10], [11]. It is akin to a mobile node that travels in the network in a predefined trajectory known in advance to all nodes. In practice a virtual mobile node is emulated by a set of nodes in the network based on a replicated state machine approach. In this paper, we assume that each node can accurately predict its own locations up to some bounded time interval in the future. Thus, in the algorithm we use the virtual mobile node to continuously travel in the network, collect the the predicted locations of all nodes, perform the neighborhood matching between nodes and then send back to each node the list of its neighbors at current and future times. Each node also stores its neighbor set at any time so that later it will be able to be queried about its past neighbors.

In order to show that our algorithm correctly implements the time-limited neighbor detector service, we present a proof of correctness. In particular, we define the conditions under which the service can be correctly implemented by the algorithm.

To the best of our knowledge, this is the first paper that introduces a neighbor detector service that can detect future neighbors (although up to some time interval) in a MANET. It is also the first paper that uses an approach based on virtual mobile nodes for neighbor discovery.

The remainder of the paper is as follows. In Section II, we describe our system model and present some definitions. In Section III, we introduce the neighbor detector abstraction in two variants: the *perfect* variant which presents the ideal case of neighbor detection and is rather impractical and the *time-limited* variant. In Section IV, we present the implementation of the time-limited variant of the neighbor detector service. In order to do so, we first describe what is a virtual mobile node and how it can be used for the implementation of the time-limited neighbor detector. We then introduce an algorithm that implements the time-limited neighbor detector and we present a proof of correctness for the algorithm. In Section V, we discuss the related work. Finally, in Section VI, we discuss open problems and future work.

II. System Model and Definitions

We consider a mobile ad-hoc network (MANET) consisting of processes that move in a bounded region R of a two-dimensional plane. We use the terms *process* and *node* interchangeably. Each process is assigned a unique identifier. Processes can move on any continuous path, however there exists a known upper bound on their movement speed. A process is prone to *crash-reboot* failures: it can fail and recover at any time, and when the process recovers, it returns to its initial state. Moreover, a process may join or leave the system at any time (where a leave is treated as a failure). A process is *correct* if it never fails. Since we do not consider *Byzantine* behaviors, the information security and privacy issues are beyond the scope of this paper.

We assume the existence of a discrete global clock, i.e., the range T of the clock's ticks is the set of non-negative integers. We also assume the existence of a known bound on the relative processing speed. Each process in the system has access to a *global positioning service*, a *timely scoped broadcast service* and a *mobility predictor* service. In the following, we first introduce some definitions that are used throughout the paper. We then present each of the above mentioned services.

A. Definitions

Let p_i be a process in the network, we introduce the definitions given hereafter in order to capture proximity-based semantics.

- location denotes a geometric point in the two dimensional plane where region R is situated and can be expressed as tuple $(x,\ y)$.
- $loc_i(t)$ denotes the location occupied by process p_i at time $t \in T$.
- $Z_i(r,t)$ denotes all the locations inside or on the circle centered at $loc_i(t)$ with given radius r.
- r_d is called *neighbor detection radius*. It is a constant known by all processes in the network. Thus, $Z_i(r_d,t)$ presents the *neighborhood region* of p_i at time t.

B. Timely Scoped Broadcast Service

This communication service allows a process to send messages to all processes located within a given radius around it. Formally, the timely scoped broadcast service exposes the following primitives:

- BROADCAST(m, r): broadcasts a message m *in* $Z_i(r,t_b)$, where p_i is the sender and t_b is the time when the broadcast is invoked.
- RECEIVE(m, p_i): callback delivering a message m broadcast by process p_i.

The service satisfies the following properties.

> ***Timely Delivery.*** If a correct process p_i broadcasts a message m, there exists a bounded time duration Δ_{bcast} such that every correct process p_j delivers m in interval $[t_b;\ t_b+\Delta_{bcast}]$, if $loc_j(t) \in Z_i(r,t)$ for all $t \in [t_b;\ t_b+\Delta_{bcast}]$.
>
> ***No Duplication.*** No message is delivered more than once.
>
> ***No Creation.*** If some process p_j delivers a message m with sender p_i, then m was previously broadcast by process p_i.

For a detailed discussion regarding the implementability of this service in both the single-hop and the multi-hop cases see [5].

C. Global Positioning Service

This service allows each mobile process p_i to know its current location and the current time via the following functions:

- GETCURRENTTIME: returns the current global time. Formally, this implies that each process p_i has access to the global clock modeled in Section II.
- GETCURRENTLOCATION: returns the location occupied by p_i at the current global time.

In this paper, we do not provide any formal properties for this service. However, we assume that the outputs of its functions are exact. In practice, such a service would typically be implemented using NASA's GPS or ESA's Galileo space-based satellite navigation technologies.

D. Mobility Predictor Service

This service allows each mobile process p_i to predict its future locations up to some bounded time $\Delta_{predict}$ via the following function:

- PREDICTLOCATIONS: returns a hash map containing the predicted locations for p_i at each time t in the interval $[t_c;\ t_c+\Delta_{predict}]$ where t_c is the time when PREDICTLOCATIONS is invoked.

The service satisfies the following property.

> ***Strong Accuracy.*** Let $t \in [t_c;\ t_c+\Delta_{predict}]$ and l be a location, if p_i is predicted to be at l at time t, then $loc_i(t) = l$.

In order for the service to predict the locations of the process in the future, we assume that the mobility model applied by the processes is such that the future locations of a process can be predicted up to a certain $\Delta_{predict}$. For instance, if a process moves according to a mobility model with temporal dependency (such as Gauss-Markov Mobility Model or Smooth Random Mobility Model), its future locations can be predicted using its past locations [2]. In practice, the future location prediction can be achieved for example by taking into account the current trajectory (direction, speed,...) of the mobile device, possibly coupled with maps information.

III. The Neighbor Detector Abstraction

In this section, we introduce the *neighbor detector* abstraction in two variants. Formally, the neighbor detector service exposes the following primitive:

- PRESENT(t): returns $N_i(t)$ i.e., the set of processes detected as neighbors of p_i at time t, where p_i is the process that invokes PRESENT.

A. Neighbor Detector Variants

1) Perfect Neighbor Detector: By querying this variant of neighbor detector service, a mobile process is able to know the set of its neighbors at any time in the past, present or the future.

> ***Perfect Completeness.*** Let p_i and p_j be two correct processes, if $loc_j(t) \in Z_i(r_d, t)$, then $p_j \in N_i(t)$.
>
> ***Perfect Accuracy.*** Let p_i and p_j be two correct processes, if $p_j \in N_i(t)$, then $loc_j(t) \in Z_i(r_d, t)$.

Roughly speaking, the *perfect completeness* property requires a neighbor detector to detect any node that is in the neighborhood region at any time in the past, present or future. At the same time, the *perfect accuracy* property guarantees that no false detection occurs. Since in practice implementing the *perfect completeness* property requires an infinite knowledge of nodes' locations in the future, we consider a more practical variant of the neighbor detector abstraction called *the time-limited neighbor detector*. We introduce this variant hereafter.

2) Time-limited Neighbor Detector: Compared to the *perfect neighbor detector*, this variant has a different *completeness* property. However, its *accuracy* property is the same. We define its properties, below.

> ***Time-limited Completeness.*** Let p_i and p_j be two correct processes and Δ_{future} be a bounded time interval such that $\Delta_{future} > 0$, if $loc_j(t) \in Z_i(r_d, t)$ and $t \leq t_c + \Delta_{future}$, then $p_j \in N_i(t)$, where t_c is the time when PRESENT is invoked at p_i.
>
> ***Perfect Accuracy.*** Let p_i and p_j be two correct processes, if $p_j \in N_i(t)$, then $loc_j(t) \in Z_i(r_d, t)$.

Similarly to the *perfect completeness* property, the *time-limited completeness* property requires a neighbor detector to detect any node that is in the neighborhood region at any time in the past or present. However, its ability to detect future neighbors is limited by a bounded time duration Δ_{future}. More precisely, it only detects a node that is in the neighborhood region at any time from the time when PRESENT is invoked up to Δ_{future}. The *perfect accuracy* property also guarantees no false detection.[1]

IV. Implementing The Time-Limited Neighbor Detector

To implement the time-limited neighbor detector, our intuition is as follows: since each node knows its own locations up to $\Delta_{predict}$ in the future, we can think of a moving entity that travels through the network, collects the predicted locations of all nodes, performs the neighborhood matching between nodes and then sends back to each node the list of its neighbors at current and future times. For this reason, we rely on the concept of *Virtual Mobile Node* (VMN) introduced in [10].

In what follows we first describe what is a VMN and we add a VMN to the system model. We then introduce an algorithm that implements the time-limited neighbor detector in the new system model. Finally, we discuss the correctness of the algorithm.

[1] For simplicity's sake, we do not assume a time bound on the availability of the past neighborhood information at this point.

A. Virtual Mobile Node (VMN)

A VMN is an abstraction that is akin to a mobile node that travels in the network in a predefined trajectory. It is first introduced in [10]. One of the main motivations behind the design of a VMN abstraction is that if the motion of a mobile node could be predefined and known to all nodes in the network, the task of designing various algorithms for mobile ad hoc networks would be significantly simplified. Therefore, a VMN is designed such that it can execute any distributed algorithm that a node can execute, however, its movement can be predefined and be known in advance to all nodes in the network.

In [10] an algorithm called *Mobile Point Emulator* (*MPE*) is introduced which implements the VMN abstraction in a system model equivalent to the system model defined in this paper. Its approach to implement the VMN is based on a replicated state machine technique similar to the one originally presented in [19]. The algorithm defines a *mobile point* to be a circular region of a radius r_{mp}, that moves according to the predefined path of the VMN, i.e., at time t the center of the mobile point coincides with the preplanned location of the VMN at time t. The MPE replicates the state of the VMN at every node within the mobile point's region, modifying the set of replicas as the nodes move in and out of the mobile point's region. MPE uses a total-order broadcast service to ensure that the replicas are updated consistently. The total order broadcast service is built using a synchronous local broadcast service (equivalent to our timely scoped broadcast) and the synchronized clocks (obtained by using a service equivalent to our global positioning service).

Similarly to a node, the VMN can communicate with other nodes using the timely scoped broadcast service (or its equivalents). Moreover, the VMN is prone to *crash-reboot* failures. It can crash if and only if its trajectory takes it into a region unpopulated by any nodes (i.e., where there are no nodes to act as replicas), however, it recovers to its initial state as soon as it renters a dense area. A VMN is *correct* if it never fails.

B. Adding a VMN to the System Model

In this section, we add a VMN to the system model defined in Section II. The movement trajectory of this VMN is such that it can be used by our algorithm for the implementation of the time-limited neighbor detector. Note that we do not provide an implementation for the VMN, however, we assume that it can be implemented by the MPE algorithm sketched in Section IV-A.

The VMN communicates with the nodes in the network using the timely scoped broadcast service where the broadcast radius equals to a constant r_{VMN} known to all nodes. This constant is, in fact, defined by the VMN implementation (see [10]).

The movement of the VMN is defined by a predetermined trajectory function loc_{VMN} which maps every t in T to a location in region R. This function is known to all nodes in the network. According to loc_{VMN}, the VMN continuously scans the region R. The scans are arranged in the form of outward-returns. More precisely, let l_A, l_B be two distinct locations in R, then an outward scan starts at l_A and ends at l_B and a return scan starts at l_B and ends at l_A (see Fig. 1). Outward and

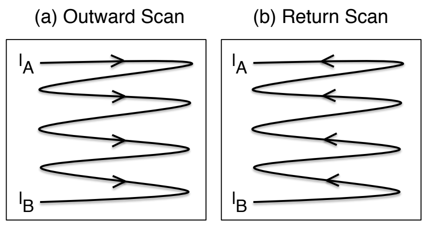

Fig. 1: VMN scans the region R

return scans alternate and the VMN uses exactly the same path in the outward and the return scans. The amount of time that the VMN spends in the outward scan is equal to the amount of time that it spends in the return scan. This time duration is denoted by Δ_{scan}.

In order for a scan to cover the entire region (and thus be useful for our time-limited neighbor detector algorithm), the trajectory function of the VMN ensures the following property.

> ***Scan Completeness.*** Let s be a scan (outward or return) and let t_{start} be the time when s is started, then the path traversed by the VMN during s is such that $\forall location \in R, \exists t \in [t_{start};\ t_{start} + \Delta_{scan}[$ such that $distance(loc_{VMN}(t), location) \leq r_{VMN}$.

C. A Time-limited Neighbor Detector Algorithm

The basic idea behind the algorithm is as follows: time is divided into rounds of duration Δ_{scan}. At each round k, the VMN scans the entire network, collects the predicted locations maps of all nodes, and also sends to each node the neighbors map of that node. The neighbors map is, in fact, generated by the VMN based on the predicted locations maps that are collected at round $k-1$. Note, however, that in the first round the VMN only collects the predicted locations maps and does not send any neighbors maps to the nodes.

The algorithm includes two parts: (1) a part that is executed on each physical node p_i in the network (Algorithm 1); (2) a part that is executed on the VMN (Algorithm 2). In the following, we discuss the algorithm in more detail.

Since the trajectory function of the VMN is globally known, each node p_i knows the value of Δ_{scan} and can determine when a new round begins (line 7). It can also calculate its distance to the VMN at any time. Thus, at each round p_i waits until its distance to the VMN becomes less than or equal to r_{VMN}. Then, if it has not already sent a message to the VMN in that round, it creates a message *msg* to send to the VMN (lines 9-11). This message encapsulates some parameters. Among these parameters the hash map $locs$ (also referred as the predicted locations map) is used to store the output of PREDICTLOCATIONS primitive of the mobility predictor service (line 12). The parameters t_{start} and t_{end} of *msg* store, respectively, the beginning and the end of the time interval for which the locations are predicted (lines 13-14).[2] In what follows, for simplicity's sake, we refer to the time interval $[msg.t_{start};\ msg.t_{end}]$ as the *epoch of msg* or E_{msg}. Once all parameters of *msg* are assigned their values, *msg* is broadcast within the radius r_{VMN}, so it can be received by the VMN (line 16).

When the VMN receives *msg* sent by p_i, it adds *msg* to the *currentRoundCollectedMsgs* set (lines 31-32). If the VMN is not in the first round and if it has collected a message from p_i in the previous round, then it creates a message *VMNmsg* to reply to p_i (lines 33-34). The message *VMNmsg* encapsulates a hash map *neighbors* (also referred as the neighbors map) and some other parameters. To obtain $neighbors$, first *pmsg* or the message collected from p_i in the previous round is found (line 35). Then, the function GETNEGHBORS is called (line 36), which uses the parameters of *pmsg* and messages collected from other nodes in the previous round to find neighbors of p_i during E_{pmsg} (lines 46-52). Thus, *neighbors* contains the set of detected neighbors of p_i at each time t during E_{pmsg}. The parameters t_{start} and t_{end} of *VMNmsg* store, respectively, the values of $pmsg.t_{start}$ and $pmsg.t_{end}$ (lines 37-38). Hence, E_{VMNmsg} is equal to E_{pmsg}. Finally, p_i is specified as the destination of *VMNmsg* and it is broadcast within the radius r_{VMN} so it can be received by p_i (lines 39-40). The process p_i has a hash map N that is used to store the set of its detected neighbors at any time t in T. Thus, when p_i receives a *VMNmsg* that is addressed to it, it looks through E_{VMNmsg} to find a time t at which $N(t)$ is undefined i.e., equals to $\perp$ (lines 18-21). Then, it assigns $VMNmsg.neighbors(t)$ to $N(t)$ (line 22).

D. Proof of Correctness

In this section we prove that the time-limited neighbor detector algorithm correctly implements the time-limited neighbor detector abstraction under certain conditions. In order to do so, we prove hereafter that the algorithm guarantees the properties of the time-limited neighbor detector abstraction defined in Section III-A2.

Note that in the following we use the notation $t_{s,k}$ to refer to the beginning time of a round k in the algorithm. Thus, for instance $t_{s,3}$ and $t_{s,1}$ denote, respectively, the beginning of round 3 and round 1 in the algorithm.

Theorem 1. *The time-limited neighbor detector algorithm satisfies the time-limited completeness property if the following conditions hold:*

(a) $\forall t \in T$, *at least one correct node resides in the circular region of radius r_{mp} around $loc_{\text{VMN}}(t)$. This condition, in fact, guarantees that the VMN is correct.*

(b) *In each round, the time elapsed from the sending of the predicted locations map to the VMN by a process p_i until the reception at p_i of the neighbors map sent by the VMN, is negligible.*

(c) $\Delta_{predict} \geq \Delta_{future} + 3 \times \Delta_{scan} - 2$.

(d) *Let* PRESENT*(t), then $t_c \geq t_{s,3}$. Recall that t_c is the time when* PRESENT*(t) is invoked. Thus, this condition guarantees that* PRESENT*(t) is called in a round k such that $k \geq 3$.*

(e) *Let* PRESENT*(t), then $t \geq t_{s,1} + \Delta_{scan} - 1$.*

Proof: We show that with the worst case scenario the algorithm satisfies the *time-limited completeness* property if Conditions (a), (b), (c), (d), (e) hold.

Let PRESENT(t) be invoked at process p_i in a round k i.e., t_c be in round k. We consider the worst case scenario according to which in round k the distance between p_i and

[2]We assume no clock tick between lines 12-13.

Algorithm 1 Time-limited Neighbor Detector Algorithm at Process p_i

```
1: initialisation:
2:    $noMsgSentInThisRound \leftarrow true$
3:    for all $t \in T$ do
4:       $N(t) \leftarrow \perp$

5: PRESENT($t$)
6:    return $N(t)$

7: every $\Delta_{scan}$ do                    {$p_i$ knows the function $loc_{VMN}$ and can calculate $\Delta_{scan}$}
8:    $noMsgSentInThisRound \leftarrow true$

9: upon DISTANCE(GETCURRENTLOCATION, $loc_{VMN}$(GETCURRENTTIME)) $\leq r_{VMN}$ do
10:    if $noMsgSentInThisRound$ then
11:       $msg \leftarrow \perp$                    {a message $msg$ is created to encapsulate some parameters}
12:       $msg.locs \leftarrow$ PREDICTLOCATIONS
13:       $msg.t_{start} \leftarrow$ GETCURRENTTIME
14:       $msg.t_{end} \leftarrow msg.t_{start} + \Delta_{predict}$
15:       $msg.sender \leftarrow p_i$
16:       trigger BROADCAST($msg, r_{VMN}$)
17:       $noMsgSentInThisRound \leftarrow false$

18: upon RECEIVE($VMNmsg, VMN$) do
19:    if $VMNmsg.destination = p_i$ then
20:       for all $t \in [VMNmsg.t_{start},\ VMNmsg.t_{end}]$ do
21:          if $N(t) = \perp$ then
22:             $N(t) \leftarrow VMNmsg.neighbors(t)$
```

Algorithm 2 Time-limited Neighbor Detector Algorithm at the VMN

```
23: initialisation:
24:    $k \leftarrow 1$                    {$k$ stores the round number}
25:    $currentRoundCollectedMsgs \leftarrow \emptyset$
26:    $previousRoundCollectedMsgs \leftarrow \emptyset$

27: every $\Delta_{scan}$ do                    {the VMN knows the function $loc_{VMN}$ and can calculate $\Delta_{scan}$}
28:    $k \leftarrow k + 1$
29:    $previousRoundCollectedMsgs \leftarrow currentRoundCollectedMsgs$
30:    $currentRoundCollectedMsgs \leftarrow \emptyset$

31: upon RECEIVE($msg, p_i$) do
32:    $currentRoundCollectedMsgs \leftarrow currentRoundCollectedMsgs \cup \{msg\}$
33:    if $k > 1 \wedge$ GETPREVIOUSMSG($p_i$) $\neq \perp$ then
34:       $VMNmsg \leftarrow \perp$
35:       $pmsg \leftarrow$ GETPREVIOUSMSG($p_i$)
36:       $VMNmsg.neighbors \leftarrow$ GETNEGHBORS($pmsg$)
37:       $VMNmsg.t_{start} \leftarrow pmsg.t_{start}$
38:       $VMNmsg.t_{end} \leftarrow pmsg.t_{end}$
39:       $VMNmsg.destination \leftarrow p_i$
40:       trigger BROADCAST($VMNmsg, r_{VMN}$)

41: function GETPREVIOUSMSG($p_i$)
42:    for all $m \in previousRoundCollectedMsgs$ do
43:       if $m.sender = p_i$ then
44:          return $m$
45:    return $\perp$

46: function GETNEGHBORS($pmsg$)
47:    for all $t \in [pmsg.t_{start},\ pmsg.t_{end}]$ do
48:       $neighbors(t) \leftarrow \emptyset$
49:       for all $m \in previousRoundCollectedMsgs$ do
50:          if $m.sender \neq pmsg.sender \wedge$ DISTANCE($m.locs(t), pmsg.locs(t)$) $\leq r_d$ then
51:             $neighbors(t) \leftarrow neighbors(t) \cup \{m.sender\}$
52:    return $neighbors$
```

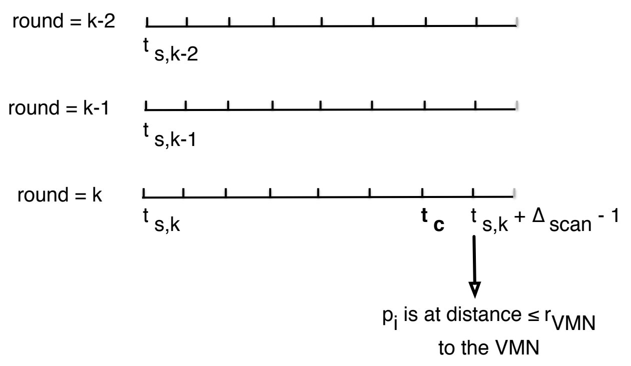

Fig. 2: The worst case scenario

the VMN becomes less than or equal to r_{VMN} only at the last clock tick of the round i.e., at $t_{s,k} + \Delta_{scan} - 1$ and $t_c = t_{s,k} + \Delta_{scan} - 2$ (see Fig. 2). In addition, for this scenario we assume that $\Delta_{predict} = \Delta_{future} + 3 \times \Delta_{scan} - 2$. Thus, we show the proof in the two following cases: (1) for $t_c \le t \le t_c + \Delta_{future}$; and (2) for $t_{s,1} + \Delta_{scan} - 1 \le t < t_c$.

(1) ***For*** $\boldsymbol{t_c \le t \le t_c + \Delta_{future}}$. In the described scenario, the neighbors map sent by the VMN at round k cannot be used by p_i for the neighbor detection at time t_c. The reason is that this neighbors map is only received by p_i at $t_{s,k} + \Delta_{scan} - 1$, i.e., after t_c. According to Condition (d), k is at least equal to 3. Thereby, at time t_c, the process p_i can rely on the neighbors map that it has received in round $k-1$ for neighbor detection.[3] According to the algorithm, the neighbors map that p_i has received in round $k-1$ is made based on the predicted locations maps collected at round $k-2$.
Since $\Delta_{predict} = \Delta_{future} + 3 \times \Delta_{scan} - 2$, predicted locations maps of p_i and p_j collected at round $k-2$ contain the predicted locations for time interval $[t_c;\ t_c + \Delta_{future}]$. In fact, according to the *scan completeness* property of the VMN scans, in a round, for each node there exists a time when its distance to the VMN becomes less than or equal to r_{VMN}. According to the algorithm, as soon as a node realizes that it is within distance r_{VMN} to the VMN, it sends its predicted locations map to the VMN. Thus, in round $k-2$ both p_i and p_j send their predicted locations maps to the VMN. Moreover, in round $k-2$, if a node is within distance r_{VMN} to the VMN at the earliest possible time (i.e., at time $t_{s,k-2}$ which is the beginning of the round), its predicted locations map is defined for time interval $[t_{s,k-2};\ t_{s,k-2} + \Delta_{predict}] = [t_{s,k-2};\ t_c + \Delta_{future}]$. Therefore, regardless of the time when p_i and p_j are within distance r_{VMN} to the VMN, their predicted locations maps in round $k-2$ contain predictions for time interval $[t_c;\ t_c + \Delta_{future}]$. Considering Conditions (a) and (b), plus the *timely delivery* property of the underlying timely scoped broadcast, the communication between the VMN and a node is reliable and with negligible delay. Therefore, the messages sent by p_i and p_j in round $k-2$ are received by the VMN in round $k-2$.
The *scan completeness* property of the VMN scans guarantees that the VMN would be within r_{VMN} of p_i at some time in round $k-1$. At this time, the VMN sends to p_i its neighbors map which is made based on the predicted locations maps collected at round $k-2$. The *strong accuracy* property of the mobility predictor service guarantees that the location predictions of p_i and p_j in their corresponding predicted locations maps are accurate. The function GETNEIGHBORS (lines 46-52) also guarantees the correct neighbor matching between nodes. According to the algorithm, the neighbors map sent to p_i at round $k-1$ is defined for the same time interval as the predicted locations map collected from p_i at round $k-2$. This means that the neighbors map has the information for time interval $[t_c;\ t_c + \Delta_{future}]$. Thus, based on the arguments above, if $loc_j(t) \in Z_i(r_d, t)$, p_j is indicated as a neighbor of p_i at time t in the neighbors map sent to p_i in round $k-1$. Again Conditions (a), (b) and the *timely delivery* property of the timely scoped broadcast guarantee that the neighbors map sent to p_i by the VMN in round $k-1$ is received by p_i in round $k-1$. Then, according to the algorithm, the values of the neighbors map for time interval $[t_c;\ t_c + \Delta_{future}]$ are assigned to N at p_i (line 22). Hence, if $loc_j(t) \in Z_i(r_d, t)$, then $p_j \in N_i(t)$.

(2) ***For*** $\boldsymbol{t_{s,1} + \Delta_{scan} - 1 \le t < t_c}$. We prove this case by induction.
Basis. We start with the smallest possible value for k, which according to Condition (d) is $k = 3$. If $k = 3$, p_i can only rely on the neighbors map that it has received in round 2 for neighbor detection. According to the algorithm the neighbors map sent to p_i at round 2 is defined for the same time interval as the predicted locations map collected from p_i at round 1. So, even if at round 1, p_i is within distance r_{VMN} to the VMN at the latest possible time (i.e., at time $t_{s,1} + \Delta_{scan} - 1$ which is the end of the round), its neighbors map in round 2 is defined for time interval $[t_{s,1} + \Delta_{scan} - 1;\ t_{s,1} + \Delta_{scan} - 1 + \Delta_{predict}] = [t_{s,1} + \Delta_{scan} - 1;\ t_{s,1} + 4 \times \Delta_{scan} + \Delta_{future} - 3]$. Thus, the neighbors map contains the neighborhood information for time interval $[t_{s,1} + \Delta_{scan} - 1;\ t_c[$ and by the same reasoning that in the case (1) above, we conclude that when $k = 3$, the *time-limited completeness* is guaranteed for $t_{s,1} + \Delta_{scan} - 1 \le t < t_c$.
Inductive Step. We consider some round $k > 3$. Then, we show that if the *time-limited completeness* is guaranteed for $t_{s,1} + \Delta_{scan} - 1 \le t < t_c$ in round k, then it is also guaranteed in round $k+1$. From the case (1) above, we know that in round k the *time-limited completeness* is guaranteed for $t = t_c$. Considering this fact and the induction hypothesis, we can say that in round k, the *time-limited completeness* is guaranteed for $t_{s,1} + \Delta_{scan} - 1 \le t \le t_c$ which is equivalent to say that it is guaranteed for $t_{s,1} + \Delta_{scan} - 1 \le t \le t_{s,k} + \Delta_{scan} - 2$ by replacing t_c with $t_{s,k} + \Delta_{scan} - 2$. Thus, since the algorithm continuously stores the neighborhood information (lines 21-22), in round $k+1$ the *time-limited completeness* is already guaranteed for $t_{s,1} + \Delta_{scan} - 1 \le t \le t_{s,k} + \Delta_{scan} - 2$. Also, we know that in round k, p_i receives a new neighbors map at time $t_{s,k} + \Delta_{scan} - 1$. This new neighbors map is made based on the predicted locations map collected at round $k-1$ and in the worst case contains the neighborhood information for the time interval $[t_{s,k-1};\ t_{s,k-1} + \Delta_{predict}]$. The time interval $[t_{s,k-1};\ t_{s,k-1} + \Delta_{predict}]$ can be written as $[t_{s,k-1};\ t_c + \Delta_{future}]$ where t_c is the t_c of round $k+1$.

[3]Note that according to the algorithm, no neighbors map is distributed by the VMN in round 1.

Thus, the neighbors map contains the neighborhood information for time interval $[t_{s,k} + \Delta_{scan} - 1;\ t_c[$. Thereby, by the same reasoning that in the case (1) above, we know that in round $k+1$, the *time-limited completeness* is also guaranteed for $t_{s,k}+\Delta_{scan}-1 \leq t < t_c$. Finally, since we showed that in round $k+1$, the *time-limited completeness* is guaranteed for $t_{s,1}+\Delta_{scan}-1 \leq t \leq t_{s,k}+\Delta_{scan}-2$ and for $t_{s,k}+\Delta_{scan}-1 \leq t < t_c$, we conclude that it is guaranteed for $t_{s,1}+\Delta_{scan}-1 \leq t < t_c$.

■

Theorem 2. *The time-limited neighbor detector algorithm satisfies the perfect accuracy property.*

Proof: According to the algorithm, if $p_j \in N_i(t)$, there exists a round during which p_i has received a *VMNmsg* with a *neighbors* map parameter such that $p_j \in$ *VMNmsg*.$neighbors(t)$. Since p_i verifies the destination of each *VMNmsg* that it receives (line 19), we know that p_i ignores a *VMNmsg*s that is not addressed to it. Also, according to the *no creation* property of the timely scoped broadcast service, we know that the *VMNmsg* is in fact broadcast by the VMN. According to the algorithm, the *neighbors* map of the *VMNmsg* is created by calling the function GETNEGHBORS (lines 46-52). This function performs the neighbor matching based on $locs$ (or the predicted locations maps) parameter of *msg* messages collected from nodes in the previous round. The parameter *sender* of each *msg* guarantees that there is no error regarding the sender of a *msg*. The *strong accuracy* property of the mobility predictor service guarantees that the location predictions in $locs$ are accurate. In addition, the function GETNEGHBORS only detects p_j as a neighbor of p_i at time t if the distance between their predicted locations at t is less than or equal to r_d (lines 50-51). Finally, the *no creation* property of the timely scoped broadcast service guarantees that each *msg* collected by the VMN from a node is indeed sent by that node. Therefore if $p_j \in N_i(t)$, then $loc_j(t) \in Z_i(r_d, t)$. ■

Theorem 3. *The time-limited neighbor detector algorithm correctly implements the time-limited neighbor detector abstraction if the following conditions hold:*

(a) *$\forall t \in T$, at least one correct node resides in the circular region of radius r_{mp} around $loc_{\text{VMN}}(t)$. This condition, in fact, guarantees that the VMN is correct.*

(b) *In each round, the time elapsed from the sending of the predicted locations map to the VMN by a process p_i until the reception at p_i of the neighbors map sent by the VMN, is negligible.*

(c) $\Delta_{predict} \geq \Delta_{future} + 3 \times \Delta_{scan} - 2$.

(d) *Let* PRESENT*(t), then $t_c \geq t_{s,3}$. Recall that t_c is the time when* PRESENT*(t) is invoked. Thus, this condition guarantees that* PRESENT*(t) is called in a round k such that $k \geq 3$.*

(e) *Let* PRESENT*(t), then $t \geq t_{s,1} + \Delta_{scan} - 1$.*

Proof: According to Theorem 1, the algorithm guarantees the *time-limited completeness* property of the time-limited neighbor detector if Conditions (a), (b), (c), (d), (e) hold. In addition, according to Theorem 2, the algorithm guarantees the *perfect accuracy* property of the time-limited neighbor detector. Therefore, the algorithm correctly implements the time-limited neighbor detector abstraction if Conditions (a), (b), (c), (d), (e) hold. ■

V. Related Work

Neighbor detection in ad hoc networks is usually studied as a building block for applications such as routing, leader election, group management and localization. Many of the existing neighbor detection algorithms belong to the *hello protocols* family [22], [24], [13], [17], [3], [16], [15], [6]. They are based on the *basic hello protocol* first described in *Open Shortest Path First* (*OSPF*) routing protocol [23]. It works as follows: nodes periodically send *hello* messages to announce their presence to close nodes, and maintain a neighbor set. The sending frequency is denoted by f_{hello}. If a *hello* message is not received from a neighbor for a predefined amount of time, then that neighbor is discarded from the neighbor set. The problem with this approach is that if f_{hello} is too low (with respect to the speed of the nodes), then the neighbor set becomes quickly obsolete. On the other hand, if it is too high, the neighbor set remains up to date but it causes a significant waste of communication bandwidth and energy [16]. However, finding the optimal f_{hello} is not obvious and the existing solutions cannot ideally solve this problem. Moreover, the *hello protocols* usually provide only the set of current neighbors and they do not satisfy any formal guarantees.

Nevertheless, in the literature there exist schemes that use different approaches than the *hello* broadcast for neighbor detection [8], [9]. For instance, in [9], a reliable neighbor detection abstraction is defined that establishes links over which message delivery is guaranteed. The authors present two region-based neighbor detection algorithms which implement the abstraction with different link establishment guarantees. The algorithms are implemented on top of a Medium Access Control (MAC) layer which provides upper bounds on the time for message delivery. The main idea behind the first algorithm is that a node sends a *join* message some time after entering a new region to establish communication links. It also sends a *leave* message some time before leaving a region to inform the other nodes so that they can tear down their corresponding link with that node. To guarantee that these notification messages reach their destination despite the continuos motion of nodes, the authors define the time limits for a node to send the *join* and the *leave* messages. These time limits are obtained using the timing guarantees of the underlying MAC layer. Since a node should send a *leave* message some time before it actually leaves a region, the algorithm assumes that a node's trajectory function is known to that node with enough anticipation to communicate with other nodes before leaving the region. The first algorithm does not guarantee the communication links when nodes are moving quickly across region boundaries. Thus, the authors introduce a second algorithm. In this new algorithm they apply a technique which overlays multiple region partitions, associating with each region partition an instance of the first algorithm. The output of each instance is then composed such that it guarantees the communication links even when nodes are moving across region boundaries. The approach applied in [9] for neighbor detection is interesting because it uses a relatively lower number of message broadcast compared to the *hello protocols*. Similarly to our work, this approach also uses the knowledge of nodes about their future locations for the neighbor detection. However, contrary to our work, no future neighbor detection is defined and only the current neighbor detection is guaranteed.

We believe that our work is the first attempt to use a virtual node for neighbor detection. The idea of using mobile entities with predefined trajectory to facilitate the design of algorithms for mobile networks was first introduced by Hatzis et al. in [14]. They define the notion of a *compulsory* protocol which requires a subset of the mobile nodes to move in a predefined way. They also present an efficient compulsory protocol for leader election. In [7] and [18] routing protocols for MANETs using compulsory protocols are introduced. In [10], [11], several basic algorithms that use a VMN to solve various problems are briefly presented. These algorithms address the problems such as routing, collecting and evaluating data in mobile ad hoc sensor networks and some other general problems such as group communication and atomic memory.

VI. Conclusion

We have introduced a time-limited neighbor detector service for MANETs. By querying this service a mobile process can know the set of its neighbors at any time in the past, present and up to some bounded time interval in the future. We presented an algorithm that implements this service using a virtual mobile node. Our algorithm is shown to implement correctly the neighbor detector service under certain conditions.

To the best of our knowledge, this is the first work that introduces a neighbor detector service that can detect future neighbors of a node in MANETs. It is also the first paper that uses an approach based on virtual nodes for neighbor detection.

Yet, some issues remain open, which we might consider as future work. For instance, in this work we assumed the *strong accuracy property* for the mobility predictor service. However, in practice such predictions are approximative. Thus, it is interesting to devise an algorithm that can guarantee a certain percentage of correct neighbor detection based on the approximative location predictions. Moreover, one of the conditions for correctness of our algorithm is the correctness of the VMN which is guaranteed if the circular region of radius r_{mp} around the location of the VMN remains populated all the time. In fact, as also claimed in [11], in the real world this density assumption is reasonable in many cases. For instance, there exist regions (specially in urban areas) that are almost always populated—such as main squares in a downtown area. However, to guarantee the neighbor detection even in less populated regions, we can think of using another type of virtual nodes called *autonomous virtual mobile nodes* [12]. This type of virtual nodes can move autonomously, choosing to change their path based on their own state and inputs from the environment. For instance, if the area in their paths appears deserted, they can change their path to the more populated areas.

Finally, as also claimed in [10], implementing a VMN is expensive. Therefore, it would be interesting to experiment our neighbor detection algorithm with a real implementation to know if the utility outweighs the implementation overhead and possibly to come up with optimizations.

Acknowledgment

This research is partially funded by the Swiss National Science Foundation in the context of Project 200021-140762.

References

[1] Apple's iGroups. http://www.patentlyapple.com/patently-apple/2010/03/igroups-apples-new-iphone-social-app-in-development.html.

[2] F. Bai and A. Helmy, A survey of mobility modeling and analysis in wireless ad hoc networks, Book Chapter In *Wireless ad hoc and sensor networks*, Springer, 2006.

[3] M. Bakht, M. Trower, and R. Kravets. Searchlight: helping mobile devices find their neighbors. In *ACM SIGOPS Oper. Syst.*, vol. 45, no. 3, pp. 71–76, 2012.

[4] Best iPhone bluetooth games. http://iphone.mob.org/genre/multipleer/

[5] B. Bostanipour, B. Garbinato and A. Holzer, Spotcast – A communication abstraction for proximity-based mobile applications, In *Proc. IEEE NCA'12*, pp. 121–129, 2012.

[6] B. Bostanipour and B. Garbinato, Improving neighbor detection for proximity-based mobile applications, In *Proc. IEEE NCA'13*, pp. 177–182, 2013.

[7] I. Chatzigiannakis, S. E. Nikoletseas and P. G. Spirakis: An Efficient Communication Strategy for Ad-hoc Mobile Networks. In *Proc. DISC'01*: pp. 285-299, 2001.

[8] A. Cornejo, S. Viqar, J. L. Welch, Reliable neighbor discovery for mobile ad hoc networks. In *Proc. DIALM-PODC'10*, pp. 63-72, 2010.

[9] A. Cornejo, S. Viqar and J. L. Welch, Reliable neighbor discovery for mobile ad hoc networks. In *Ad Hoc Networks*. 12 (January 2014), pp. 259–277, 2014.

[10] S. Dolev, S. Gilbert, N. A. Lynch, E. Schiller, A. A. Shvartsman, J. L. Welch: Virtual Mobile Nodes for Mobile Ad Hoc Networks. In *Proc. DISC'04*, pp. 230–244, 2004.

[11] S. Dolev, S. Gilbert, N. A. Lynch, E. Schiller, A. A. Shvartsman, J. L. Welch. Virtual mobile nodes for mobile ad hoc networks. Tech Report LCS-TR-937, MIT, 2004.

[12] S. Dolev, S. Gilbert, E. Schiller, A. A. Shvartsman, J. L. Welch: Autonomous virtual mobile nodes. In *DIALM-POMC*, pp. 62-69, 2005.

[13] P. Dutta, D. Culler, Practical asynchronous neighbor discovery and rendezvous for mobile sensing applications. In *Proc. SenSys'08*, 2008.

[14] K. P. Hatzis, G. P. Pentaris, P. G. Spirakis, V. T. Tampakas and R. B. Tan. Fundamental control algorithms in mobile networks. In *Proc. ACM SPAA'99*, 1999.

[15] D. He, N. Mitton, and D. Simplot-Ryl, An energy efficient adaptive HELLO algorithm for mobile ad hoc networks, In *Proc. ACM MSWiM'13*, pp. 65–72, 2013.

[16] F. Ingelrest, N. Mitton, and D. Simplot-Ryl, A turnover based adaptive hello protocol for mobile ad hoc and sensor networks, In *Proc. MASCOTS'07*, pp. 9–14, 2007.

[17] A. Kandhalu, K. Lakshmanan, and R. R. Rajkumar, U-connect: a low-latency energy-efficient asynchronous neighbor discovery protocol, In *IPSN'10*, pp. 350–361, 2010.

[18] Q. Li and D. Rus. Sending messages to mobile users in disconnected ad-hoc wireless networks. In *Proc. MobiCom*, 2000.

[19] L. Lamport. Time, clocks, and the ordering of events in a distributed system. *Communications of the ACM*, 21(7):558–565, 1978.

[20] LocoPing. http://www.locoping.com/

[21] LoKast. http://www.lokast.com/

[22] M. J. McGlynn and S. A. Borbash. Birthday protocols for low energy deployment and flexible neighbor discovery in ad hoc wireless networks. In *Proc. ACM MobiHoc'01*, pp. 137–145, 2001.

[23] J. Moy, OSPF – Open Shortest Path First, RFC 1583, 1994.

[24] G. Wattenhofer, G. Alonso, E. Kranakis, and P. Widmayer. Randomized protocols for node discovery in ad hoc, single broadcast channel networks. In *Proc. IPDPS'03*, 2003.

[25] Waze. https://www.waze.com/en/

[26] WhosHere. http://whoshere.net/