Neighbor Detection Based on Multiple Virtual Mobile Nodes

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 an algorithm that implements a time-limited neighbor detector service in mobile ad hoc networks. The time-limited neighbor detector enables a mobile device to detect nearby devices in the past, present and up to some bounded time interval in the future. In particular, it can be used by a new trend of mobile applications known as proximity-based mobile applications. To implement the neighbor detector, our algorithm uses $n = 2^k$ virtual mobile nodes where k is a non-negative integer. 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. Our algorithm implements the neighbor detector for nodes located in a circular region. We assume that each node can accurately predict its own locations up to some bounded time interval $\Delta_{predict}$ in the future. The key idea of the algorithm is that the virtual mobile nodes regularly collect location predictions of nodes from different subregions, meet to share what they have collected with each other and then distribute the collected location predictions to nodes. Thus, each node can use the distributed location predictions for neighbor detection. We show that our algorithm is correct under certain conditions. Compared to a solution that works with a single virtual mobile node, our algorithm has a main advantage: as n grows, it remains correct with smaller values of $\Delta_{predict}$. This feature makes the real world implementation of the neighbor detector more feasible. In fact, although there exist different approaches to predict the future locations of a node, usually predictions tend to become less accurate as $\Delta_{predict}$ increases.**

Keywords*—*Neighbor Detection; Virtual Mobile Node; MANET;

I. Introduction

With the increasing use of mobile devices and particularly smartphones, a new trend of distributed applications known as *Proximity-Based Mobile* (*PBM*) applications has recently emerged [1], [2], [3], [4]. These applications enable a user to interact with others in a defined range and for a given time duration e.g., for social networking (WhosHere, LoKast, iGroups, LocoPing) or driving (Waze).[1]

Discovering who is nearby is the basic requirement of various PBM applications. It is the preliminary step for further interactions between users. It also enables users 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, with the social networking applications such as WhosHere or LoKast, a user first discovers others in her proximity and then decides to view their profiles, start a chat with them 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 or LocoPing, a user can discover others 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 the future and thus create the potential for new types of social interactions [3].

To address the requirements of neighbor detection in PBM applications, in a previous work [3], we introduced a *time-limited neighbor detector* service and proposed a simple algorithm for it. This service enables a device 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).

In this paper, we present a more advanced and general algorithm that implements the time-limited neighbor detector using $n = 2^k$ virtual mobile nodes where k is a non-negative integer. A virtual mobile node is an abstraction that was already introduced in the literature and used for tasks such as routing or collecting data in MANETs [6], [7]. 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 using a replicated state machine approach.

Our algorithm implements the neighbor detector for nodes located in a circular region. We assume that each node can accurately predict its locations up to some time interval $\Delta_{predict}$ in the future. Thus, the region is divided into n equal subregions and each subregion is associated with one virtual mobile node. Each virtual mobile node regularly collects the location predictions from the nodes in its subregion and meets other virtual mobile nodes to share what it has collected with them. After the sharing, each virtual mobile node has the location predictions collected from the entire region, which it distributes to the nodes in its subregion. In this way, each node can find its neighbors at current and future times based on the collected location predictions that it receives from a virtual mobile node. It can also store the collected location predictions so it can be queried about its past neighbors. We show that our algorithm is correct under certain conditions.

To the best of our knowledge, our algorithm is the first that uses multiple virtual mobile nodes to implement the time-limited neighbor detector. Our algorithm has a main advantage over a solution that works with a single virtual node [3]: as n grows, it remains correct with smaller values of $\Delta_{predict}$. This

[1] whoshere.net www.lokast.com www.locoping.com www.waze.com tinyurl.com/Apple-iGroups

feature makes the real world implementation of the neighbor detector more feasible. In fact, although there exist different approaches to predict the future locations of a node, usually predictions tend to become less accurate as $\Delta_{predict}$ increases.

The remainder of the paper is as follows. In Section II, we describe our system model and introduce some definitions. In Section III, we present the time-limited neighbor detector service. In Section IV, we present an implementation of the time-limited neighbor detector service. In order to do so, we first describe what a virtual mobile node is and how it can be used for the implementation of the neighbor detector. We then add n virtual mobile nodes to the system model and introduce our algorithm that implements the neighbor detector in the new system model. We sketch a correctness proof for the algorithm (for complete proof see [5]). In particular, we define the value of $\Delta_{predict}$ for which the algorithm is correct. Then, we show the evolution of this value as n grows. Finally, we discuss related work in Section V before concluding in Section VI.

II. System Model and Definitions

We consider a mobile ad-hoc network (MANET) consisting of a set P of processes that move in a two dimensional plane. We use the terms *process*, *node* and *real node* interchangeably. Each process has a unique identifier. Processes can move on any continuous path, however there exists a known upper bound on their motion 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. A process is *correct* if it never fails. 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 *timely scoped broadcast service*, a *global positioning service* and a *mobility predictor* service. In the following, we first introduce some definitions. We then present each of the above mentioned services.

A. Definitions

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

– A *location* denotes a geometric point in the two dimensional plane and can be expressed as tuple (x, y).

– $loc(p_i, t)$ denotes the location occupied by p_i at time $t \in T$.

– $Z(p_i, r, t)$ denotes all the locations inside or on the circle centered at $loc(p_i, t)$ with given radius r.

– r_d is called the *neighbor detection radius*. It is a constant known by all processes in the network. So, $Z(p_i, r_d, t)$ presents the *neighborhood region* of p_i at time t.

B. Timely Scoped Broadcast Service

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

• BROADCAST(m, r): broadcasts a message m *in* $Z(p_i, r, t_b)$, 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 $loc(p_j, t) \in Z(p_i, r, t)$ 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 about the implementability of this service in both the single-hop and the multi-hop cases see [1].

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.

• GETCURRENTLOCATION: returns the location occupied by p_i at the current global time.

In practice, such a service would typically be implemented using NASA's GPS space-based navigation technology.

D. Mobility Predictor Service

This service allows each mobile process p_i to predict its future locations up to some bounded time duration $\Delta_{predict}$ 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 $loc(p_i, t) = l$.

In order for the service to predict the locations of a process, we assume that the processes move in a way that their future locations can be predicted up to a certain $\Delta_{predict}$ e.g., if a process moves according to a mobility model with temporal dependency (such as Gauss-Markov Mobility Model), its future locations can be predicted using its past locations.

III. The Time-limited Neighbor Detector Service

This service was first introduced in [3]. Intuitively, it allows a process to know its neighbors at a given time. Formally, it exposes the following primitive:

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

The service satisfies the two following properties.

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 $loc(p_j, t) \in Z(p_i, r_d, t)$ and $t \leq t_c + \Delta_{future}$, then $p_j \in N(p_i, t)$, 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 $p_j \in N(p_i, t)$, then $loc(p_j, t) \in Z(p_i, r_d, t)$.

Roughly speaking, 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 the future neighbors is limited by a bounded time duration Δ_{future}. That is, it only detects a node which is in the neighborhood region at any time from the time when PRESENT is invoked up to Δ_{future}. The *perfect accuracy* property guarantees that no false detection occurs.

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 location predictions of all nodes, and then distributes all the collected location predictions to the nodes. In this way, each node can find its neighbors at current and future times based on the collected location predictions. It can also store the collected location predictions so it can be queried about its past neighbors. In our solution, we consider a *virtual mobile node* (first introduced in [6]) to be used as the moving entity. Moreover, to simplify the problem, we perform the neighbor detection only for real nodes which are in a circular region *R* of the two dimensional plane. However, using only one virtual mobile node has a main disadvantage: as the size of the region *R* grows, the virtual mobile node spends more time to travel through the network. This can cause the collected location predictions to expire before they can be used for neighbor detection. One way to overcome this problem is to increase $\Delta_{predict}$ of the mobility predictor. But, implementing mobility predictors with long $\Delta_{predict}$ is not easy. In fact, although there exist different approaches to predict the future locations of a node, usually predictions tend to become less accurate as $\Delta_{predict}$ increases.

Another way to deal with this problem is to decrease the traveling time of the virtual mobile node. In order to do so, our solution consists of using more than one virtual mobile node. In fact, our solution can work with $n = 2^k$ virtual mobile nodes where k is a non-negative integer. Thus, the region is divided into n equal subregions and each subregion is associated with one virtual mobile node. Virtual mobile nodes collect simultaneously the location predictions from the real nodes in their subregions and meet at the center of *R* to share what they have collected with each other. After the sharing, every virtual mobile node has the location predictions collected from the entire *R*. Then, the virtual mobile nodes simultaneously distribute the collected location predictions to the real nodes in their corresponding subregions. As we further show, as n grows, our solution correctly implements the neighbor detector with smaller values of $\Delta_{predict}$. Intuitively, this is because as n grows, *R* is divided into more and consequently smaller subregions and each virtual mobile node spends less time to travel through its subregion.

In the following, we first describe what a virtual mobile node is and we add n virtual mobile nodes to the system model. We then introduce an algorithm that implements the neighbor detector in the new system model and we sketch its correctness proof. In particular, we define the value of $\Delta_{predict}$ for which the algorithm is correct. Then, we show the evolution of this value as n grows.

A. Virtual Mobile Node

A virtual mobile node (also referred to as a *virtual node*) is an abstraction that is akin to a mobile node that travels in the network in a predefined trajectory. It was introduced in [6]. It is designed such that it can execute any distributed algorithm that a node can execute, however, its movement can be predefined and known in advance to all nodes in the network.

In [6] an algorithm called *Mobile Point Emulator* (*MPE*) is introduced, which implements the virtual mobile node abstraction in a system model equivalent to the system model defined in this paper. The implementation of the virtual mobile node is based on a replicated state machine technique similar to the one originally presented in [8]. The algorithm defines a *mobile point* to be a circular region of a radius r_{mp}, which moves according to the predefined path of the virtual mobile node, i.e., at time t the center of the mobile point coincides with the preplanned location of the virtual mobile node at time t. The MPE replicates the state of the virtual mobile node 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 synchronized clocks (obtained by using a service equivalent to our global positioning service). A virtual mobile node 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 virtual mobile node is *correct* if it never fails, i.e., $\forall t \in T$, at least one correct node resides in the circular region of radius r_{mp} around the preplanned location of the virtual mobile node at time t.

B. Adding Virtual Mobile Nodes to the System Model

In this section, we add a set of n virtual mobile nodes $V = \{v_1, ..., v_n\}$ to the system model where $n = 2^k$ and k is a non-negative integer. Each virtual node is assigned a unique identifier. Note that we do not provide an implementation for the virtual nodes, however, we assume that they can be implemented by the MPE algorithm sketched in Section IV-A.

Let region R be a closed disk of radius r_{map}, centered at location $l_{map\text{-}center}$ which is the origin of the two dimensional plane. Each virtual node v_i is associated with a subregion R_i of R. The subregion R_i is a sector of *R* enclosed by two radii and an arc, where the arc subtends an angle $\frac{2\pi}{n}$. All subregions have the same area and $\bigcup_{i=1}^{i=n} R_i = R$.[2]

A virtual node can communicate with other virtual nodes or the real nodes using the timely scoped broadcast where the broadcast radius equals to a constant non-negative integer r_{com} known globally. This constant is defined by the virtual node implementation (see [6]). Moreover, similar to a real node, a virtual node has access to the global positioning service.

The movement of a virtual node v_i is defined by a predetermined trajectory function $loc(v_i, t)$, which maps every t in T to a location. This function is known to all virtual and real nodes in the network. The average speed of v_i's movement is equal to a constant v_{avg}. This constant is defined by the speed of the real nodes and the speed of the subprotocols of the

[2] In general, a disk can be divided using straightedge and compass into n equal parts if $n = 2^k m$ where k is a non-negative integer and m is either equal to 1 or else m is a product of different Fermat primes.

MPE algorithm. The trajectory function of v_i is defined such that it can be used by our algorithm for the implementation of the neighbor detector. According to the trajectory function, v_i continuously scans the subregion R_i. The scans are arranged in the form of collect-distribute. More precisely, let $l_{init}(v_i)$ be a location different from $l_{map\text{-}center}$. Then, a collect scan starts at $l_{init}(v_i)$ and ends at $l_{map\text{-}center}$ and a distribute scan starts at $l_{map\text{-}center}$ and ends at $l_{init}(v_i)$. The first scan starts at time t_0 and is a collect scan. Collect and distribute scans alternate and v_i uses exactly the same path in the collect and the distribute scans. This path is called the *scan path* of v_i and its length is denoted by $L_{scan\text{-}path}(v_i)$. The amount of time that v_i spends in a collect scan is equal to the amount of time that it spends in a distribute scan. This time duration is denoted by $\Delta_{scan}(v_i)$. In order to be useful for our neighbor detection algorithm, the scan path of v_i should satisfy the three following properties:

Scan Completeness. Let s be a scan (collect or distribute) and let t_{begin} be the time when s begins, then the path traversed by v_i during s is such that $\forall location \in R_i, \exists t \in [t_{begin};\ t_{begin} + \Delta_{scan}(v_i) - 1]$ such that $distance(loc(v_i, t), location) \leq r_{com}$.

Equal Scan Path Lengths. Let v_j be a virtual node different from v_i, then $L_{scan\text{-}path}(v_i) = L_{scan\text{-}path}(v_j)$.

Proportional Scan Path Length. $L_{scan\text{-}path}(v_i)$ is an inverse function of n.

The *scan completeness* property guarantees that a scan covers the entire subregion R_i in terms of r_{com}. With regard to the *equal scan path lengths* property, it has a direct result, i.e., the value of Δ_{scan} is the same for all virtual nodes (recall that all virtual nodes have the same average speed v_{avg}). Since all virtual nodes start their scanning at t_0 and with a collect scan, this guarantees that all virtual nodes meet at the end of each collect scan at $l_{map\text{-}center}$. Finally, *proportional scan path length* guarantees that as n grows, the scan path length and consequently Δ_{scan} of each virtual node decreases.

In the longer version of this paper [5], we describe a method to find the scan path that satisfies these properties and is used by the trajectory functions of the virtual nodes. Here, due to the lack of space, we only mention the key idea behind our method. Thus, we first find the optimal path that satisfies the *scan completeness* property. It is the shortest possible path that goes through a set of locations called *covering centers*. Roughly speaking, the covering centers are such that if v_i broadcasts a message at all covering centers then the message is disseminated at all locations in R_i. Thus, covering centers are centers of disks of radius r_{com} that cover the whole surface of R_i such that the number of disks is minimum. As shown in [5], the path found in this way also satisfies the other properties of a scan path and hence, is identified as the scan path. Thus, $L_{scan\text{-}path}(v_i) = (NOC(R_i) - 1) \times \sqrt{3} r_{com}$ where $NOC(R_i)$ is the number of covering centers of R_i and is output by an algorithm that also finds the covering centers (see [5]). $L_{scan\text{-}path}(v_i)$ can be used to calculate $\Delta_{scan}(v_i)$ since $\Delta_{scan}(v_i) = \frac{L_{scan\text{-}path}(v_i)}{v_{avg}}$. Let c_1 and c_2 be two constants, as shown in [5], we also find the following upper bound on $\Delta_{scan}(v_i)$:

$$\Delta_{scan}(v_i) < \frac{1}{v_{avg}} \times (c_1 + \frac{c_2}{n}) \tag{1}$$

As we discuss in detail in Section IV-E, Eq. 1 plus the correctness conditions of the algorithm imply that as n grows, the algorithm remains correct with smaller values of $\Delta_{predict}$.

C. Neighbor Detector Algorithm

The algorithm includes two parts: a part that is executed on each real node p_i (Algorithm 1) and a part that is executed on each virtual node v_i (Algorithm 2). The algorithm relies on the movement of the virtual nodes. So, it divides time into rounds of duration Δ_{scan} where $\Delta_{scan} = \Delta_{scan}(v_i)$ with $v_i \in V$ and is globally known. Note that since the value of $\Delta_{scan}(v_i)$ is the same for all virtual nodes, there is no difference which virtual node v_i is used for calculation of Δ_{scan}.

There exist two types of rounds: collect and distribute rounds, which alternate. The first round is a collect round. Given this fact and since the execution of the algorithm starts at time t_0 (i.e., when the virtual nodes start their movement by a collect scan), the collect and distribute rounds coincide with the collect and distribute scans of virtual nodes, respectively.

Thus, the basic idea of the algorithm is as follows: in each collect round, every virtual node scans its subregion and collects the location predictions sent to it by real nodes. Then, the virtual nodes share their collected location predictions with each other when the collect round terminates (i.e., when they meet at $l_{map\text{-}center}$). At the distribute round, each virtual node distributes the collected location predictions to real nodes in its subregion. Every real node stores the collected location predictions that it receives to use them for neighbor detection. In the following, we discuss the algorithm in more detail.

Since the trajectory function of all virtual nodes are globally known, each real node p_i can calculate its distance to every virtual node at any time. So, at each collect round p_i waits until its distance to a virtual node v_i becomes less than or equal to r_{com} (note that v_i can be any virtual node in V) (line 12). Then, if p_i has not already sent a message to any virtual node in that round, it creates a message *realmsg* to send to v_i (line 14). This message encapsulates a hash map *locs* which is used to store the output of PREDICTLOCATIONS primitive of the mobility predictor service (line 15). To store each location prediction of p_i, the hash map *locs* uses one key which is the time instant for which the location is predicted e.g., $locs(t)$ returns the predicted location at time t. Once *locs* is assigned its value, *realmsg* is broadcast within the radius r_{com}, so it can be received by v_i (line 16). Each virtual node has a hash map *collectedLocs*. It is used to store the location predictions that the virtual node collects. When v_i receives *realmsg* from p_i, it stores every location prediction that exists in *locs* in its *collectedLocs* (lines 32-34). For this storage, two keys are used where one key is the name of the real node for which the prediction is made and the other key is the time instant for which the prediction is made e.g., $collectedLocs(p_i, t)$ returns the predicted location of p_i at time t. When a collect round terminates (i.e., when all virtual nodes are at $l_{map\text{-}center}$), v_i creates *intervirtmsg* to share its *collectedLocs* with other virtual nodes (lines 35-38). It broadcasts *intervirtmsg* within the radius r_{com}, so it can be received by all virtual nodes (line 39). When a virtual node receives *intervirtmsg*, it combines its own *collectedLocs* with *collectedLocs* of *intervirtmsg*, so that at the next distribute round, all virtual nodes have the same location predictions in their *collectedLocs* maps (lines 44-45). In a distribute round, v_i encapsulates its *collectedLocs* in a *virtmsg* and broadcasts it whenever it is on a covering center of its subregion R_i (lines 46-50), so it can be disseminated in the whole R_i. Each real node has a hash map called *networkLocs* that is used to store the location predictions of all real nodes in

Algorithm 1 Neighbor Detector Algorithm at Real Node p_i

```
1: initialisation:
2:    round ← collect
3:    noMsgSentInThisRound ← true
4:    networkLocs ←⊥

5: PRESENT(t)
6:    N ← ∅
7:    if networkLocs(p_i, t) ≠⊥ then
8:       for all p_j ∈ networkLocs do
9:          if p_j ≠ p_i ∧ DISTANCE(networkLocs(p_j, t), networkLocs(p_i, t)) ≤ r_d
           then
10:            N ← N ∪ p_j
11:   return N

12: upon DISTANCE(GETCURRENTLOCATION, loc(v_i, GETCURRENTTIME) ≤
    r_com such that v_i ∈ V do
13:   if round = collect ∧ noMsgSentInThisRound then
14:      realmsg ←⊥
15:      realmsg.locs ← PREDICTLOCATIONS
16:      trigger BROADCAST(realmsg, r_com)
17:      noMsgSentInThisRound ← false

18: every Δ_scan do
19:   noMsgSentInThisRound ← true
20:   if round = collect then
21:      round ← distribute
22:   else if round = distribute then
23:      round ← collect

24: upon RECEIVE(virtmsg, v_i) do
25:   for all (p_k, t) ∈ virtmsg.collectedLocs do
26:      if networkLocs(p_k, t) =⊥ then
27:         networkLocs(p_k, t) ← virtmsg.collectedLocs(p_k, t)
```

Algorithm 2 Neighbor Detector Algorithm at Virtual Mobile Node v_i

```
28: initialisation:
29:   round ← collect
30:   coveringCenters ← {l_1, ..., l_NOC(R_i)}
31:   collectedLocs ←⊥

32: upon RECEIVE(realmsg, p_i) do
33:   for all t ∈ realmsg.locs do
34:      collectedLocs(p_i, t) ← realmsg.locs(t)

35: every Δ_scan do
36:   if round = collect then
37:      intervirtmsg ←⊥
38:      intervirtmsg.collectedLocs ← collectedLocs
39:      trigger BROADCAST(intervirtmsg, r_com)
40:      round ← distribute
41:   if round = distribute then
42:      collectedLocs.CLEAR()
43:      round ← collect

44: upon RECEIVE(intervirtmsg, v_j) do
45:   collectedLocs.COMBINE(intervirtmsg.collectedLocs)

46: upon GETCURRENTLOCATION = l_i such that l_i ∈ coveringCenters do
47:   if round = distribute then
48:      virtmsg ←⊥
49:      virtmsg.collectedLocs ← collectedLocs
50:      trigger BROADCAST(virtmsg, r_com)
```

the network. Similarly to *collectedLocs*, *networkLocs* has two keys to store a location prediction: one key is the name of the real node for which the prediction is made and the other key is the time instant for which the prediction is made. For instance, $networkLocs(p_i, t)$ returns the predicted location of p_i at time t. The *networkLocs* map is extended in distribute rounds, i.e., when new location predictions are received in *collectedLocs* of a *virtmsg* (lines 24-27). Thus, whenever primitive PRESENT(t) is invoked at p_i, the map lookups on *networkLocs* as well as distance comparisons are performed to find the real nodes which are in the *neighborhood region* of p_i at time t (lines 5-9). The names of real nodes found in this way, are stored in set *N* which is returned as the result (lines 10-11).

D. Proof of Correctness

In this section, due to the lack of space, we only sketch a correctness proof for the algorithm (for complete proof see [5]). In what follows, we use the notations below:

– P_{R} is a subset of P such that $\forall p_i \in P_{\text{R}}$, p_i never leaves region R and the movement of p_i during Δ_{scan} is negligible.

– $t_{b,k}$ and $t_{e,k}$ refer to the first clock tick and the last clock tick in a round k of the algorithm, respectively. For instance, $t_{b,3}$ and $t_{e,1}$ denote, respectively, the beginning of round 3 and the end of round 1. Note that, $t_{e,k} = t_{b,k} + \Delta_{scan} - 1$.

We show that our algorithm satisfies the neighbor detector properties (stated in Section III) under the conditions below:

Conditions:

C1. All virtual mobile nodes are correct.

C2. Δ_{bcast} of the timely scoped broadcast is negligible.

C3. The execution time of lines 36-40 and line 45 of the algorithm is negligible.

C4. $\Delta_{predict} = 4 \times \Delta_{scan} + \Delta_{future} - 1$.

C5. Let p_i and p_j be the processes defined in the *time-limited completeness* property, then $p_i, p_j \in P_{\text{R}}$.

C6. If PRESENT(t) of the neighbor detector is called, then $t \geq t_{e,1}$ and $t_c \geq t_{b,3}$ where t_c is the time when PRESENT is called.

For the proof, we use Lemma 1 and Theorems 1 to 3 where Lemma 1 is used to prove Theorem 1.

Lemma 1. *Let p_i and p_j be the processes defined in the* time-limited completeness *property. Then, at every round k s.t. $k \geq 3$, networkLocs of p_i contains accurate location predictions for both p_i and p_j for the time interval $[t_{e,1};\ t_{e,k} + \Delta_{future}]$.*

Proof Sketch: For the proof of the lemma, we use the fact that at each distribute round m, p_i receives accurate location predictions for both p_i and p_j which are valid for time interval $[t_{e,m-1};\ t_{e,m+2} + \Delta_{future}]$. This can be easily shown based on Conditions C1-C5 plus the *scan completeness* property of the scan path, the *strong accuracy* property of the mobility predictor and the *timely delivery* property of the underlying broadcast. So, we prove the lemma by induction. *Base case* corresponds to round $k = 3$. According to the algorithm, round 2 is a distribute round. We know that in round 2, p_i has received accurate location predictions for both p_i and p_j for interval $T_1 = [t_{e,1};\ t_{e,4} + \Delta_{future}]$ and has stored them in its *networkLocs*. Let $T_2 = [t_{e,1};\ t_{e,3} + \Delta_{future}]$, then $T_2 \subset T_1$, so, the lemma holds in this case. For *Inductive step*, we show that if the lemma holds for a round k s.t. $k \geq 3$, then it holds for round $k + 1$. From inductive hypothesis, we get that in round k, *networkLocs* of p_i contains accurate location predictions for both p_i and p_j for $T_3 = [t_{e,1};\ t_{e,k} + \Delta_{future}]$. Then, there exists two cases: (a) *Round k+1 is a collect round*. In this case, in round k, which is a distribute round, p_i has received accurate location predictions for both p_i and p_j for interval $T_4 = [t_{e,k-1};\ t_{e,k+2} + \Delta_{future}]$ and has stored them in its *networkLocs*. Since $[t_{e,1};\ t_{e,k+1} + \Delta_{future}] \subset (T_3 \cup T_4)$, then the lemma holds for round $k + 1$ in this case; (b) *Round k+1 is a distribute round*. In this case, in round $k - 1$, which is a distribute round, p_i has received accurate location predictions for both p_i and p_j for interval $T_5 = [t_{e,k-2};\ t_{e,k+1} + \Delta_{future}]$ and has stored them in its

networkLocs. Since $[t_{e,1};\ t_{e,k+1} + \Delta_{future}] = T_3 \cup T_5$, then the lemma holds for round $k+1$ in this case. ■

Theorem 1. *The neighbor detector algorithm satisfies the time-limited completeness property.*

Proof Sketch: From Condition C6, we have $t_c \geq t_{b,3}$, which means that PRESENT(t) is called in a round k s.t. $k \geq 3$. Also, $t \geq t_{e,1}$ implies that the neighbor detection is not guaranteed for time instants before $t_{e,1}$. Thus, considering Condition C6, we can reformulate the theorem as follows. Let p_i and p_j be the processes defined in the *time-limited completeness* property. Let PRESENT(t) be invoked at p_i in round k s.t. $k \geq 3$. If $loc(p_j, t) \in Z(p_i, r_d, t)$ and $t_{e,1} \leq t \leq t_c + \Delta_{future}$, then $p_j \in N(p_i, t)$ where t_c is the time when PRESENT(t) is called at p_i. For the proof, we proceed as follows. By Lemma 1, we know that at every round k s.t. $k \geq 3$, *networkLocs* of p_i contains accurate location predictions for both p_i and p_j for interval $T_1 = [t_{e,1};\ t_{e,k} + \Delta_{future}]$. In round k, $t_c \in [t_{b,k};\ t_{e,k}]$. Thus, let $T_2 = [t_{e,1};\ t_c + \Delta_{future}]$ we have $T_2 \subset T_1$. Therefore, in round k, *networkLocs* of p_i contains accurate location predictions for both p_i and p_j for T_2. Since the algorithm guarantees the correct neighbor detection matching (line 9), if $loc(p_j, t) \in Z(p_i, r_d, t)$, then $p_j \in N(p_i, t)$ for $\forall t \in T_2$. So, the theorem holds. ■

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

Proof Sketch: Let p_i and p_j be the processes defined in the *perfect accuracy* property, if $p_j \in N(p_i, t)$, then there exists a round where p_i has received from a virtual node a *virtmsg* with a *collectedLocs* map such that a location prediction for key pair (p_j, t) exists in *virtmsg.collectedLocs*. The *no creation* property of the underlying broadcast guarantees that no location prediction is created during collection, distribution and sharing of location predictions. The *strong accuracy* property of the mobility predictor also guarantees that the location predictions collected from real nodes are accurate. Moreover, the algorithm 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 (line 9). Hence, if $p_j \in N(p_i, t)$, then $loc(p_j, t) \in Z(p_i, r_d, t)$ and the theorem holds. ■

Theorem 3. *The neighbor detector algorithm correctly implements the time-limited neighbor detector service.*

Proof: The proof follows from Theorems 1 and 2. ■

E. Impact of Increasing the Number of Virtual Mobile Nodes on $\Delta_{predict}$ required for Correctness of the Algorithm

By Theorem 3, one of the conditions under which the algorithm implements correctly the neighbor detector is $\Delta_{predict} = 4 \times \Delta_{scan} + \Delta_{future} - 1$. Considering this fact and Eq. 1, we find an upper bound for $\Delta_{predict}$:

$$\Delta_{predict} < \frac{4}{v_{avg}} \times (c_1 + \frac{c_2}{n}) + \Delta_{future} - 1 \qquad (2)$$

According to Eq. 2, as n grows the algorithm requires smaller values of $\Delta_{predict}$ to correctly implement the neighbor detector. However, as n approaches infinity, the right hand side of the equation approaches $\frac{4}{v_{avg}} \times c_1 + \Delta_{future} - 1$ which is a constant. In practice, this means that if n is very large, increasing n does not change any more $\Delta_{predict}$ required for correctness of the algorithm.

V. Related Work

The idea of using virtual mobile nodes to facilitate the design of algorithms for MANETs was first introduced in [6]. In [7], several basic algorithms that use virtual mobile nodes to solve problems such as routing and collecting data in MANETs are briefly presented. However, contrary to our work, no explicit properties for the trajectory functions of the virtual nodes are defined to guarantee their coordination. In our previous work [3], we presented an algorithm that implements the time-limited neighbor detector in a MANET. Similarly to the present work, real nodes can predict their locations up to $\Delta_{predict}$. However, the algorithm uses only a single virtual mobile node that travels in the network, collects location predictions and distributes neighbor detection-related information. Thus, in order to stay correct, the algorithm requires greater $\Delta_{predict}$ values as the map size grows. This drawback is one of the main motivations for the present work.

VI. Conclusion

We have introduced an algorithm that implements the time-limited neighbor detector service for MANETs using $n = 2^k$ virtual mobile nodes where k is a non-negative integer. We showed that our algorithm is correct under certain conditions. We also showed that as n grows, the algorithm remains correct with smaller values of $\Delta_{predict}$. To the best of our knowledge, this is the first work that uses multiple virtual mobile nodes to implement a neighbor detector service. It is also the first paper that defines a set of explicit properties for the trajectory functions of the virtual nodes to guarantee their coordination.

In this paper, one of the conditions required for the correctness of the algorithm is the correctness of all virtual nodes. So, as future work, we intend to design an algorithm that implements the neighbor detector with the same guarantees but at the same time can tolerate the failure of virtual nodes.

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

References

[1] 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.

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

[3] B. Bostanipour and B. Garbinato, Using virtual mobile nodes for neighbor detection in proximity-based mobile applications, In *Proc. IEEE NCA'14*, pp. 9–16, 2014.

[4] B. Bostanipour and B. Garbinato, Effective and efficient neighbor detection for proximity-based mobile applications, In *Computer Networks Journal*, Elsevier, vol. 79, pp. 216–235, 2015.

[5] B. Bostanipour and B. Garbinato, Neighbor detection based on multiple virtual mobile nodes, Tech Report DOP-20150615, UNIL, 2015.

[6] 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.

[7] 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.

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