Scalable Address Allocation Protocol for Mobile Ad Hoc Networks

Yu Chen, Eric Fleury, Tahiry Razafindralambo

Scalable Address Allocation Protocol for Mobile Ad Hoc Networks

Yu Chen
Google, UK
chenyu@google.com

Eric Fleury
ENS Lyon, D-NET/INRIA, France
Eric.Fleury@inria.fr

Tahiry Razafindralambo
INRIA Lille - Nord Europe, France
tahiry.razafindralambo@inria.fr

***Abstract*—In this paper, we present for mobile ad hoc networks an efficient distributed address allocation protocol which is immune to topology changes caused by node's mobility. Contrary to the common belief that mobility makes protocol design more difficult, we show that node's mobility can, in fact, be useful to provide efficient address allocation in ad hoc networks. In our protocol, each node that has been assigned an address manages a disjoint subset of free addresses independently. By taking advantage of node mobility, we can achieve roughly even distribution of free addresses amongst nodes in the system, which enables a new joining node to be configured by its neighbors via only local communication. Theoretical analysis and extensive simulation results are presented. We show that most of the address allocation requests can be processed in a timely fashion via local communication in the requester's neighborhood with time and message complexity in the order of node's degree, regardless of the network size.**

I. Introduction

A wireless mobile ad hoc network is a collection of mobile nodes that communicate over wireless links in a multi-hop fashion without any fixed infrastructure or centralized servers. The focus of this work is on dynamic address allocation in wireless mobile ad hoc networks. Address allocation is a fundamental functionality required in mobile ad hoc networks. A majority of routing protocols for mobile ad hoc networks assume that nodes are configured a *priori* with a unique address before they communicate. However, such an assumption is not trivial, since there is no global identification which is truly unique; e.g., duplicates exist in IEEE medium access control (MAC) addresses. Hence, it is required to dynamically configure wireless nodes with a unique address upon their entry into the network.

Address allocation should be designed specifically to address the characteristics of mobile ad hoc networks. Mobility is one fundamental issue that needs to be addressed. In traditional networks, dynamic address allocation can be performed by a Dynamic Host Configuration Protocol (DHCP) [2] server. But this solution is not well suited in wireless ad hoc networks due to the unavailability of centralized servers. Thus the goal should be to design a distributed fast efficient address allocation protocol that is immune to topology changes.

In this work, we present a solution that achieves this goal in spite of its simplicity. Our basic idea is to let each node play an independent equal role in the address allocation service in the sense that each node manages independently a disjoint subset of free addresses. In most existing works on address allocation, the assignment of an address to a new joining node is either authorized by multiple nodes, which are potentially distributed in the whole network, (e.g., [6]) or by a single node (e.g., [10]). In this work, we let each node take a full responsibility in managing a disjoint set of free addresses, A fundamental difference between our work and existing works is that our approach takes the advantage of node mobility to achieve roughly even distribution of free addresses, and thus, different from most existing works, the assignment of an address requires the authorization from only *one* node and there are *multiple* nodes that can authorize address assignments, which implies a reduction in communication cost if new joining nodes can find such a node in their neighborhood.

Contrary to the common belief that mobility makes protocol design more difficult, we show that node mobility can, in fact, be useful to provide efficient address allocation in ad hoc networks. Although, node mobility creates uncertainty on the network topology, it can also be exploited to disseminate information without incurring much communication overhead. Based on this observation, we aim to achieve even distribution of free addresses as follows: when a new node acquires a free address, it gets the free addresses held by all its neighbors and redistributes them evenly among its neighbors and itself. Since mobility increases the chance that different nodes meet each other, such a mechanism can even out the distribution of the free addresses held by nodes in the network, as verified by our analyses and simulation results.

We show by theoretical analyses and simulations that, in our approach, most nodes can be allocated addresses via local communication. As acquiring an address from a neighbor only generates a small number of messages (in the order of the new joining node's degree) and can be done a timely way regardless of the network size, our work provides fast efficient address allocation and it scales well to larger networks. Furthermore, as our approach mainly relies on local communication, it is less vulnerable to topology changes caused by node's mobility. In this work, our proposed solution is targeted at address assignment, but the idea can be easily extended to general resource management.

An overview of related work is given in Section II. Then we give the basic idea of our approach in Section III and describe our implementation in Section IV. Theoretical analyses are given in Section V and simulation results are presented in Section VI. Last we conclude our paper in Section VII.

II. Related Work

A number of address auto-configuration protocols have been proposed for ad hoc networks, which aim to provide an efficient address assignment in a dynamic environment; a comprehensive survey can be found in [12].

One strategy is to employ some duplication address detection mechanism in address allocation. The basic idea is to let a new joining node pick an address by itself and checks the uniqueness of the picked address by some duplication detection mechanism, and if duplication is detected, a new address is chosen; this procedure is repeated until a unique address is found. Examples of protocols that take this strategy including [11] and [5]

Another strategy is based on a central or a distributed entity that assigns unique addresses to new nodes. The Dynamic Host Configuration Protocol (DHCP) [2] is an example of this strategy. DHCP is a four phases protocol, a new joining node floods the network with a DHCP discover packet. Every DHCP server that can configure the new joining node sends (using flooding) a DHCP offer packet containing the assigned address and other configuration information to the client. The client sends back a DHCP request packet to the first server to confirm the reception of an address. In the last phase, the DHCP server sends an acknowledgment to the client. In [8], each node maintains a list of all the addresses in use in the network. When a new node joins the system, it requests an address through one of its neighbors that have joined the system. The latter chooses an address that is free according to its address list and query throughout the network for the permission to assign the chosen address; the assignment is granted only if a positive acknowledgment is received from all known nodes. In [6], address space is stored in a distributed way on a subset of specific nodes called ADA (ADdresses Agents). Each node is at least one hop away from an ADA. Each ADA periodically sends an HELLO message. A new joining node that receives the HELLO message is assigned an address by the ADA. If the new joining node does not receives an HELLO message, it becomes an ADR (ADdresses Root) with an address pool it also randomly generates partition identifier for merging purpose.

Works that are most similar to our work is Dynamic Configuration Distribution Protocol (DCDP) [7], which uses a transactional model whereby nodes are either requesters of or responders to individual configuration requests. As in our approach, each node in DCDP owns an address pool; the essential difference is how the free addresses are distributed in the address pools of nodes in the network. In DCDP, when node responds a requesting node, it sub-leases part of its available address pool to the requester; while in our approach, instead of obtaining the address pool from a single node, a requester gets the information of the free addresses in the address pools of all its neighbors and redistributes them evenly in the address pools held by its neighbors and itself; since mobility increases the chance that different nodes meet each other, such a mechanism can even out the distribution of the free addresses among nodes in the network. As we will see, the performance is much improved in our approach. The work presented in [4] is very close to [7].

A fundamental characteristic of mobile ad hoc networks is nodes' mobility. In most works, mobility has a negative impact on protocols' performance and most of them do not consider node's mobility or assume that nodes are static until they are configured. However, it is also shown mobility can also help to provide better services, such as capacity [3], security [1]. In this work we exploit mobility to design and efficient address allocation protocol.

III. Basic idea of our approach

In our protocol, in order to reduce communication overhead and latency, we let each node take a full responsibility in managing a disjoint set of free addresses — when a new joining node's address allocation request is successfully processed, it is assigned an address and a set of free addresses that it is responsible to manage. We aim to evenly distribute the free addresses among nodes, thus a new joining node can be configured by its neighbors via local communication, provided that it is connected to at least one node in the system. Our basic idea is that, during the process of its address allocation request, the new joining node redistributes evenly among its neighbors and itself the free addresses that were previously held by its neighbors. In this paper, the term "broadcast" stands for message propagation in a node's neighborhood and the term "flooding" refers to network-wide message propagation. We refer *address space* to the set of addresses from which addresses are to be assigned to nodes in the system. The system can be initialized manually by assigning one or several nodes an address and a set of free disjoint addresses. These particular nodes are only important for initialization, since nodes play an equal role after this initialization phase.

After the system initialization, when a new joining node, called *client*, acquires an address, the following steps are taken. **1)** The client broadcasts an address allocation request. **2)** When a node in the system receives a request, it responds with its free addresses ; we refer a *server* to a node that has responded to an address allocation request. **3)** The client listens to the responds from its neighbors for a timeout period. If it receives at least one free address, it picks one of them as its own address and divides the rest into $D+1$ roughly equal size portions, where D is the number of servers from which the client has received the responses. The client keeps one portion as its free addresses and send one portion to each of its servers, which will update its free addresses as the received addresses.

If the client does not get any free address during the timeout period, it resends the request; as nodes are moving around in the network, the client might meet some nodes that hold an non-empty set of free addresses when it resends the request.

Address release upon nodes departure is simple in our protocol: when a node leaves the system, it sends its address and the set of free addresses it holds to one of its neighbors which can be chosen randomly. These addresses are merged

into the set of free addresses held by this neighbor and thus they can be reassigned to other nodes later.

Note in our approach, as free addresses managed by each node are disjoint, duplicate addresses will not occur in the scenarios with node crashes, network partitions or message loss; instead, address loss might happen. Our approach does not deal with network partitions or node crashes explictly. The idea behind is that, from the aspect of address allocation, the detection of network partition and node crashes is unnecessary as long as free addresses are available for new joining nodes. If a node does not get an address after a given number of requests, in most of solutions described in the literature, a request is flooded in the network and the new joining node is assigned an address in a multi-hop flooding way (since nodes are mobile). The flooding can also be used to recover lost addresses or to generate new address pools.

It is worth noting that even if a flooding can be time and resource consuming from the network point of view it can be used in our protocol. However, in the protocol implementation proposed in this paper, thanks to nodes' mobility, we remove the flooding approach and allow some nodes, not to be configured. Our analysis and simulations show that the percentage of unconfigured node is very small and that most of the address allocation requests can be processed in a timely fashion via local communication, regardless of the network size. Moreover, unlike most of the works proposed in the literature, our work is imune to topology change.

IV. Protocol description

In this section, we present a protocol implementation of our algorithm. We will also discuss in details the implementation of basic enhacements that can improve the performance of our protocol.

As many low-level protocols (e.g., MAC protocols), we assume that each node is equipped with a timer and each node has an identifier that is unique within its two-hop neighborhood; here we use the term "identifier" to distinguish it from the one, termed "address", assigned by our address allocation protocol. We denote the identifier of node n by $id(n)$. A node's identifier can be a number randomly generated from a sufficiently large space. It is worth noting that the this identifier has to be unique at least in the two-hop neighborhood of the node. In our protocol, such identifiers are used only in local communication. Thus even though the length of identifiers is large, the overhead is restricted. Note once nodes are assigned addresses by our protocol, it is the assigned addresses, instead of the identifiers, that are used in high level communication.

A. Local variables and message formats

The local variables maintained by each node and the formats of messages exchanged by our protocol are given in Algorithm 1, where we denote the data type of node identifiers by Identifier and the data type of node addresses by Address.

A node n's address allocation state is represented by three variables: the address $addr$ assigned to n, the set $addr\_space$ of free addresses managed by n, and the identifier $client$ of the node whose request is being processed by n; $client$ is set -1 if n is not processing any request. Temparory variables are also used.

Three types of messages are defined by our protocol. The first type is ARQ (Address ReQuest) messages. This type of messages are sent by new joining nodes that require addresses. An ARQ message contains the identifier of the sender in the field src.

Upon the reception of an ARQ message, nodes that process this request responds with ARR(Address Request Response) messages. An ARR message has three fields: the identifier of the sender src, the identifier of the destination dst, that is, the identifier of the client whose address allocation request is being processed by the sender, and the set $addr\_space$ of the free address space managed by the sender.

After a client receives ARR messages from servers that are processing its request, it broadcasts an ASU (Address Space Update) message to update the servers' state. An ASU message has three fields: the identifier of the sender src, a list $server\_list$ of identifiers of the servers whose free address spaces will be updated by the client; the updated space of the ith node in $server\_list$ is specified by $addr\_space[i]$.

In the sequel, given a node n and a name x that identifies a variable, we use the denotation $n.x$ to refer to variable x at node n, and given a message m and a name x that identifies a field, we use the denotation $m.x$ to refer to field x in message m.

B. Algorithm sketch

The protocol version of the proposed algorithm, named MAAA (Mobility-Aided Address Allocation) protocol, is given in Algorithm 1. A new joining node follows Part I of Algorithm 1 to request an address. We use a variable try, initialized in lines 1 and updated in line 2, to trace the number of requests a node has sent.

A client sends an address allocation request via an ARQ message and it sets the timer to expire in time WaitTimeOut (lines 3-4). Before the timer expires, the client collects the free address spaces managed by the servers that process its request (lines 5-11); this information is carried in the ARR messages and the received free space and the identifiers of servers are recorded in variables $recv\_addrs$ and $recv\_servers$ respectively. If the client receives any free address (line 12), it picks one free address as its address (line 13) and divides the received free addresses into roughly equal size sets (line 14). The client picks one set as the free space managed by itself and it broadcasts an ASU message to update the free address space managed by servers in $recv\_servers$.

If the client fails to receive any ARR message, it resends the request (lines 17-25); the number of requests sent by a client is bounded by parameter MaxTry (line 18). The client waits for IntervalARQ (lines 19-20) before it resends the request (line 21).

Nodes that have joined the system (is already configured) follow Part II of Algorithm 1 to process address allocation requests. Each server can process more than one request at

a time. When a node receives an ARQ message, even if it has no free addresses or if it is processing another request, sends an ARR message that carries the information of its free address space which can be 0 if it has no free addresses; a random backoff mechanism is used to avoid collisions (lines 3-4), where MaxMACDelay is an estimation on the maximum latency to send a message in one hop by the underlying MAC protocol. The accuracy of MaxMACDelay will not affect the correctness of the protocol. When a server receives an ASU message from a client (line 7), it updates its free space as indicated by this ASU message (lines 8-11).

Algorithm 1 MAAA (Mobility-aided address allocation) protocol

- **Message formats:**
  ARQ messages: Identifier src;
  ARR messages: Identifier src, dst; Address $addr\_space[\,]$;
  ASU messages: Identifier src, $server\_list[\,]$; Address $addr\_spaces[\,][\,]$;
- **Local variables:** Address $addr$, $addr\_space[\,]$; Identifier $client = -1$;
- **Parameter:**
  double WaitTimeOut, IntervalARQ, MaxMAXDelay;
  int MaxTry;

PartI — Code on a client n:

1: int $try = 0$;
2: $try++$;
3: broadcast an ARQ message m with $m.src = id(n)$;
4: set TIMER to expire in time WaitTimeOut;
5: Address $recv\_addrs[\,] = \emptyset$;
6: Identifier $recv\_servers[\,] = \emptyset$;
7: **while** (TIMER $\neq 0$) **do**
8: Upon reception of an ARR message m with $m.dst = id(n)$
9: $recv\_addrs = recv\_addrs \cup \{m.addr\_space\}$;
10: $recv\_servers = recv\_servers \cup \{m.src\}$;
11: **end while**
12: **if** ($recv\_addrs \neq \emptyset$) **then** {{/* Successfully get an address */}}
13: $addr$ = one address from $recv\_addrs$;
14: divide $recv\_addrs - \{addr\}$ into $k = |recv\_servers| + 1$ subsets a_0, . . ., a_k with roughly equal size;
15: $addr\_space = a_k$;
16: broadcast an ASU message m with $m.src = id(n)$, $m.server\_list = recv\_servers$ and $\forall i \in [0, |recv\_servers|]$, $m.addr\_space[i] = a_i$;
17: **else** {{/* Fail to get an address */}}
18: **if** try $<$ MaxTry **then**
19: set TIMER with expiration time IntervalARQ;
20: **while** (TIMER $\neq 0$) **do** no-op; **endwhile**
21: go to line 2;
22: **end if**
23: **end if**

PartII — Code on a server n:

1: **while** receive an ARQ message m **do**
2: $client = m.src$;
3: backoff random time in [0, WaitTimeOut − MaxMACDelay];
4: send ARR message m' with $m'.src = id(n)$, $m'.dst = client$, $m'.addr\_space = n.addr\_space$;
5: $n.addr\_space = 0$
6: **end while**
7: **while** receive an ASU message m with $(m.src == client)$ **do**
8: **if** $(id(n) \in m.server\_list)$ **then**
9: i = index such that $m.server\_list[i] = id(n)$;
10: $n.addr\_space = n.addr\_space + m.addr\_spaces[i]$;
11: **end if**
12: **end while**

V. Performance analyses

Theoretical analyses are presented in this section. Intuitively, with higher node density or node mobility, more nodes can be assigned addresses via local communication, as they have more chance to meet different nodes. This is verified by our analyses. Our algorithm is designed for general mobile scenarios. However, as arbitary mobility prevents most problems from being analyzed, we present our analyses for a simplified scenario modeled as follows, where for presentation simplification, we allow each node to hold a "floating-point number" of addresses (non-integer).

- Initially there are N_0 nodes, denoted by $\{n_1, \ldots, n_{N_0}\}$, each of which holds R addresses, including one address allocated to itself and $R-1$ free addresses that can be allocated to new joining nodes. We denote by $S = N_0 R$ the size of the address space.
- Only one node joins the system at a time.
- When a node n joins the system, it connects to D nodes such that each node in the system becomes n's neighbor with an equal probability.

Our analyses indicate that the performance improves as node mobility or node density increases. We show in this section that the expected number of addresses at a node, including its address and the free addresses it holds, is proportional to the ratio of the total number of available addresses to the number of nodes in the system. In particular, we prove that, when $\frac{1}{r}$ of the available addresses have been assigned to $N = \frac{S}{r}$ nodes, the expected number of addresses at a node is at least $r\frac{D}{D+1}$. This result implies that, as long as the total number of nodes is no more than $\frac{D}{D+1}S$, the expected number of addresses at a node is at least one.

Note the case $N_0 < D+1$ can be modeled by a system with $D+1$ initial nodes, each of which has $\frac{N_0 R}{D+1}$ addresses, since when $D+1-N_0$ new nodes join the system, the system will reach a state with $D+1$ nodes and each node having $\frac{N_0 R}{D+1}$ addresses. So we only need to consider the case $N_0 \geq D+1$. We divide the execution into *steps* such that given any $t > 1$, the tth new node joins the system at step t; we denote by $N_t = N_0 + t$ the total number of nodes that have joined the system at the end of step t and denote by n_{N_t} the node that joins the system at step t. In particular, we refer "the state at the end of step 0" to the initial state. For any $t \geq 0$ and any $i \in [1, N_t]$, we denote by $r_{(i)}(t)$ the amount of addresses held by node n_i at the end of step t.

Addresses are reallocated by our algorithm as follows. Here we focus on the steps in which address allocation can be done via the standard procedure; this represents the number of nodes that can be assigned addresses via only local communication. Since we are interested by the total amount of addresses held by a node, for each t, we present the update on the value of $r_{(i)}(t)$ for each node n_i, instead of the specific $r_{(i)}(t)$ addresses allocated to n_i.

- Initially, only nodes $n_1, \ldots, n_{N_0}$ exist in the system, each of which holds R addresses. That is,
$$r_{(i)}(0) = R, \forall i = 1, \ldots, n_{N_0}$$
- At step t, node n_{N_t} joins the system and the free addresses are reallocated as follows.
  - A set D_nodes $= \{i_1, \ldots, i_D\}$ of D indices are randomly chosen from $\{1, \ldots, N_{t-1}\}$, representing the D nodes that are randomly picked as the neighbors of n_{N_t}.

– Note the address reallocation is simplified by allowing each node to hold a floating-point number of addresses. We have

$$r_{(i)}(t)=\begin{cases}\frac{\sum_{k\in \mathsf{D\_nodes}} r_{(k)}(t-1)}{D+1} & \text{if } i\in \mathsf{D\_nodes}\cup\{N_t\}\\ r_{(i)}(t-1) & \text{otherwise}\end{cases}$$

Only nodes in $\{n_i | i \in \mathsf{D\_nodes} \cup \{N_t\}\}$ update their addresses. These nodes have the same number, $r_{N_t}(t)$, of addresses at the end of step t. The new join node n_{N_t} cannot obtain an address from its neighbors if and only if $r_{(N_t)}(t) < 1$.

Since our focus is on address allocation via local communication, we define *local allocation processes* as those from step 0 to step t where t satisfies $r_{(N_t)}(t) < 1$.

In the sequel, we present our analyses for (1) the number of addresses held by each node at the end of each step, and (2) the number of nodes in the system when a local address allocation process stops. We denote the step in which a local address allocation process stops by step $\mathcal{T}$, that is, $r_{N_\mathcal{T}-1}(\mathcal{T}) \geq 1$ and $r_{N_\mathcal{T}}(\mathcal{T}) < 1$. The total number of nodes that have been allocated an address via only local communication is $N_\mathcal{T} - 1$. In this section, given a vector v, we use $v_{(i)}$ to denote the ith element in v and given a matrix M, we use $M_{(i,j)}$ to denote the element at the ith row and jth column in M.

A. State vector and transition matrix

We describe the state of address allocation at the end of step $t \geq 0$ by a vector of N elements for some large number $N \geq N_0$; the ith element, $i \in [1, N]$, is the number of addresses held by node n_i at the end of step t. Note only nodes $\{n_1, \ldots, n_{N_t}\}$ have joined the system, and the number of addresses held by nodes in $\{n_{N_t+1}, \ldots, n_N\}$ are defined to be 0.

Definition 1 (State Vector $r(t)$): Given any $t \geq 0$, the state of address allocation at the end of step t is described by a state vector $r(t)$ defined as:

$$r(t) \equiv \langle r_{(1)}(t), r_{(2)}(t), \ldots, r_{(N_t)}(t), 0, \ldots, 0\rangle,$$

where $r_{(i)}(t)$, $i \in [1, N_t]$, is the number of addresses held by node n_i at the end of step t.

As D nodes are randomly chosen at each step, $r(t)$ is a random variable. Given $t \geq 1$ we denote by $\mathcal{R}(t)$ the sample space of the random variable $r(t)$. Given any $t \geq 0$ and $r \in \mathcal{R}(t)$, we denote by $P(r, t)$ the probability that the state at the end of step t is r. Formally,

$$\forall t \geq 0, \forall r \in \mathcal{R}(t), P(r, t) \equiv \text{ the probability of } (r(t) = r)$$

The initial state vector and sample space are given below:

- the state vector

$$r(0) = \langle r_{(1)}(0), \ldots, r_{(N_0)}(0), 0, \ldots, 0\rangle,$$

where $r_{(1)}(0) = \ldots = r_{(N_0)}(0) = R$, and

- the sample space of the state vector

$$\mathcal{R}(0) = \{\langle r_{(1)}(0), r_{(2)}(0), \ldots, r_{(N_0)}(0), 0, \ldots\ldots\rangle\}.$$

The state $r(t)$ is decided by the state $r(t-1)$ at the end of step $t-1$ and the action taken at step t. We model the action taken at step t by a matrix $A(t)$, called a *transition matrix*.

Definition 2 (Transition Matrix): Given $\forall t \geq 1$, we use an $N \times N$ matrix $A(t)$, called the transition matrix, to represent the action taken at step t on the state of address allocation. That is, $\forall r(t-1) \in \mathcal{R}(t-1)$, we have

$$r(t) = r(t-1)A(t),$$

which is equivalent to

$$r_{(i)}(t) = \sum_{k=1}^{N} r_{(k)}(t-1)A_{(k,i)}(t).$$

Given any $t > 1$, the transition matrix $A(t)$ is a random variable. We denote by $\mathcal{A}(t)$ its sample space. Given $A \in \mathcal{A}(t)$, we denote by $P(A, t)$ the probability that the transition matrix at step t is A. Formally

$$\forall t \geq 1, \forall A \in \mathcal{A}(t), P(A, t) \equiv \text{ the probability of } (A(t) = A).$$

The matrix $A(t)$ that models our algorithm is computed as follows. At step $t \geq 1$, given the indices $\{i_1, \ldots, i_D\} \subseteq [1, N_{t-1}]$ of the D picked nodes, the transition matrix is

$$A(t) = T(i_1, \ldots, i_D, t),$$

where $T(i_1, \ldots, i_D, t)$ is an $N \times N$ matrix such that

$$T_{(i,j)}(i_1, \ldots, i_D, t) \equiv \begin{cases}\frac{1}{D+1} & \text{if } i, j \in \{i_1, \ldots, i_D\}\\ \frac{1}{D+1} & \text{otherwise, if } i \in \{i_1, \ldots, i_D\}, j = N_t\\ 1 & \text{otherwise, if } i = j, i, j \leq N_{t-1}\\ 0 & \text{otherwise}\end{cases}$$

Given a matrix $A = T(i_1, \ldots, i_D, t)$, we denote $\{i_1, \ldots, i_D\}$ by $\mathsf{D\_nodes}(A)$; intuitively, these are the D nodes picked in the action modeled by A. We can verify that the $A(t)$ so computed is consistent with our algorithm, since letting $r = r(t-1)A(t)$, we have

$$r_{(i)} = \begin{cases}\frac{\sum_{k\in \mathsf{D\_nodes}(A(t))} r_{(k)}(t-1)}{D+1} & \text{if } i\in \mathsf{D\_nodes}(A(t))\cup\{N_t\}\\ r_{(i)}(t-1) & \text{otherwise}\end{cases}.$$

The sample space of $A(t)$, denoted by $\mathcal{A}(t)$, is the set of matrix $T(i_1, \ldots, i_D, t)$ for all the possible combinations $\{i_1, \ldots, i_D\}$ of D indices from $\{1, \ldots, N_{t-1}\}$. Formally, we have

$$\mathcal{A}(t) = \left\{ T(i_1, \ldots, i_D, t) : \begin{array}{l}\big(i_{k_1} \neq i_{k_2}, \forall k_1, k_2 \in [1, D]\big) \wedge \\ (i_k \in [1, N_{t-1}], \forall k \in [1, D])\end{array} \right\}$$

B. Expected state vector

Now we investigate the expected value of the state vector at step t. We define

$$\overline{r(t)} \equiv \sum_{r\in\mathcal{R}(t)} P(r, t)r \quad \text{and} \quad \overline{A(t)} \equiv \sum_{A\in\mathcal{A}(t)} P(A, t)A$$

Below we give a property of the expected state vector.

Lemma 1:

$$\overline{r(t+1)} \quad = \quad \overline{r(0)} \cdot \overline{A(1)} \cdot \overline{A(2)} \ldots \overline{A(t+1)}$$

Proof: Due to space limitation, the proof will not be developped here. The intuition of the proof is that by defintion we have: $\overline{r(t+1)} = \sum_{r\in\mathcal{R}(t+1)} P(r, t+1)r$. And we can show that $\overline{r(t+1)} = \sum_{k=1}^{N} \left(\overline{r_{(k)}(t)} \cdot \overline{A_{(k,i)}(t+1)}\right)$. ■

Since $\overline{r(0)} = \langle r_1(0), r_2(0), \ldots, r_{N_0}(0), 0, \ldots, 0\rangle$ is known, $\overline{r(t+1)}$ can be computed if $\overline{A(1)}$, …, $\overline{A(t+1)}$ are computed. In the next lemma, we present a computation of $\overline{A(t)}$; in particular, we compute $\overline{A_{(i,j)}(t)}$, $\forall i, j \in [1, N]$.

Lemma 2: Given $t \geq 0$, letting $x(t) = 1 - \frac{D^2}{N_t(D+1)}$, $y(t) = \frac{D}{N_t(D+1)}$ and $z(t) = \frac{1}{N_t - 1}\left(\frac{D^2 - D}{N_t(D+1)}\right) = \frac{1 - x(t) - y(t)}{N_t - 1}$, we have

$$\overline{A_{i,j}(t)} = \begin{cases} x(t) & \text{if } (i \in [1, N_t]) \wedge (j \in [1, N_t]) \wedge (i == j) \\ z(t) & \text{if } (i \in [1, N_t]) \wedge (j \in [1, N_t]) \wedge (i \neq j) \\ y(t) & \text{if } (i \in [1, N_t]) \wedge (j == N_{t+1}) \\ 0 & \text{otherwise.} \end{cases}$$

Proof: First note at step $t+1$,

- the probability for a node n_i, $i \in [1, N_t]$, to be picked is $\sum_{A \in \mathcal{A}(t), i \in \mathsf{D\_nodes}(A)} P(A, t) = \frac{D}{N_t}$, and
- the probability for two nodes n_i, n_j, $i, j \in [1, N_t]$, $i \neq j$, to be picked is $\sum_{A \in \mathcal{A}(t), i,j \in \mathsf{D\_nodes}(A)} P(A, t) = \frac{D(D-1)}{N_t(N_t-1)}$.

Here we compute $\overline{A_{(i,j)}(t)}$, $\forall i, j \in [1, N_t]$. Note $\overline{A_{(i,j)}(t)} = \sum_{\forall A \in \mathcal{A}(t)} P(A, t) \cdot A_{(i,j)}$. There are four cases.

- **Case 1:** $(\mathbf{i} \in [\mathbf{1}, \mathbf{N_t}]) \wedge (\mathbf{j} \in [\mathbf{1}, \mathbf{N_t}]) \wedge (\mathbf{i == j})$**.**
  In this case, $\forall A \in \mathcal{A}(t)$, $A_{(i,i)} = \frac{1}{D+1}$ if $i \in \mathsf{D\_node}(A)$, and $A_{(i,i)} = 1$ otherwise. So we have
  $$\overline{A_{(i,i)}(t)} = 1 - \frac{D^2}{N_t(D+1)} = x(t)$$
- **Case 2:** $(\mathbf{i} \in [\mathbf{1}, \mathbf{N_t}]) \wedge (\mathbf{j} \in [\mathbf{1}, \mathbf{N_t}]) \wedge (\mathbf{i \neq j})$.
  In this case, $\forall A \in \mathcal{A}(t)$, $A_{(i,j)} = \frac{1}{D+1}$ if $i, j \in \mathsf{D\_node}(A)$, and $A_{(i,j)} = 0$ otherwise. So we have
  $$\overline{A_{(i,j)}(t)} = \frac{D^2 - D}{N_t(N_t - 1)(D+1)} = z(t)$$
- **Case 3:** $(\mathbf{i} \in [\mathbf{1}, \mathbf{N_t}]) \wedge (\mathbf{j == N_{t+1}})$.
  In this case, $\forall A \in \mathcal{A}(t)$, $A_{(i,N_{t+1})} = \frac{1}{D+1}$ if $i \in \mathsf{D\_node}(A)$, and $A_{(i,N_{t+1})} = 0$ otherwise. So we have
  $$\overline{A_{(i,N_{t+1})}(t)} = \frac{D}{N_t(D+1)} = y(t)$$
- **Otherwise, that is,** $(\mathbf{i} \in [\mathbf{N_t + 1}, \mathbf{N}]) \,||\, (\mathbf{j} \in [\mathbf{N_t + 2}, \mathbf{N}])$.
  In this case, $\forall A \in \mathcal{A}(t)$, $A_{(i,j)} = 0$. So we have $\overline{A_{(i,j)(t)}} = 0$.

Note that the prove is shorten du to space limitation. ■

Given a state $r(t-1)$ at step $t-1$, addresses are redistrbuted at step t according to $A(t)$. Note for any step t, $\forall k \in [1, N_t]$, the sum of entries at row k is $\sum_{i=1}^{N_{t+1}} \overline{A_{(k,i)}(t)} = x(t) + (N_t - 1)z(t) + y(t) = 1$. This means no address lost. The difference of two entries $\overline{A_{(i,j)}(t)}$ and $\overline{A_{(i',j')}(t)}$, $i, i' \in [1, N_t]$, $j, j' \in [1, N_t + 1]$, is

- $x(t) - y(t) = \frac{N_t - D}{N_t} \geq 0$,
- $x(t) - z(t) = \frac{(N_t - D)(N_t D + N_t - 1)}{N_t(N_t - 1)(D+1)} \geq 0$, or
- $y(t) - z(t) = \frac{D(N_t - D)}{N_t(N_t - 1)(D+1)} \geq 0$,

each of which decreases as D increases; in particular, the minimum value, 0, is achieved when $D = N_t$. Since a smaller difference in the $A(t)$ means addresses are more evenly distributed in the system, this explains why a better performance can be achieved with a larger D.

Below we present a property of $\overline{r(t)}$, which will be used in our proof for the main theorem.

Lemma 3: $\sum_{i=1}^{N_t} \overline{r_{(i)}(t)} = \sum_{k=1}^{N_{t-1}} \overline{r_{(k)}(t-1)} = \ldots = \sum_{i=1}^{N_0} \overline{r_{(i)}(0)} = RN_0$

Proof: We can prove the lemma by showing $\forall t > 0$,

$$\begin{aligned} \sum_{i=1}^{N_t} \overline{r_{(i)}(t)} &= \sum_{i=1}^{N_t} \left(\sum_{k=1}^{N_{t-1}} \overline{r_{(k)}(t-1)} \cdot \overline{A_{(k,i)}(t-1)} \right) \\ &= \sum_{k=1}^{N_{t-1}} \overline{r_{(k)}(t-1)} \end{aligned}$$

Note here that the proof is shorten due to space limitation. ■

We have our main theorem below, which states that the expected number of addresses at a node, including its address and the free addresses it holds, is proportional to the ratio of the total number of available addresses to the number of nodes in the system.

Theorem 4: At step $t \geq 0$, we have

$$\overline{r_{(i)}(t)} \geq \overline{r_{(N_t)}(t)} = RN_0 \frac{D}{(N_0 + t - 1)(D+1)}.$$

Proof: It is easy to see $\forall t \geq 0$, $\overline{r_{(i)}(t)} \geq \overline{r_{(N_t)}(t)}$. For any $t \geq 0$, we have

$$\begin{aligned} \overline{r_{(N_t)}(t)} &= \sum_{k=1}^{N_{t-1}} \overline{r_{(k)}(t-1)} \cdot \overline{A_{(k,N_t)}(t-1)} \\ &= \sum_{k=1}^{N_{t-1}} \overline{r_{(k)}(t-1)} \cdot y(t-1) \\ &= RN_0 y(t-1) \\ &= RN_0 \frac{D}{(N_0 + t - 1)(D+1)} \end{aligned}$$

■

In particular, when $\frac{1}{r}$ of the available addresses have been assigned to $N = \frac{S}{r}$ nodes, the expected number of addresses at a node is at least $r\frac{D}{D+1}$. This result implies that, as long as the total number of nodes is no more than $\frac{D}{D+1}$ of the total number of addresses, the expected number of addresses at a node is at least one.

VI. Simulation results

We evaluate the performance of our address allocation scheme through simulations using ns-2 [9]. As our protocol distinguishes from others in that we aim to achieve address allocation via only local communication, our focus in the simulations is on the percentage of nodes that are assigned addresses by neighboring nodes; we denote this value by "% of nodes" in the sequel. We examine the impact of the *mobility*, the *protocol's parameters* and the *time needed for a node to be configured*. Due to space limitation, we do not investigate the protocols parameters such as MaxTry or IntervalARQ. In our protocol, whether a new node can get free addresses via local communication depends on the state of servers that handle its request when it sends (up to MaxTry) address allocation requests; the probability that it is allocated free addresses by neighbors is higher if it meets more servers and thus depends on node mobility and size of address space.

We consider scenarios where nodes are randomly deployed in an 1000 meters $\times$ 1000 meters square; the radio range of each node is set to be 120 meters. The total number of nodes is denoted by N and the address space is R. We first consider $R = N$. The value of N is varied from 50 to 400 to achieve different levels of node density. Initially there is exactly one node in the system and new nodes are randomly joining the system as they are moving in the network area according to the a *random waypoint mobility model* (RWP), a *random waypoint mobility model with attractors* (ATT) or a *Manhattan mobility model* (MAN). In RWP model, nodes travel from a starting point to a randomly chosen destination at randomly chosen speed from $[2, 25]$m/s. When a node reaches its destination it pauses for 2 seconds before it randomly choose a new destination. ATT model is the same as the RWP model except that we define 4 attractors and the node randomly chooses a destination within a range of an attractor. In the MAN model the network area is divided into 10×10 grid. A node follow the edges of the grid. At a given intersection and after a pause time of 2s, the node randomly chooses an edge and leave the actual intersection at a random speed. The simulation duration is 300 seconds. We set WaitTimeOut to be 0.10 second, MaxMACDelay to be 0.05 second, IntervalARQ to be 5 seconds. We also run simulations where nodes are static to show how mobility can aid address allocation.

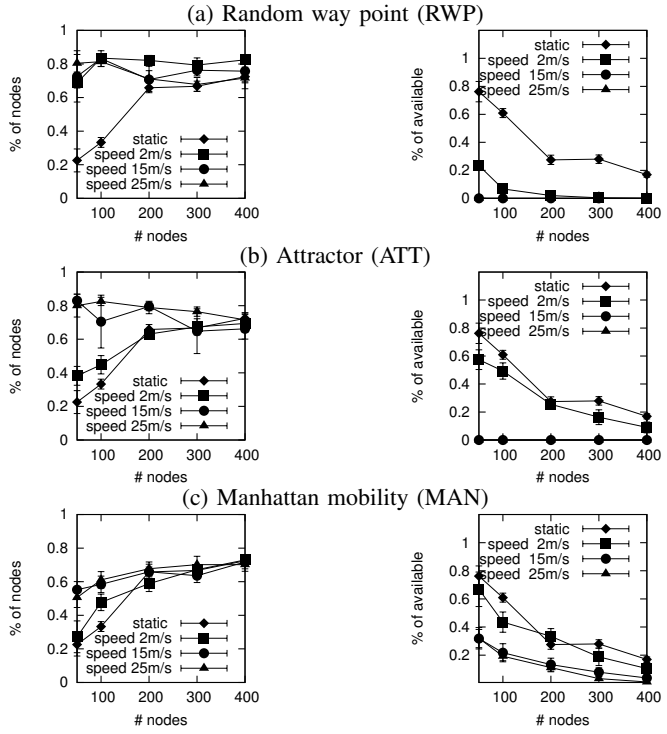

Fig. 1. Simulation results (Part I)

Our simulations consist of two parts. In part I, we first evaluate the percentage of nodes that are allocated addresses via only local communication. Our simulation results show that $\sim$70% of nodes can be allocated via only local communication (for $Speed \geq 15$m/s, $R = 200$ and RWP). Since address allocation via local communication can achieve efficiency and immunity to mobility, it is interesting to investigate whether it is possible to enable most nodes to be allocated addresses locally by giving extra bits in addresses as shown in our theorem. In part II, given a number of nodes N, we evaluate the percentage of nodes that are allocated addresses via local communication with up to 3 extra bits in addresses, that is, we consider address space $R = rN$ with $r = 1, 2, \ldots 8$. Our simulation results show that a very small number (up to 3) of extra bits in the addresses can enable most of the nodes to be allocated addresses via only local communication. The simulation results are given in Figures 1 and 2. More explanations and discussions follow.

A. Impact of mobility

Impact of Speed: We observe that for all mobility models an increasing speed, roughly increases the number of node allocated via local communication (left figures of Part I of Fig. 1). Indeed, when node's speed is high, it enables a new node to meet different sets of nodes when it sends allocation requests. It is worth noting that the % of nodes are roughly similar for different speed because MaxTry is set to infinity. Results in Subsection VI-C shows clearly how different speed can affect the protocol's performances.

Impact of density: The impact of node density differs depending on node mobility models. Surprisingly, we observe that for the random way point (RWP) model the % of nodes is stable when the density increases. The right figures of PartI of Fig. 1 show the % of available address at the end of the simulation. We can see that this value is decreasing for all mobility models but the sum of % of nodes and % of available address is not 100%. This tells us that the increasing density increase the address losses due to message collisions and thus reduce the % of nodes. As in our analysis we do not consider message losses the results of our theorem and the impact of density are different.

Impact of node's mobility model: We observe that the shape of the curves are different depending on the mobility model. However, an increasing speed increase the % of nodes for all model compared to static network. For RWP, the % of nodes of node is stable. Indeed, when the node density increase the probability for a new joining node to meet a server increases. As stated earlier, when the node density is high, messages are more likely to be lost especially ASU and thus address pool are lost and % of nodes decreases. For ATT, the attractor are useful when node's mobility is low since new joining nodes are more likely to meet server around an attractor. However, the density around an attractor is very high and thus increases the message losses and address pool loss. For the Manhattan model (MAN), the nodes are evenly distributed on the plane which reduce the collision probability compared to the other model. Since nodes are evenly distributed, the probability for a new joining node to meet a server is also low compared to the other models.

B. Impact of protocol's parameters

Impact of R: We observe in Figure 2 (left figures) that increasing the address space increases the % of nodes. These behaviour is related to the right figures of PartI of Figure 1 since increasing the value of R reduce the effect of address pool losses.

Impact of initialized nodes: We observe in Figure 2 (the right figures) that the impact of the number of initialized node is important only for low speed. Indeed for high speed, the address pool is more likely to be distributed among nodes. It is worth noting that for MAN, increasing the number of initialized nodes has a greater impact since nodes are evenly distributed on the plane with this model.

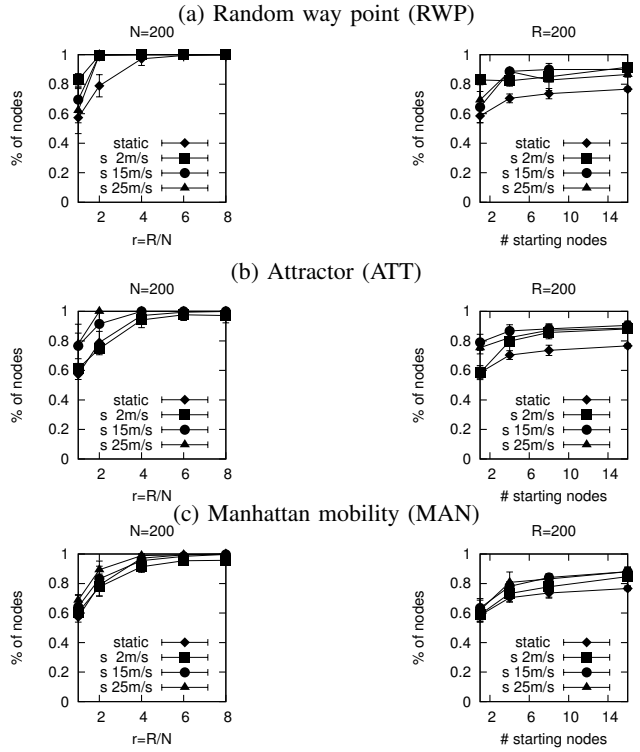

Fig. 2. Simulation results (Part II)

C. Protocol's performance evaluation

In this section we show the performance evaluation results of our protocol[1]. We assume that $R = 2 \times N$ with $N = 200$, and that we have 4 starting nodes. With these paremeters, the number of configured node is $\sim 99\%$ except for static networks. The Figure 3 plots the distribution of configuration time for RWP. We can see from this graphs that mobility increase the performance of address allocation because with the same density of node, the maximum time needed to configure all nodes when the nodes' speed is 25m/s is $\sim 40s$, this value is $\sim 70s$ for a speed of 15m/s and $\sim 138s$ for a speed of 2m/s. These distributions also tell us that more than 80% of the nodes are configured in less than 6s which means that these nodes are configured after the first ARQ request (for RWP, 25m/s).

[1]Due unfair assumptions, comparison with other address allocation protocols are omitted since most of the protocols presented in the literature consider the mobility as an issue. Moreover since our protocol does not explicitly deal with network merging, duplication address detection, we do not compare our work with protocols that focus on these properties

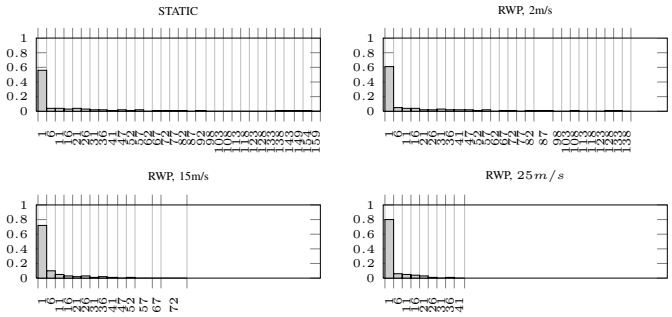

Fig. 3. Simulation results (Performance evaluation)

VII. Conclusion

We present in this work an efficient distributed address allocation protocol which is immune to topology changes. In our protocol, each node that has been assigned an address manages a disjoint subset of free addresses independently. By taking advantage of node mobility, we can achieve roughly even distribution of free addresses amongst nodes in the system, which enables a new joining node to be configured by its neighbors. Theoretical analyses and extensive simulation results are presented. We show that most of the address allocation requests can be processed in a timely fashion via local communication, regardless of the network size.

References

[1] S. Capkun, J.-P. Hubaux, and L. Buttyan. Mobility helps security in ad hoc networks. In *MobiHoc '03: Proceedings of the 4th ACM international symposium on Mobile ad hoc networking & computing*, pages 46–56, New York, NY, USA, 2003. ACM Press.

[2] R. Droms. Dynamic host configuration protocol, rfc 2131, 1997.

[3] M. Grossglauser and D. N. C. Tse. Mobility increases the capacity of ad-hoc wireless networks. In *INFOCOM*, pages 1360–1369, 2001.

[4] Z. Hu and B. Li. Zal: Zero-maintenance address allocation in mobile wireless ad hoc networks. In *ICDCS'05*, pages 103–112, Washington, DC, USA, 2005. IEEE Computer Society.

[5] J. Jeong and J. Park. Autoconfiguration technologies for ipv6 multicast service in mobile ad-hoc networks. *Networks, 2002. ICON 2002. 10th IEEE International Conference on*, pages 261–265, 2002.

[6] J.-L. Lu, F. Valois, D. Barthel, and M. Dohler. Low-energy address allocation scheme for wireless sensor networks. *PIMRC 2007*, pages 1–5, Sept. 2007.

[7] A. Misra, S. Das, A. McAuley, and S. Das. Autoconfiguration, registration, and mobility management for pervasive computing. *IEEE Wirel. Comm.*, 8(4):24–31, Aug 2001.

[8] S. Nesargi and R.Prakash. Manetconf: Configuration of hosts in a mobile ad hoc network. In *INFOCOM 2002*, June 2002.

[9] V. P. Team. *The network simulator – ns-2*. VINT Project Team, Available at http://www.isi.edu/nsnam/ns/, November 2000.

[10] M. Thoppian and R. Prakash. A distributed protocol for dynamic address assignment in mobile ad hoc networks. *IEEE Trans. Mob. Comput.*, 5(1):4–19, 2006.

[11] N. H. Vaidya. Weak duplicate address detection in mobile ad hoc networks. In *MobiHoc'02*, pages 206–216, New York, NY, USA, 2002. ACM.

[12] K. Weniger and M. Zitterbart. Address Autoconfiguration in Mobile Ad Hoc Networks: Current Approaches and Future Directions. *IEEE Network Mag.*, July 2004.