Lazy Autoconfiguration in Mobile Ad Hoc Networks and Dynamic Sets of Mobile Agents

Jan van Leeuwen

Jiří Wiedermann

Technical Report UU-CS-2006-018
April 2006

Department of Information and Computing Sciences
Utrecht University, Utrecht, The Netherlands
www.cs.uu.nl

ISSN: 0924-3275

Department of Information and Computing Sciences
Utrecht University
P.O. Box 80.089
3508 TB Utrecht
The Netherlands

Lazy Autoconfiguration in Mobile Ad Hoc Networks and Dynamic Sets of Mobile Agents ⋆

Jan van Leeuwen[1] Jiří Wiedermann[2]

[1] Department of Information and Computing Sciences, Utrecht University, Padualaan 14, 3584 CH Utrecht, the Netherlands
j.vanleeuwen@cs.uu.nl
[2] Institute of Computer Science, Academy of Sciences of the Czech Republic, Pod Vodárenskou věží 2, 182 07 Prague 8, Czech Republic
jiri.wiedermann@cs.cas.cz

Abstract. In MANETs and similar link-free networks of communicating objects there is no central authority for naming and connection management. Autoconfiguration of network nodes is therefore desirable and, building on approaches in IPv6, a number of 'zero configuration' networking protocols have been proposed for this case. Typically these protocols do not easily scale and have difficulty with network partitioning and merging. In this paper we propose a number of novel, decentralized techniques for name resolution in zero configuration protocols that are more flexible and yet lead to name extensions of smallest possible length, assuming that objects sufficiently mix within their ranges. Finally, the techniques are converted into a fully decentralized, scalable autoconfiguration protocol for use in ad hoc networks with directional antennas. The protocol is 'lazy' in the sense that name resolution is postponed until the moment that conflicts prevent the correct functioning of the communication structure.

1 Introduction

In mobile ad hoc networks (MANETs) and similar link-free networks, objects communicate by passing information between themselves without a centrally controlled network infrastructure. Lacking a central authority for naming and network management, special protocols are needed for providing and maintaining a routable system in which objects can enter and leave at arbitrary times. In particular it is desirable that these networks are 'Zeroconf' [24] and thus autoconfiguring, when the network dynamics is taken into account. In this paper we focus on the issue of naming and name resolution in Zeroconf environments.

In any networking environment, objects and/or their communication ports need to be uniquely identifiable. One typically distinguishes between the *name* of an object and its *address*, where the latter is meant for the lower-layer routing protocols. In traditional dynamic networking, name and address assignment are handled by the DHCP servers in the network. In MANETs and similar networks of objects this can no longer be assumed. Based on the approaches in IPv6, a number of protocols have been proposed to autoconfigure MANETs (cf. [5]). An important goal of autoconfiguration is to resolve name conflicts in the network and ensure the network-wide uniqueness of names, given a decentralized mechanism for assigning initial names. Many existing autoconfiguration protocols do not easily scale and having difficulty coping with network partitioning and merging. We explore some novel, alternative proposals for conflict resolution that may be more flexible and that exploit several new concepts which may be of interest in their own right. We apply it to the case of networks in which

⋆ This research was partially supported by project BRICKS (Basic Research for Creating the Knowledge Society), by Institutional Research Plan AV0Z10300504 and by grant No. 1ET100300419 within the National Research Program 'Information Society'.

objects may use so-called *smart antennas*, which allow for communication with directional control among the objects.

Although uniqueness of names is an ultimate goal of any naming system, in certain applications the requirements may be less stringent. In particular, objects may be part of different relations ('subnets') and one may only be interested in uniquely naming objects within each relation, allowing objects of the same name to exist in different relations or subnets simultaneously. All this can also happen under a dynamic scenario: new objects keep arriving, and subnets keep forming and/or merging. This is the framework we shall be adopting. Note that the issue of uniquely naming all objects is thus only a special case of the autoconfiguration problem, in which all objects happen to belong to the same relation. Of special interest to us will be the 'size' of names in *large* MANETs. The simple methods we propose in this paper keep name extensions as short as possible.

Depending on the approach taken, autoconfiguration protocols can be extensive and demanding on local storage. In MANETs this is not very desirable, as the objects cannot be expected to accommodate it. Following the design of IPv6 [20], the most common form of automatic configuration proposed for use in MANETs is *stateless* autoconfiguration. In this case it is immaterial what exact names are assigned, as long as they are unique and of use for routing purposes. (In 'stateful autoconfiguration' more information is used and thus a larger database of names and addresses can be employed.) If names are assigned, it is up to the MANET system to advertise and use them.

In this paper we propose a number of (nearly) stateless techniques for name resolution that are flexible and yet lead to unique names of minimized length, assuming that the objects sufficiently 'mix' within their ranges. Also, the latter system is converted into a fully decentralized, scalable autoconfiguration protocol for use in ad hoc networks with directional antennas. In the protocol we pursue the idea that in an ad hoc network it is sufficient if names are unique in all *neighbourhoods*. This leads to a novel 'lazy' protocol which only resolves name conflicts when they arise and when they prevent the correct functioning of the (local) communication structure.

The paper is organized as follows. In Section 2 we describe some existing approaches to name resolution in MANETs and in networks of mobile agents. In Section 3 we propose a simple protocol which starts with an unknown number of objects with initially given names, and which resolves name conflicts whenever the relevant objects meet, i.e. come within each other's range. This proposal can be tuned to obtain a protocol which leads to names of provably minimal length bounded by $H + \lfloor \log n \rfloor$ for distinguishing among n objects, where H is a bound on the size of the initial names. In the same Section we consider the robustness of the protocol under network partitioning and merging. The simple name resolution protocols are applied in the design of a 'lazy' autoconfiguration protocol for neighbourly communication in ad hoc networks with directional antennas in Section 4.

2 Preliminaries

In MANETs and similar networks, objects (agents) collaborate in spite of the absence of a supporting network infrastructure. Each node of the network can directly communicate with all nodes within a certain communication radius given by the reach of its communication hardware (possibly limited by the presence of obstacles like buildings etcetera). We do not assume that a node 'knows' the objects (nodes) in its neighbourhood or within a multihop distance away from it a priori.

The problems of naming and name management in networks are by no means new and well-studied in static contexts, e.g. in the design of distributed operating systems ([8], Ch. 7). Naming assigns to each object in a set a unique label, which must be algorithmically generated. Ensuring name uniqueness is a complex issue in dynamic sets which acquire new objects in a non-sequential manner, not one by one and without a unique *gate* which could name them. We are therefore interested in object naming in this case, when new objects keep joining a set individually or in batches and under no central control. We want methods that keep names *as short as possible*, in the interest of low cost or low energy transmissions.

While object names can easily be generated locally, one needs techniques for *duplicate name detection* (like 'duplicate address detection' of DAD [23]) and for *conflict resolution*. As duplicate detection involves probing, the techniques are often combined with *neighbour discovery* or even full reconstruction of the entire MANET topology. (Known distributed algorithms as in [2] or [7] are not sufficient for this.) Among the standard approaches to naming in ad hoc networks ([19]) are the following:

- assignment of names by an external (or global) naming authority [16],
- taking a unique 'name' stored in the hardware of the respective mobile device (e.g. the MAC number of the wireless access card) [23], or
- generating random names, using a random number generator with suitable ranges in the nodes ([18], [22]).

The first approach is often not applicable, and the second may not be in the interest of the object. The third approach is most commonly used but does not guarantee unique names and thus requires conflict resolution. When a conflict is detected, the objects involved must somehow choose other names and hope that no new conflicts are introduced. Autoconfiguration could require *symmetry breaking* between objects and thus, as leader objects are neither available nor desired in our context, we assume that objects possess a simple *random number generator* for chosing initial names randomly and for *coin-tossing* when symmetry must be broken. In e.g. [4] a good account is given of the available techniques and their shortcomings.

In some models, names are long enough to practically exclude the possibility of having doubles. In our approach we consider the case in which relatively *short* initial names are generated e.g. randomly, and conflict resolution is used to *scale up* to a network of any size. The initial object names can also be generated by *hashing* a unique but private object identifier to a small domain of short strings.

Let an object c have a current name $h = h_c$. It is appropriate to view h as the encoding of a whole set of names owned by c. Most proposals for name resolution to date follow this viewpoint. A typical example is the use of the buddy system in dynamic storage allocation for the purposes of dynamic name space allocation ([10]). If a new but unnamed object x enters the network and finds a named object c, then c assigns to x a (unique) name from its name space and subsequently splits off half of it name space and gives it to x. A similar approach using sets of integer intervals $\subseteq [0, 2^k]$ as subspaces was described in [25] and used also in the protocol presented in [21].

In this paper we interpret every name h_c as a binary string $\in \Sigma^\star$ with $\Sigma = \{0, 1\}$. A name h_c implicitly represents the full infinite set of potential names $h_c\,\Sigma^\star$, and it is 'owned' by c. In Section 3 we will show that this leads to a simple and flexible decentralized method for name resolution, without the need for much administration and with names that can remain short. We show that it gives a basically optimal solution for the naming problem, assuming that nodes suitably mix in each other's communication ranges.

We consider the name resolution problem in a general framework. Instead of only considering wireless networks of devices, we also apply it to systems of communicating agents. Agents can make their own independent decisions and, especially, they move around as they like. As in the real world, agent naming seems to be unavoidable in any community in which agents want to distinguish themselves in interactions. Obviously, agents which never interact need not tell themselves apart since they do not even know about their simultaneous existence. Agents that have ever met form a relation in the system of relations mentioned in the Introduction. That is, we see agents as mobile processes having temporary names which depend on context and which can change over time. For such a model the simple protocols we design can be used to disambiguate agent names whenever two agents with the same names encounter. This ultimately leads to the lazy protocol in Section 4.

3 A simple name resolution protocol

Consider a network of communicating objects. Every object c that wants to actively join the network, is assumed to generate an initial name $h = h_c \in \Sigma^\star$ for itself when it joins, e.g. randomly using the simple random generator it possesses. We assume that all initial names are *equally long*, so no initial name is a proper prefix of another name. This is reasonable, although the validity of our protocols will not depend on it. Let $|x|$ denote the length of a string x.

We assume that every initial h is small (e.g. $|h| = 8$). The initial choice of h_c implies that are likely to be name conflicts ('collisions') and thus name resolution is required. We want a name resolution method that does *not* depend on knowledge of N, the number of current or future nodes in the network, and that allows the number of objects to grow and shrink.

For the analysis of our protocols we will adopt a suitable model for mobile objects, which we call the *encounter model*. In this model the objects mix in each other's ranges such that eventually all objects will encounter in pairs of two and are able to check each other's name in an encounter. Objects are said to 'mix well' if they encounter uniformly at random. If objects mix well, any name resolution method will use an expected number of at least $\Omega(N)$ encounters.

3.1 Basic protocol

The basic version of our proposed name resolution method simply 'divides' the name spaces between objects of the same name, quite similar to the buddy system in [10]. The chosen representation makes this easy to implement and scalable to networks of any size.

Protocol A

1. *(initialize)* every object c that enters, generates an initial name $h = h_c$.
2. *(encounter)* if two objects of equal name h meet, then one of them is renamed to $h0$ and the other to $h1$.

(End of Protocol)

The implementation of rule 2 in Protocol A clearly requires symmetry breaking between two objects, if they find themselves having the same name h. This can be done by coin-tossing, using the simple random generator which we assume the objects to possess.

Note that every encounter of two equally named objects may lead to a further name conflict: the names $h0$ and/or $h1$ may already exist and thus, by resolving one name-conflict we may create two more. Interestingly, Protocol A may be seen as a form of *distributed hashing* if the initial names are assigned by a hash function, with distributed collision detection and conflict resolution. Assume that the set of objects in the network stabilizes at some point. We first show that Protocol A converges.

Proposition 1. *On the assumption that objects mix well, Protocol A eventually leads to unique names for all objects.*

Proof. Two objects with different names remain differently named, no matter how their names are extended in encounters with other objects. Two objects that have the same name when they meet, resolve their name-conflict in the encounter and are thus named differently forever from that moment on. The assumption implies that all pairs will eventually meet and thus that all names become unique in the long run. More precisely, this will happen with probability tending to 1 as time goes to infinity. □

Proposition 2. *Suppose the number of objects with the same initial name h has grown to some (unknown) number n. Then the unique names that will eventually result for these objects using Protocol A will all have length $\leq |h| + \lfloor \log n \rfloor$.*

Proof. As the n objects mix, the result will be that $\lfloor \frac{n}{2} \rfloor$ of them are named $h0$ and $\lfloor \frac{n}{2} \rfloor$ of them become named $h1$, eventually. Precisely $n - 2\lfloor \frac{n}{2} \rfloor \leq 1$ agent will remain with name h, which is then unique. The argument simply continues recursively. Let an object be in *level i* if it has obtained a name $h\alpha$ of length $|h| + i$. To analyse the result, write $n = a_m 2^m + \cdots + a_1 2^1 + a_0$ in binary notation, with $m = \lfloor \log n \rfloor$. The protocol eventually resolves names level after level such that in the end:

$a_i = 0 \rightarrow$ no string $h\alpha$ of length $|h| + i$ is the name of an object in the network, and
$a_i = 1 \rightarrow$ every string $h\alpha$ of length $|h| + i$ is the name of an object, of precisely one object that permanently resides in level i.

This implies that the longest names that result in Protocol A will be those corresponding to the nodes in level $m = \lfloor \log n \rfloor$. □

Proposition 3. *Suppose the number of objects with the same initial name h has grown to some (unknown) number n. The objects obtain their unique name with Protocol A after a total of $\mathcal{O}(n \log n)$ name changes.*

Proof. If level i is full in the end, the 2^i names in the level are all used and account for $2^i \cdot i$ name-changes in the process towards unique names. The total number is at most $\sum_{a_i=1}^{m} i \cdot 2^i = \mathcal{O}(m 2^m) = \mathcal{O}(n \log n)$. □

In the practical case that n is *small*, the protocol is simple and very efficient. For larger n the proof shows that in the long run a considerable number of extensions of length $\leq \lfloor \log n \rfloor$ will not be used, although they are used in intermediate stages. To be more precise, let $\overline{n} = b_m 2^m + \cdots + b_1 2^1 + b_0$ be the 1-*complement* of n, with $b_m = 0$ and $b_i = \overline{a}_i$. Then precisely the extensions in the levels i with $b_i = \overline{a}_i = 1$ (i.e. $a_i = 0$) will not be used in the end, if no new objects enter the network. In case $n = 1 \cdot 2^m$ this amounts to $1 + 2 + \cdots + 2^{m-1} = 2^m - 1$ names, thus about 50% of the available name space remains unused here.

An arbitrary object c needs to undergo up to $\log n$ name changes in order to attain its unique final name, with n as above. On the other hand, an object needs the specific encounters in order to make this happen. If the objects are mobile and mixing arbitrarily, the objects encounter randomly and thus many more than $\log n$ encounters may be needed before the desired effect is reached.

To estimate the effect properly, let us assume by a slight change of notation that we have a fixed set of N mobile objects initially and that there are $n = n_c$ objects c with initial name $h = h_c$. By the assumption that objects mix well, object c encounters other objects in the set uniformly at random, step after step. We will say that objects mix *aggressively* if the probability of meeting an equal-named counterpart is greater than 0 in every step, as long as the name resolution process has not finished. Assume also that $n = 2^m$ for some integer $m \geq 2$. (For $m = 0$ there is no name conflict and for $m = 1$ the expected number of encounters is precisely $N - 1$.)

Proposition 4. *An object c with initial name $h = h_c$ attains its unique final name in Protocol A after an expected number of at least $\frac{10}{9}(N-1)$ and at most $(2n - \log n)(N-1)$ encounters, the latter under the assumption that objects mix aggressively.*

Proof. Suppose object c has reached the ith level in the name resolution process and thus has just gotten a name $h\alpha$ with $|\alpha| = i$, for some i with $0 \leq i < m$. There will be $n_i = \frac{n}{2^i}$ nodes in level i that eventually have name $h\alpha$, and they have to encounter with another object of name $h\alpha$ (after they were named $h\alpha$) in order to resolve the name conflicts. This applies to object c in particular. Because the other objects with name $h\alpha$ only appear gradually and also disappear again after their conflict is resolved, the probability of c to encounter another object of name $h\alpha$ may be much smaller than $p_i = \frac{n_i-1}{N-1}$ in every trial.

We model the name resolution process of object c in level i by a *Poisson trial* with success probabilities $r_1, r_2, \cdots$. Here r_k denotes the probability that c encounters an object of name $h\alpha$ in the kth try. (A Poisson trial is a Bernoulli trial with varying probabilities of success, cf. [6].) We clearly have $0 \leq r_k \leq p_i$. The expected number of encounters before c can resolve its name conflict and move to the next level, is equal to the expected number of trials to obtain a success in the Poisson trial for the first time. We can bound this from below by first omitting all steps with $r_k = 0$ and then considering the resulting Poisson trial, thus effective assuming that $\frac{1}{N-1} \leq r_k \leq p_i$. The expected number of trials to reach a first success in this trial is at least equal to

$$\sum_{k=1}^{\infty} k \prod_{j=1}^{k-1} (1-r_j) r_k \geq \frac{1}{N-1} \sum_{k=1}^{\infty} k \prod_{j=1}^{k-1} (1-r_j) \geq \frac{1}{N-1} \sum_{k=1}^{\infty} k (1-p_i)^{k-1} = \frac{1}{N-1} \frac{1}{p_i^2} = \frac{N-1}{(n_i-1)^2}.$$

With this bound we can estimate the expected number of encounters in order for c to go from level 0 to level m, the level at which it will have finally resolved all its name conflicts. By linearity of expectation and using that $m \geq 2$, this number is at least equal to

$$\sum_{i=0}^{m-1} \frac{N-1}{(n_i-1)^2} = (N-1) + \frac{1}{9}(N-1) + \sum_{i=0}^{m-3} \frac{N-1}{(n_i-1)^2} \geq \frac{10}{9}(N-1).$$

In order to derive an upperbound, we return to the original Poisson trial. Let us assume that the objects mix aggressively, i.e. that $r_k > 0$ and thus $\frac{1}{N-1} \leq r_k \leq p_i$ for all $k \geq 1$. The expected number of trials to reach a first success in the trial is then equal to

$$\sum_{k=1}^{\infty} k \prod_{j=1}^{k-1}(1-r_j)r_k \leq p_i \sum_{k=1}^{\infty} k \prod_{j=1}^{k-1}(1-r_j) \leq p_i \sum_{k=1}^{\infty} k(1-\frac{1}{N-1})^{k-1} = p_i(N-1)^2 = (n_i-1)(N-1).$$

By linearity of expectation, the total expected number of encounters in order for c to go from level 0 to level m and have its name fully resolved is then bounded by

$$\sum_{i=0}^{m-1}(n_i - 1)(N-1) = (\sum_{i=0}^{m-1} \frac{1}{2^i} n - m)\,(N-1) \leq (2n - \log n)(N-1)$$

which was to be shown. □

If all objects start with initial names that have at most $n = \mathcal{O}(1)$ conflicts, the given argument shows that under reasonable assumptions *the conflicts are all resolved in an expected number of* $\Theta(N)$ *rounds of encounters.* The basic protocol thus satisfies the basic requirement of any name resolution protocol.

Under the same assumption on the mixing behaviour of the objects as above, the expected number of encounters also has a limited deviation from its mean. In particular, let X_i be the random variable denoting the number of trials needed by c in order to move from level i to level $i+1$ according to the given model.

Proposition 5. *Using protocol A and assuming that the objects mix aggressively, we have for every* $t > 0$

$$Prob(|X_i - E(X_i)| \geq t\sqrt{n_i}\,(N-1)) \leq \frac{2}{t^2}.$$

Proof. We model the random process for object c in level i as before by a Poisson trial with success probabilities $r_1, r_2, \cdots$. Assuming that the objects mix aggressively means that we let $r_k > 0$ and thus $\frac{1}{N-1} \leq r_k \leq p_i$ for all $k \geq 1$. The value of $E(X_i)$ was estimated in the proof of Proposition 4. The variance of X_i can be estimated as follows:

$$\begin{aligned} &Var(X_i) = E(X_i^2) - E(X_i)^2 \leq \textstyle\sum_{k=1}^{\infty} k^2 \prod_{j=1}^{k-1}(1-r_j)\, r_k \leq p_i \sum_{k=1}^{\infty} k^2 (1-\frac{1}{N-1})^{k-1} = \\ &= p_i(2-\tfrac{1}{N-1})(N-1)^3 = (2-\tfrac{1}{N-1})n_i(N-1)^2 \leq 2n_i(N-1)^2, \end{aligned}$$

where we use that $\sum_{k=1}^{\infty} k^2 x^{k-1} = \frac{1+x}{(1-x)^3}$. The desired estimate for $Prob(|X_i - E(X_i)| \geq t\sqrt{n_i}\,(N-1))$ now follows from Chebyshev's Inequality. □

If the set of objects in the network does not stabilize but continues to grow, Protocol A works properly and all objects eventually become uniquely named if they mix sufficiently well in the pool of objects.

3.2 Name length-optimal resolution

The basic Protocol A did not make optimal use of the available name space and could eventually leave 50% of the available names unused. This does not make a great difference in the maximum name length that is implied, but it can be avoided. We show how this can be done, in principle. The key is not to destroy all object names in the encounter rule but leave all existing names in use.

A possible approach is to split the rule '$h\,h \rightarrow h0\,h1$' into two rules '$h\,h \rightarrow h\,h0$' and '$h\,h \rightarrow h\,h1$', and to let two objects of equal name h apply one of these rules *at random* when they encounter. One can show that this leads to names of length at most $\mathcal{O}(H + \log n)$ again, in the *expected* sense. To achieve it as a worst case and thus have the guarantee of balanced name lengths, we eliminate the random choices at the expense of a marker bit. Introduce a companion name $h^\dagger$ with every name h, and change the basic protocol as follows.

Protocol B

1. *(initialize)* every object c that enters, generates an initial name $h = h_c$.
2. *(encounter)* if two objects of name h or $h^\dagger$ meet, then the name conflict is resolved according to the following rules:
   $h\ \ h\ \ \rightarrow h^\dagger\ h0$
   $h\ \ h^\dagger \rightarrow h\ \ h1$
   $h^\dagger\ h^\dagger \rightarrow h^\dagger\ h1$
   ($h^\dagger\ h$ is treated as $h\ h^\dagger$.)

(End of Protocol)

In all other encounters, objects with name $h^\dagger$ are treated as objects with name h. (The name $h^\dagger$ is implemented with a single indicator bit separate from the name field.) The implementation of rule 2 in Protocol B again requires that arising symmetries are broken by coin-tossing, using the simple random generators that all objects possess.

Names now carry a tiny bit of extra semantics: names h stand for an equal number of objects that were renamed to $h0$ and to $h1$, and names $h\dagger$ stand for an equal number of objects that were renamed to $h0$ and to $h1$ plus one more object that was renamed to $h0$. Assume that the number of objects in the network stabilizes at some point. We first show that Protocol B again converges.

Proposition 6. *On the assumption that objects mix well, Protocol B eventually leads to unique names for all objects.*

Proof. Objects whose initial names differ, will have different names forever. We thus consider an arbitrary initial name h and show that the name conflict between all objects with initial name h is resolved properly in the limit. As the protocol leads to objects with names $h0$ and $h1$ (the latter only if there are more than 2 objects with name h), the result then follows by induction. Define the following counters, where we assume that the number of objects that eventually enter the network with name h is n:

$a =$ the number of objects with name h $\quad b =$ the number of objects with name $h^\dagger$,
$c =$ the number of objects with name $h0$ $\quad d =$ the number of objects with name $h1$.

Protocol B can be seen to maintain the following invariants:

(I1) $a + b + c + d = n$,
(I2) $c = b + d$,
(I3) $a + b$ decreases by 1 in each encounter of two objects named h or $h\dagger$.

The result now follows from invariant I3. Eventually, i.e. with probability tending to 1 as time goes to infinity, all objects with name h meet and we obtain that $a + b = 1$ and thus either $a = 1$ and $b = 0$ or vice versa. Also $c + d = n - 1$ and, whatever c and d are, they are $< n$ and thus the name resolution converges in the limit by induction. □

Proposition 7. *Suppose the number of objects with the same initial name h has grown to some (unknown) number n. Then the unique names that will eventually result for these objects using Protocol B will all have length $\leq |h| + \lfloor \log n \rfloor$.*

Proof. Observe the following additional invariant for the protocol. Let s count the number of steps in the process of resolving a name conflict involving h or $h^\dagger$:

(I4) $d = a + c - n + s$.

Note that the resolution process leads to a single name h in $s = n - 1$ name changes: the result will be a name h without a † if n is odd and a name $h^\dagger$ with a † if n is even. From the invariant it follows that at this point the following occurs:

- if n is *odd*, then $a = 1, b = 0, s = n - 1$ and thus $d = c = \lfloor \frac{n}{2} \rfloor$.
- if n is *even*, then $a = 0, b = 1, s = n - 1$ and thus $d = c - 1$ and $c = \lfloor \frac{n}{2} \rfloor$.

Proceeding recursively we obtain a name tree T_n which has an object named h or $h^\dagger$ in its root, $T_{\lceil \frac{n-1}{2} \rceil}$ as its left subtree, and $T_{\lfloor \frac{n-1}{2} \rfloor}$ as its right subtree, with the subtrees filled recursively. Write $n = (2^m - 1) + l$ for some $0 \leq l < 2^m - 1$. Then T_n is a binary tree with its first m levels completely filled and the $(m+1)$-st level filled with l nodes. The bound on the name-lengths implied by using Protocol B follows, as $\lfloor \log n \rfloor$ is the lowest level of T_n. □

Proposition 8. *Name resolution using Protocol B leads to names with extensions of minimum possible length.*

Proof. This follows because T_n is a minimum-depth binary tree with n nodes. Names of length $|h| + \lfloor \log n \rfloor$ (in the bottom level) are only used insofar as names in this level are needed. □

3.3 Joining, leaving, and migrating objects

The main advantage of Protocol B is its usage of name extensions of minimum possible length, its scalability and its flexibility w.r.t. migration or failures of objects. Objects can *join* at any time and name conflicts are automatically resolved, in a reasonable number of expected steps for every object. Objects that *leave* or *fail* do not obstruct the protocol either, and the protocol just continues to operate correctly.

Clearly objects that leave take their name with them and thus make 'holes' in the used name space. These holes are automatically filled up again if the corresponding names are generated again in a name resolution step. If they are not filled up, the effect of leading to unnecessary name lengths is limited because of the smoothing effect of the logarithmic factor. Thus, an explicit 'name reclamation' protocol is not really necessary.

The more general case of migrating objects is not much harder. Assume that a new object arrives from a foreign subnetwork or community into an already existing subnetwork or community of objects. Suppose that in both communities, objects used protocols A or B for name resolution. Obviously, the appearance of a new object with a given name is *not harmful* to these protocols, which work regardless of the different initializations of names. In fact there is no mechanism by which encountering agents can sense that one of them is foreign, and the protocols just work. The minimality of names in a given network can be disturbed, though.

3.4 Network partitioning and merging

Consider now the situation in which two communities, of sizes n_1 and n_2 respectively, merge i.e. are brought together and begin to mix. Consider the case that both communities have been using Protocol B. The protocol can simply continue to resolve the name conflicts in the merged set without change. The protocol works simply as if the objects from the different subcommunities have not met until now. Clearly, it does not even matter how many communities merge. Assume that in both communities names of maximal length have been used, i.e. names of length $H + \lfloor \log n_1 \rfloor$ and $H + \lfloor \log n_2 \rfloor$, respectively. Since $\max\{\lfloor \log n_1 \rfloor, \lfloor \log n_2 \rfloor\} \leq \lfloor \log(n_1 + n_2) \rfloor$, the names in neither community are 'too long' at the moment of the merge and Protocol B will resolve names with a minimal extension again.

Proposition 9. *If Protocol B has been used from the very beginning in the given subnetworks or communities, then it continues to yield name extensions of minimum possible length after the subnetworks or communities merge.*

In a merge, renaming is necessary only if objects of the same name from the originally separate communities encounter. Potentially, our name resolution scheme gives shorter names than other protocols, but the disadvantage is that there might exist duplicate names and that names are never final in a dynamic context. On the other hand, when merging sets of agents, the non-uniqueness and thus non-definiteness of names is always to be expected.

If the objects in a network or community get partitioned, the objects can keep their names and, insofar their names were not yet unique, the protocol can continue to operate without intervention. Without any further measures, there is no way for objects to discover their non-presence within a given network. However, it is clear that, again, it may lead to big holes in the set of names in a subnetwork and the naming then ceases to be name length-optimal. On the positive side, when objects return they can use their original names as if nothing happened, i.e. as if in the meantime they have not encountered any object.

4 Lazy autoconfiguration protocol for dynamic networks of agents

We now show how to incorporate the name resolution method into an automatic configuration protocol, given an 'initial name service' that is used at the creation of every object and given the lazy properties of the detection protocol. We abandon the 'encounter model' which we only needed for proving some basic properties. We take the general viewpoint of dynamic networks of (autonomous, mobile) *agents*. An agent can be any artificial, software- or hardware-based entity that can act autonomously in a dynamic environment.

Recent developments indicate that the full potential of ad hoc networks can be achieved by making use of mobile devices equipped with so-called *smart antennas*. Smart antennas enable a directional sending and receiving of messages. Their use has a number of advantages over so-called *omnidirectional* antennas (cf. [14, 15]), e.g. they avoid the need of 'flooding' an entire neighbourhood when a message is to be sent to an specific agent in a known direction within the neighbourhood. In this section we assume a model that reflects this, more powerful mode of communication.

For message routing, the agents in a network need to have unique names. However for local message routing in ad hoc networks, 'short' names that are locally unique are sufficient. This is where lazy duplicate detection comes in as a useful concept. Nevertheless, as uniqueness of names is hard to authenticate even locally, naming methods that aim at short public names must use 'private' information expressed in (unique) $agent\_identifiers$. These identifiers may be derived from some owner-specific number (cf. Section 2 and [23]). We shall adopt this idea as well. The $agent\_identifiers$ are used only for authentication purposes but avoided in message exchanges between named agents whenever possible.

4.1 A self-organizing naming and neighbourly communication protocol

In our model we assume that the agents move in 3D Euclidean space and have no special knowledge about their location. We also assume that agents do not move 'too fast' compared to the speed of the protocol operations. We capture this by defining the so-called *timing margin* ΔT: it is the fraction of time during which an agent cannot leave the reach of the sending agent at a given time and several actions of protocol C below can be performed. That is, we assume that the relative positions of agents remain approximately the same during several rounds of the protocol below. This assumption seems to be fulfilled in most imaginable applications.

The basis for all communication under such conditions is for each agent to learn its neighbours. Object c is a *neighbour* of agent d at a given time if d is reachable from c by a direct communication at that time. In order to identify neighbours uniquely in case messages must be sent to them, each agent c maintains a *table* of its neighbours. The entries in this table contain *directional addresses* of the form $\langle$*agent_name*,*agent_identifier*,*direction* $\rangle$, where *direction* denotes the direction in which an agent with the respective address was seen at the occasion of its last communication with c. The agent names and directions are used for detection of name conflicts and their resolution. The agent identifiers are revealed only among those agents who need to (and agree to) communicate. An agent's identifier is not used for message routing.

Known solutions to establish a communication structure exploit e.g. leader election (cf. [11], [12], and [22]) and require storing address books with names and identities of all agents within the network in every node. We aim at an approach without leaders and that is respects the privacy of agents, in the sense that each agent "knows" only the identifiers of its immediate partners and minimizes the transfers of their identifiers.

We adapt Protocol A so that it can be used as a part of a neighbour detection and naming protocol, in combination with a neighbourly message communication scheme. We need to make the following assumptions on the communication abilities of the agents:

- each agent maintains the table of its (known) neighbours, together with their (known) directional addresses,
- each agent has the ability to transmit to and receive messages from other agents directionally,
- each agent has the ability of receive omnidirectionally and to determine the direction from which a message has arrived,
- the communication protocol allows that only one agent can be involved in a communication with another agent; if two or more agents try to engage in a communication with an other agent, all participating agents will recognize this fact and neither addressee will obtain the message.

The first three conditions are the same as in the model of [16] used for neighbour discovery. The fourth one is necessary in order to prevent interference of messages exchanged between two agents. Such communication protocols do exist in this context (cf. [14, 15]).

We describe Protocol C, which is based on Protocol A, allowing initially unnamed agents to form dynamic networks in which neighbourly messages can be delivered to their addresses. We specify the protocol in a self-descriptive, *action-driven* format for any agent c, in three parts. The first part is simple and concerns the first action of any agent coming to life in the system for the first time.

Subprotocol C.init

% If an agent c becomes active and enters the set of agents, it carries out the folllo-
% wing action to acquire an initial, external name.

1. *Initialization*
   Agent c generates an initial name $h = h_c$ ('prenaming').

(End of Subprotocol)

The next part of Protocol C deals with neighbor discovery and, also, table maintenance. An important invariant that must be maintained is the connection between an agent's name and its $agent\_identifier$, as known and registered *locally* by other agents in their tables. Note that it cannot be a global invariant due to the allowed occurrence of doubles in the network. Locally we control the invariant, but we allow for agents to be mobile without necessarily checking everything every time. This implies that we allow the invariant to go 'out of bounds' temporarily, in cases that are not likely to occur (cf. the déjà vu-situations as explained in Section 4.2). These cases eventually correct themselves again.

Subprotocol C.nd

% During 'neighbour discovery' each agent c intermittently switches between the
% following two actions.

1. *Neighbour discovery: sending advertisements to neighbours*
   In *directional* mode c sends advertisements in a sequence of probes issued in all directions covering the part of space under consideration. Within a probe, c advertises its presence by sending its *name*.

2. *Neighbour discovery: Receiving advertisements from neighbours*
   In *omnidirectional* mode, c waits for advertisements. If c receives an advertisement from agent d, agent c determines the direction of the message it received. Then it compares its own name with the name of the sending agent. Now the following cases may arise:
   (a) *the names are different and d is known to c*:
   In this case c checks the direction from which the advertisement came. If it coincides with the direction c knows for d from its table plus or minus a *mobility margin*, then c corrects the direction of d in its table if necessary, and responds *directionally* with its own advertisement. If the direction of d's advertisement differs by more than the allowed margin from the registered value, then c treats d as a new agent and acts as in the next case.

(b) *the names are different but d is unknown to c*:
In this case the agents introduce themselves by exchanging their values of $agent\_identifier$. If the identifier of agent d already occurs with a name d' in c's table (with d' necessarily a prefix of d in this case), c updates the entry for d' by substituting its new name and direction. If the identifier of agent d did not occurs in c's table, then c now creates an entry for d in its table.

(c) *the names collide*:
Now c and d apply Protocol A to resolve their name conflict. Next they introduce themselves by exchanging their values of $agent\_identifier$, for the purpose of future communication. Then both agents update the respective entry in their tables of reachable agents and both agents inform their neighbours about their current (new, changed) name by sending directional advertisement to them (as in Action 1).

% Note that step (c) guarantees that the agents announce their new names and
% thus force the agents in their neighborhood to update their tables according-
% ly, in this way restoring there the right correspondences between name and
% $agent\_identifier$.

(End of Subprotocol)

Subprotocol C.nd can be triggered in two ways: whenever the agent wants to carry it out, or whenever there is a name conflict detected that must be resolved. (The name conflict may be detected in several directions.) In the latter case especially, surrounding agents rely on the name resolution as part of the autoconfiguration and we must assume that a run through Subprotocol C.nd takes place quickly, within timing margin ΔT. By definition this also implies that the local situation does not change too drastically, i.e. that no mobility margins are exceeded within this time period (which would otherwise induce a possibly lengthy chain of new introductions by the last clause of action 2.(a) of the subprotocol). Thus we assume that within ΔT time two agents can resolve a name conflict and advertise their new names by action 2.(c), and that within another ΔT time the surrounding neighbors can process this information according to action 2.(a), for a total of $2\Delta T$ time.

The final part of protocol C uses the autoconfiguring infrastructure of the agent network and deals with the exchange of messages in local neighborhoods, the basic communication mode. The approach relies on Subprotocol C.nd to maintain the directional addresses around every agent. If an agent c wants to send a message to local neighbor e, it first sends a 'message announcement' to trigger any needed autoconfiguration in the neighborhood, i.e. resolve any name conflicts that may still exist, before sending the intended message to the intended neighboring agent e. In this context Subprotocol C.nd is assumed to be pro-active and quick, and Subprotocol C.com is running as a slower client on top of it.

Subprotocol C.com

% Each agent c wanting to send a message to a neighbour e engages in the various
% actions in the following series.

1. *Neighbour communication: Sending a message announcement*
   Agent c sends a message announcement in *directional* mode to the selected neighbour named e in its table. A message announcement has the form $\langle(c, e)\rangle$ and is delivered to all agents with name e within the chosen direction, i.e. plus or minus the mobility margin.

   % If c receives a notification from the communication systems that a name conflict
   % has arisen (see below), then it drops the announcement and tries again later,
   % otherwise it waits for e to respond. Agent e may or may not have moved away.

   If there is no agent e in the chosen direction, no response will come and the attempt 'times out', and c moves to action 3 below.

   % As argued above, a suitable time-out value will be $2\Delta T$.

2. *Neighbour communication: Receiving a message announcement*
   Upon receiving a message announcement, every neighbouring agent with name e wants to respond in a directional mode back to c. Now one of the following actions applies:

   % By assumption, if two or more agents now want to respond to c, the communica-
   % ation protocol reveals to them (and to c) whether there is a name conflict or
   % not.

   (a) *Receiving a message announcement: no name conflict*
   If there is no name conflict detected and thus e is a single agent with that name in c's neighbourhood, agent e checks two further possibilities:
   – *c is known:* c occurs in e's neighbourhood table in the direction from which the message announcement has arrived (plus or minus a mobility margin), meaning that both agents have already introduced themselves in the Subprotocol C.nd actions. Therefore e can notify c of this and c can send the message safely to e without requiring any further authentication from e.
   – *c is unknown:* c is not found in e's neighbourhood table in the direction from which the message announcement came, meaning that the agents have never met before or have changed their relative positions in the meantime. Therefore, they first identify themselves, revealing their identities and updating their table entries, and c can send his message to e (cf. action 3 below).

   % If there is a name conflict, the agents e are immediately diverted to the
   % following action however:

   (b) *Receiving a message announcement: name conflict*
   If it is detected that there are two or more agents with name e ('doubles') in c's neighbourhood, then c postpones its message sending to 'e', it saves the $agent\_identifier$ of e to mark it, and *every* agent e in its neighbourhood engages to resolve the name conflict as in Subprotocol C.nd of Protocol C.

% Note that in the framework of this protocol the former agents e, after having
% gotten their new names, advertise themselves anew to c who can update its
% neighbourhood table. Eventually, c can send its postponed message to the
% correct agent e again even though e may now be known under a new name.

3. *Neighbour communication: Sending a message*
   If c receives a suitable notification from e, it sends the intended message to e. If c receives no notification from e within $2\Delta T$ time, it can either give up (assuming e.g. that e is not there) or check its table for the current name e' of the agent with the identifier of e it has saved and resend the message announcement to e'.

   % Note that $2\Delta T$ time should be sufficient for the needed name resolution between
   % some of the agents e. It may be that some agents e remain but the action will
   % chose the right agent for sending the message to in the retry. Note that name e'
   % will be an extension of name e. If e is not responding within the set time after
   % another retry, c can choose to delete it.

(End of Subprotocol)

This completes the descriptions. Further Subprotocols can be built on top of C.com, for example for multi-hop routing in a MANET.

4.2 Adequacy of Protocol C

Protocol C consists of the 'stack' of Subprotocols C.init, C.nd, and C.com that together create an autoconfiguring communication infrastructure for the agent network.

In Protocol C we took a generic approach and let agents send a separate message announcement to the intended neighbour first, prior to actually sending a messages to this neighbour. This avoids with high probability that a message could be received by a 'double' to whom the message is not addressed. In case this is considered harmless, the message could be included in the announcement and the further exchange can be aimed at a notification in action 3 of Subprotocol C.com that is only to be interpreted as an 'acknowledgement of receipt'.

Note that, after the agents have initially advertised themselves to their neighbours, their value of $agent\_identifier$ is examined only in statistically unlikely cases, i.e. when in a given direction (sector) there appears a newly arriving agent with an 'existing' name (a 'double') or when there is a yet undetected name conflict in that direction. This is warranted because we have assumed that the subprotocols all run quick enough compared to the local movements of the agents.

Protocol C makes essential use of the smart antennas. For example, when doubles are detected after sending a message announcement, these doubles must be within reach of each other in order to be able to communicate. In particular, when r is the reach of broadcasting, they must not be farther than r from each other. Thus, they both must lie in a cone with at most a 60 degree spherical angle (in 3D space). Hence directional antennas are needed, otherwise the protocol would not work correctly. (Note that in the case of an omnidirectional antenna the same message can be heard by two doubles whose distance is more than r.)

Protocol C can err in case agent c sends a message to e and e is a newly arriving agent which has already met an agent c (in some other part of the network) *different* from the one which is sending the message. Especially, if the relative positions of both c and the 'old' e, and that of the 'other' c and the 'other' e, coincide (or: differ by less than the mobility margin), this may cause confusion. This situation is called a *déjà vu* situation. However, this situation seems to be highly improbable especially thanks to the fact that 'prenaming' avoids most of the name collisions from the very beginning.

Proposition 10 (Correctness and finiteness of Protocol C). *Let c be an agent in a network controlled by Protocol C, and let at any time* $t = t_0$ *there be an agent e with directional address* $\langle e, e\_identifier, d\rangle$ *recorded in c's neighbour table. Then, provided that at any time* $t > t_0$ *there is an agent with identity* $e\_identifier$ *in the neighbourhood of c that is responding and with the exception of* déjà vu *situations, Protocol C will deliver a message sent by c to that agent in finite time, irrespective of whether there were doubles of e or not in its neighbourhood.*

Protocol C is a *lazy* protocol since it does name conflict detection at any moment the agents want to and can postpone resolution to the last possible moment, i.e. to the moment when duplicate names cause message delivery to two or more recipients c.q. to non-unique addressees. Even then only one of the possible conflicts needs to be resolved at a time (cf. [18] for a similar idea using randomly assigned names). This contributes to the minimization of traffic in the network: only the conflicting agents and the partners of the agent which had to change their name are involved. This is to be compared with so-called DAD-based protocols (Duplicate Address Detection, cf. [13], [5]) which use flooding in order to eliminate duplicates in the whole network and hence lead to a large number of DAD messages, especially in the case of network merging.

4.3 Joining, leaving and migrating agents

Joining a network is easy – a name for a joining agent is provided immediately and the agent is fully operational in terms of having the possibility to communicate with agents chosen in the neighbour discovery process even, if name conflicts are to be expected. Thus, with respect to the resolution of duplicate names, Protocol C is related to the proposal in [23] and in fact it is similar to that of [13] where a joining agent has to check immediately whether its randomly chosen address is not used by somebody else. "No reply" is interpreted as "no", but this can be due to the failure or absence of the queried agent. Therefore, without a further measure this approach does not lead to a reliable duplicate names resolution. In our case, however, the duplicate checking is performed only within the neighbour discovery range; on the other hand, it is repeated all over again, which also helps in the case of a node failure.

Protocol C efficiently handles the migration of agents also. Only in cases when several agents at the same time decide to join a network, a proliferation of conflict resolution messages can occur. This seems to be the case with any decentralized protocol without leaders (cf. [19]). However, thanks to the lazy technique, in our case there is no network flooding when assigning names or resolving duplicates, in contrast to techniques that preventively detect all possible naming conflicts without waiting whether they really occur (cf. [11]). The absence of leaders in our protocol has a further advantage in making the protocol more robust against the failure or temporary absence of nodes. Note that failures are accommodated in very much the same way as nodes 'leaving'. Neighbouring agents will eventually detect that the failed or migrated agent is no longer responding and cease to communicate with it.

5 Conclusion

Name resolution protocols for ad hoc networks always seem prepared for the worst case scenario, in which every object will communicate with every other object. In contrast to it, our basic protocols offer a smooth transition from the initial state when hardly any object communicates, to the intermediate but very probable state when groups of communicating objects have more or less stabilized and changes occur only sporadically. It has thus lead us to the useful concept of *lazy autoconfiguration*.

We presented some novel protocols to show that lazy autoconfiguration in MANETs and dynamic networks of agents is indeed feasible. The protocols derive from a basic 'lazy' version which resolves names conflicts only when they arise in an encounter. An optimization of this protocol lead to a version which even uses name extensions of minimum possible length. The basic lazy name resolution protocol was then converted into a lazy autoconfiguration protocol which can be used e.g. in ad hoc networks with directional antennas and for the purposes of neighbourly communication.

Unlike several other proposed protocols, our protocol scheme easily scales and is resilient to the joining, leaving (including failure), and migration of agents. Last but not least, the protocol is completely decentralized, making use of no leader nodes and no global information.

References

1. B. Awerbuch, Chr. Scheideler, Group Spreading: A protocol for provably secure distributed name service, in: J. Diaz, J. Karhumäki, A. Lepistö, D. Sanella (Eds.), *Automata, Languages, and Programming*, Proceedings 31st Int. Colloquium (ICALP 2004), *Lecture Notes in Computer Science* Vol. 3142, Springer-Verlag, Berlin, 2004, pp. 183-195.
2. J. Beauquier, P. Gastin, V. Villain, A linear fault-tolerant naming algorithm, in: J. van Leeuwen, N. Santoro (Eds.), *Distributed Algorithm*, Proceedings 4th International Workshop, *Lecture Notes in Computer Science* Vol. 486, Springer-Verlag, Berlin, pp. 57-70, 1990.
3. F. Bellifemine, I. Constantinescu, S. Willmott, *A Uniform Resource Name (URN) namespace for Foundation for Intelligent Physical Agents (FIPA)*, RFC 3616, Network Working Group.
4. C. Bernardos, M. Calderon, Survey of IP address autoconfiguration mechanisms for MANETs, internet draft, *http://bgp.potaroo.net/ietf/idref/draft-bernardos-manet-autoconf-survey/#ref-1.*
5. Z. Fan, IPv6 stateless address autoconfiguration in ad hoc networks, in: M. Conti *et al.* (Eds.), *Personal Wireless Communications* (PWC 2003), IFIP-TC6 8th International Conference, Lecture Notes in Computer Science, Vol. 2775, Springer-Verlag, Berlin, 2003, pp. 665-678.
6. W. Feller, *An introduction to probability theory and its applications*, Vol. 1, 3rd edition, J. Wiley & Sons, New York, 1968.
7. P. Fraigniaud, A. Pelc, D. Peleg, S. Pérennes, Assigning labels in unknown anonymous networks, *Distrib. Computing* 14 (2001) 163-183.
8. A. Goscinski, *Distributed operating systems: The logical design*, Addison-Wesley Publ. Comp., Sydney, 1991.
9. E. Guttman, Autoconfiguration for IP networking: enabling local communication, *IEEE Internet Computing* May-June (2001) 81-86.
10. M. Mohsin, R. Prakash, IP address assignment in a mobile ad hoc network, in: *MILCOM 2002*, Proc 2nd IEEE Military Communications Conference, 2002, pp. 856-861.
11. S. Nesargi, R. Prakash, MANETconf: Configuration of hosts in a mobile ad hoc network, in: *INFOCOM'2002*, Proceedings 21st Joint IEEE Conference, IEEE, June 2002.
12. P. Patchipulusu, *Dynamic Address Allocation Protocols for Mobile Ad Hoc Networks*, MSc. thesis, Computer Science, Texas, A&M University, August 1997
13. C.E. Perkins, J.T. Malinen, R. Wakikawa, E.M. Belding-Royer, Y. Sun, IP address autoconfiguration for ad hoc networks, *IETF Internet draft*, November 2001.
14. R. Ramanathan, On the performance of ad hoc networks using beamforming antennas, in: *Second ACM International Symposium on Mobile Ad Hoc Networking and Computing* (MobiHoc 2001), Proceedings, ACM Press, October 2001, pp. 95-105.

15. R. Ramanathan, J. Redi, C. Santivanez, D. Wiggins, S. Polit, Ad hoc networking with directional antennas: A complete system solution, *Journal of Selected Areas in Communications*, Vol. 23, Issue 3, March 2005, pp. 496-506.
16. M.E. Steenstrup, Neighbor discovery among mobile nodes equipped with smart antennas, in: *3rd Scandinavian Workshop on Wireless Ad-hoc Networks* (ADHOC'2003), Proceedings, 2003, *http://www.wireless.kth.se/adhoc03*.
17. M. Straßer, J. Baumann, F. Hohl, M. Schwem, K. Rothermel, *ATOMAS: A Transaction-oriented Open Multi Agent-System*, Fakultätsbericht Nr. 11, Fakultät Informatik, Universität Stuttgart, 1998.
18. Y. Sun, E.M. Belding-Royer, Dynamic address configuration in mobile ad hoc networks, UCSB Tech. Rep. 2003-11, Dept of Computer Science, University of California at Santa Barbara, June 2003.
19. Y. Sun, E.M. Belding-Royer, A study of dynamic addressing techniques in mobile ad hoc networks, *Wireless Communications and Mobile Computing*, Vol. 4, April 2004, pp. 315-329.
20. S. Thomson, T. Narten, *IPv6 stateless address autoconfiguration*, RFC 2462, Network Working Group, *http://www.faqs.org/rfcs/*, 1998.
21. M.R. Thoppian, R. Prakash, A distributed protocol for dynamic address assignment in mobile ad hoc networks, *IEEE Trans. Mobile Computing* 5 (2006) 4-19.
22. S. Toner, D. O'Mahony, Self-organising node address management in ad hoc networks, in: M. Conti *et al.* (Eds.), *Personal Wireless Communications* (PWC 2003), IFIP-TC6 8th International Conference, Lecture Notes in Computer Science, Vol. 2775, Springer-Verlag, Berlin, 2003, pp. 476-483.
23. N.H. Vaidya, Weak duplicate address detection in mobile ad hoc networks, in: *Third ACM International Symposium on Mobile Ad Hoc Networking and Computing* ((MobiHoc 2002), Proceedings, ACM Press, June 2002, pp. 206-216.
24. Zeroconf Working Group, *Internet Engineering Task Force* (IETF), *http://www.zeroconf.org*, 1999.
25. H. Zhou, L.M. Ni, M.W. Mutka, Prophet address allocation for large scale MANETs, *Ad Hoc Networks* 1 (2003) 423-434.