jet INTL JOURNAL OF ELECTRONICS AND TELECOMMUNICATIONS, 2016, VOL. 62, NO. 2, PP. 175–178
Manuscript received November 21, 2015; revised June, 2016. DOI: 10.1515/eletel-2016-0024

Performance of IP address auto-configuration protocols in Delay and Disruptive Tolerant Networks

Radosław O. Schoeneich, Patryk Sutkowski

***Abstract*—At this moment there is a lack of research respecting Mobile Ad-hoc Networks (MANET) address assignment methods used in Delay Tolerant Networks (DTN). The goal of this paper is to review the SDAD, WDAD and Buddy methods of IP address assignment known from MANET in difficult environment of Delay and Disruptive Tolerant Networks. Our research allows us for estimating the effectiveness of the chosen solution and, therefore, to choose the most suitable one for specified conditions. As a part of the work we have created a tool which allows to compare these methods in terms of capability of solving address conflicts and network load. Our simulator was created from scratch in Java programming language in such a manner, that implementation of new features and improvements in the future will be as convenient as possible.**

***Keywords*—address assignment, auto configuration, MANET, DTN.**

I. Introduction

DELAY and Disruptive Tolerant Network (DTN) [1] is composed of highly mobile, wireless nodes which cooperatively form a network. The DTN network origins from Mobile Ad-Hoc Networks (MANET) and has similar characteristics. A network node usually communicates directly with other nodes within its range, and is independent from any infrastructure. For a long range communication the node uses a multi-hop communication through other nodes in the network. The extension of DTN network is, in contrast to the MANET network where the path set up is necessary, the use of carry messages between isolated nodes and sub-networks. It is done by the use a store-carry-forward paradigm [2, 3].

A high mobility of network nodes is the reason for changes in the network topology and its frequent disruptions. The classic multi-hop ad-hoc routing protocols like DSR [4], AODV [5], OLSR [6] etc. are in-effective in this difficult conditions. Therefore, DTN communication is done by specialized routing protocols like Spray-and-Wait [7], Spray-and-Focus [8], Prophet [9], Maxprop [10], BubleRap [11] and Rapid [12].

Most of DTN routing protocols are IP-based. Most of research effort bypasses the issue of node configuration and network address assign. Usually, it is assumed that nodes in network are configured in advance, before the network is established. Due to the mobility of nodes, it should be able to enter and leave the network. Therefore all nodes should have procedures for dynamic address configuration. The uniqueness of assigned IP address should be maintained despite the separation between sub-networks, and should not be changed during the network node activity.

The standard dynamic address configuration protocols known from a wired network like DHCP [13] and SAA [14] are ineffective in ad-hoc environment, because of a centralized server as a basic approach. Since MANET is distributed in nature and there is no centralized point of administration, this approach cannot be taken. Hence, some protocols for MANET networks have been proposed. All this work has been done with the assumption of at least one path between any pair of nodes in the network. Some solutions like DAD [15] and based on binary split (Buddy System) [16] are very popular.

In DTN network a path formulation is often impossible, and the nodes work separately. There are no DTN specific auto-configuration protocols. Therefore, the question how the MANET auto-configure protocols work in difficult DTN environment appears. In this paper we present the simulation results of selected MANET protocols. The selection was done based on popularity of each solution.

The paper is organized as follows: Section 2 presents existing auto-configuration protocols, their performance, and reasons for selection. Section 3 presents simulation assumptions and results. The work is concluded in Section 4.

II. Basic Auto-Configuration Protocols

The IP address auto-configuration in wireless ad-hoc networks can be divided into three basic categories: (a.) conflict detection, (b.) free allocation, (c.) best-effort allocation. The first group of algorithms is based on finding a free IP-address for a new node based on detection of conflicts. The idea of this solution is that the new node initially chooses an IP address, and sends the question for acceptance to all other nodes in the network. The conflict is detected if the sending node receives negative response. The negative response is sent by the node with the same IP address. If the address conflict is detected a new address is chosen and the procedure is repeated until there is no duplicate address. Once the procedure is completed, a new chosen address is stable and it is set as permanent. The conflict-detection algorithm is the protocol proposed in [15]. Other solution dedicated for IP v6 was proposed in work [17]. The procedure is called as Duplicate Address Detection (DAD). This solution is characterized by a lack of routing protocol dependency, it is fully distributed and it does not use

R. Schoeneich is with the Institute of Telecommunications, Warsaw University of Technology, Warsaw, Poland, e-mail: rschoeneich@tele.pw.edu.pl

P. Sutkowski is with Faculty of Electronics and Information Technology, Warsaw University of Technology, Warsaw, Poland, e-mail: sutkowski.p@gmail.com

any centralized server. The solution does not support nodes and networks merging. For this reason there exist modifications which handle network merging like Weak Duplicate Address Detection [18]. This solution uses proactive routing protocols and requires some modification to routing protocols.

Free of conflict allocation algorithms are the second group of protocols. The main idea of this solution is the assignment of an unused IP address to a new node which is based on collective work of the nodes taking part in allocation by using disjoint address pools. Disjoint pools make sure that the allocated addresses are different. Dynamic Configuration and Distribution Protocol (DCDP) proposed in [19] is an example of this group of solutions. The idea of this protocol is that every new node joining to the network receives half of the address pool from the first node in the network with which it had communicated. The main advantage of DCDP is that it takes into account network partition and merger, because if the network becomes partitioned, the nodes in different partitions still have different address pools. This means that the allocated IP addresses are different as well. For the subnetwork merge, there is no necessary further work to be done. Another improved solution is presented in [20] where the pool is split into equal parts in the whole network.

The third group of protocols are best-effort allocation algorithms. The solution works basing on nodes responsible for allocation which assign an unused address to a new node as far as possible. At the same time, the new node uses conflict detection to guarantee the free address. A Distributed Dynamic Host Configuration Protocol (DDHCP) is an example of the best-effort allocation protocol. This solution maintains a global allocation state, which means that all mobile nodes are tracked. This way is known which IP addresses have been used, and which addresses are still free [21].

A full discussion of the various address assignment protocols is presented in the documents of MANET Zero-config Group [22] and [23]. For the purposes of this study we has chosen basic, but reliable and popular protocols designed for MANET networks. Selected protocols are Strong Duplicate Address Detection [15], Weak Duplicate Address Detection [18] and based on binary split idea [20] which we call Buddy. We compare their performance in difficult environment of DTN networks.

III. Simulations

A. The Simulator

We made simulations based on our specially designed tool. The basic assumption was that the tool has to support events which are the key features of DTN, such as frequent disconnections or network merging. They could be a challenge in a field of address assignment. Therefore, the following features determine if chosen algorithm suits conditions well: a network simulator creates nodes, and connections between them, it merges and disconnects networks, and collects desired statistics like number of resolved conflicts, a number of messages relayed trough the network during one simulation etc.

Simulation tool is a program written in Java programming language. It is built of objects executing various tasks, starting from reading options out of certain option file, representing network nodes, connections and data flow, through log creation, and finally gathering statistics. The number of objects is not fixed, because of the adding new features to the simulator for future work.

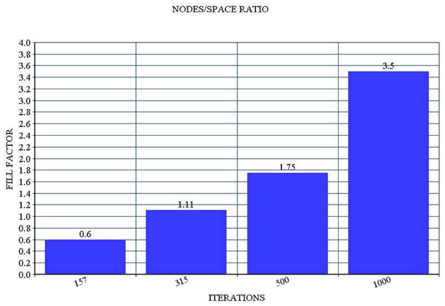

Fig. 1. Relation between number of nodes and address space

There are two most fundamental objects this project is built on: Supervisor and Simulator. Supervisor is responsible for reading simulation options, instructing lower level classes to take certain actions and running actual simulation. Each mode (Buddy, Strong, Weak) has its own supervisor, e.g. Supervisor Buddy. These lower level supervisors inherit from main Supervisor and are responsible for various actions taken in network simulations such as: (a.) Creating new single nodes or new networks, (b.) Creating new nodes connected to existing network, (c.) Connecting already existing nodes within the same network, (d.) Connecting nodes between different networks, and (e.) Disconnecting nodes from networks. Actions listed above are the most basic and they are extended by more specific actions, depending on used mode such as different ways of looking for conflicts.

Simulator is an object responsible for choosing right action before each cycle, basing on the settings given by Supervisor. Afterwards, it passes that information to Supervisor, which instructs its inheritors about actions they have to take. These two main objects provide core methods which allow certain events to happen.

Objects connected with node management are the lower level of application objects necessary to run it. Each mode has its own node management object which is responsible for storing and processing various information, i.e. ID, address, lists of neighbours, routing tables and others.

Separate group of methods is held in objects connected with utility tools. These tools are not required to run simulator, but they are necessary to have any knowledge about actions that are happening inside. These tools are used to analyse network, to gather data, and to present them in a readable and usable form.

B. Results

This section presents results obtained by simulations. The first result which we present is Fill factor.

Fill factor (see Figure 1) is the ratio between total amount of nodes created in the network and the total number of available addresses in the simulation. It helps to determine

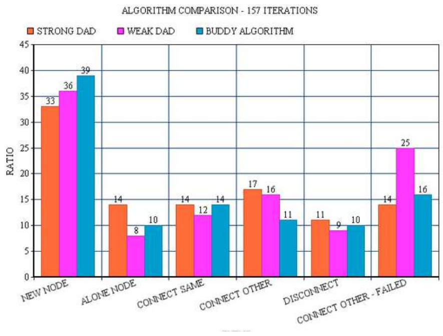

Fig. 2. Success rate.

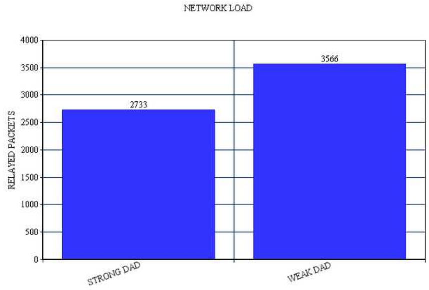

Fig. 3. DAD - network load.

what configuration the user has chosen. Fill factor has a significant influence on received results and it is important to keep it in mind. The ratio is calculated for the worst scenario when all created nodes are present in networks at the same time, which hardly ever happens because of disconnections.

The Figure 2 presents how many of the nodes created during this simulation caused conflicts. Next columns present percentage of conflicts that were resolved successfully. If we discuss small networks containing around 60-70 nodes created during 157 iterations (FF=0.6) BUDDY and Weak DAD algorithms do not have problems with solving conflicts. Free address space is big enough so that, in case of conflict, it is easy to find a replacement. It is clearly visible, that Strong DAD effectiveness departs from other two competitors. Despite the fact of the existence of many free addresses, Strong DAD does not handle network merging. These 19% of conflicts that occur are partially conflicts of new node joining the network and partially conflicts connected with network merging. Taking into consideration free address space, we can deduce that all of new node conflicts (71%) were solved successfully while failed attempts are connected with network merges.

Figure 3 shows the number of relayed ICMP packets during one simulation run. The 27% growth of Weak DAD in comparison with Strong DAD is justified by the complete elimination of non-solved conflicts during network merges. Such price is worth paying. In this case the size of the network is quite small, therefore the number of transmitted packets is not breathtaking. With the growth of the number of devices in the network, number of connections may grow even faster. At some point it becomes problematic, as in epidemic routing message is relayed from one node to every device connected with it, when generated traffic starts to load resources significantly.

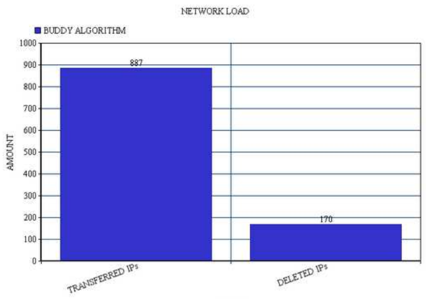

Fig. 4. Changes in address pools.

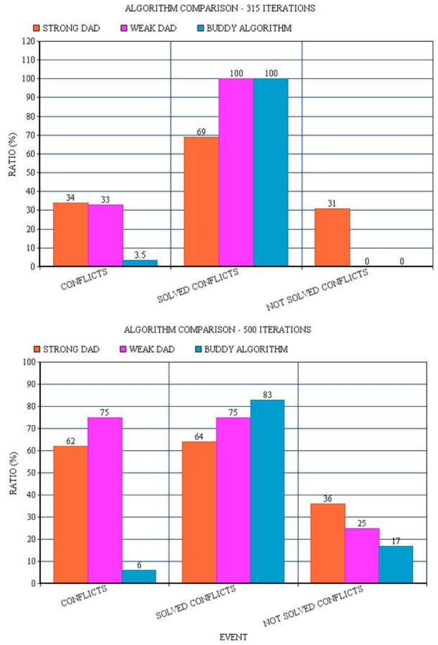

Fig. 5. Success rate.

Figure 4 presents how many changes were done to address tables in context of two events. Every event of transferring IP address to neighbouring node during its configuration or deleting addresses from address pools forces resynchronization of the whole structure. The traffic generated by this synchronization is even bigger than the one caused by multiple ping request in case of Strong DAD or Weak DAD.

On the basis of Figure 5 it is clearly visible, that growth of fill factor has a great influence on the number of conflicts in case of Strong and Weak DAD. However, it did not have a great influence on the effectiveness of conflicts solving. Although the number of nodes is close to the number of available addresses, it did not cause any network to be almost filled. It was rather the case that nodes spread among available networks. It is worth to notice that in case of the Buddy algorithm there is a number of nodes rejected by network due to the lack of free address in master node. As it is presented in a Figure 5, 62% of connecting nodes, which in this case is total to 40 nodes, is rejected by the algorithm. Extended versions of

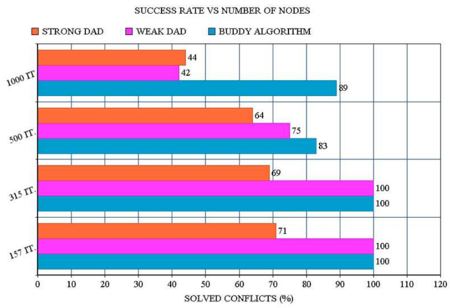

Fig. 6. Influence of fill factor on success rate.

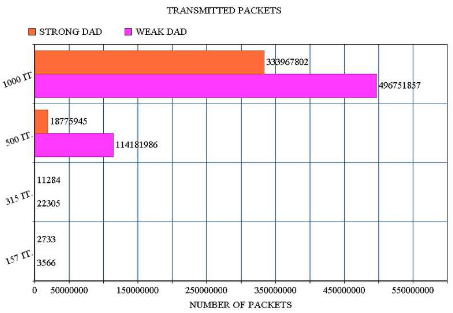

Fig. 7. Cost comparison.

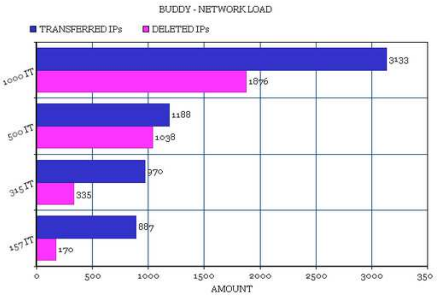

Fig. 8. Table maintaining cost.

Buddy are able to borrow addresses from master neighbours if there is such necessity, but in the basic algorithm we implemented as reference, one of this disadvantage becomes serious. Full statistics presenting the number of node rejects due to the lack of address in master node while there are still addresses available in total versus the fill factor is presented in the later part of this work. More than half of nodes for Strong and Weak DAD ran into its duplicate during process of first initialization or merging. The Weak DAD did not manage to solve all conflicts before passing the maximum number of trials, which was set to 10000. In case of a bigger number of trials allowed, simulation time becomes inconveniently long. During the process of network filling it is harder and harder to blindly pick a free address. Every Weak DAD repetition generates, if taking bigger networks into consideration, huge network traffic caused by ping messages alone.

Figure 6 presents relation between the number of iterations corresponding to Fill factor and the ability to solve conflicts. None of the tested algorithms, in case of the smallest simulated network, have problems connected with configuring new node joining certain network, due to the excess number of addresses in comparison to created nodes. Score of 71% in case of Strong DAD is achieved in the context of network merges, which situations the Strong DAD algorithm can not handle. With Fill factor of 1.11 for 315 iterations, both Weak DAD and Buddy are able to handle nodes joining already existing networks. Created nodes are spread among few networks, so probability of filling in completely one of them is relatively low, considering simulation settings. Further increase of the fill factor causes the fall of success rate down to 40%. Relatively good score of Buddy algorithm is caused by the fact that it does not allow a node to connect in case of lack of free address in certain node. It is obviously a disadvantage and it is presented on a Figure 9, later in this paper. We can also draw obvious conclusion, that with the growth of the network size, the kind of algorithm used to assign addresses has less and less impact, because all of them fail in case a nodes trying to join full network.

Figure 7 presents the cost, in terms of transmitted packets, which is paid for solving conflicts. When the networks are small and probability of solving a conflict in one or few trials is high, network is not flooded by information propagating constantly back and forth. In this case, the cost of reassignments is worth paying. On the other side of the coin are situations in which network is almost full and merges or tries to accept the incoming node. Let us assume that there are 124 out of 126 nodes in one of the networks. There are only two addresses that can be accepted which give us 1.5% chance of resolving conflict. Theoretically, we should pick the available address in less than 100 trials, but if we are unlucky, it may take many more. This situation becomes even worse when there is no address available at all. Reassignment is then done 10.000 times as there is no protection again repeating the same trials, which is maximum set by us, as algorithms provide no feedback information to connecting node i.e. network is already full, try again later. This leads to generation of huge, unnecessary, traffic.

The Figure 8, similarly to the Figure 7, shows the cost which we have to pay for maintaining the list of available addresses in every node by every node. Thanks to this the feature algorithm is able to prevent conflicts, which was clearly visible in figures above. These numbers of transferred and deleted addresses may not look so impressive as number of transmitted packets in case of 1000 iterations of DAD algorithms, but we must remember that nodes have to synchronize all the address tables. As a result, every change of address pool connected to transfer or to deletion forces resending all tables among whole networks which generates huge traffic comparable or even bigger that one caused by DAD.

The buddy node rejects are presented in the Figure 9. The amount of rejected nodes is not, in to a large extent, dependent on the number of nodes created during simulation. It is due to the fact of dividing free address spaces into halves and passing them to connecting node. After few such divisions in one line, endpoint node has no addresses to pass left and the new device trying to connect to this node is rejected by the network. As

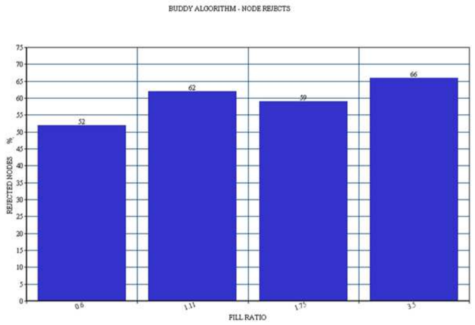

Fig. 9. Buddy node rejects.

the network grows, there is more and more of such empty, in terms of addresses and nodes. Picking randomly one of them becomes more probable. We must also notice, that the root of the network, if only it is connected to the one node in the beginning of the simulation, holds half of addresses available in the whole network, which is an inefficient way of resource management.

In our research we compared three address assignment algorithms simulated in DTN environment. Basing on obtained results, we are able to draw the following conclusions. Strong DAD algorithm is the simplest among all tested solutions. It does not require big computational power. However, it has one critical disadvantage, i.e. it does not support network merges. There is no tool designed to handle already configured nodes joining together. While in the traditional stationary wireless network it would not cause problems, but in DTN this makes this solution not acceptable. In this environment, the constant movement is the basic feature of the network, so any solution that does not support it, must be discarded.

The situation looks better when we take into consideration Weak DAD algorithm. Here the mobility support is implemented. Conflicts after network merges can be detected thanks to the unique key generated by each device generated most often on the basis of its physical address. Another advantage is that it is quite easy to predict when algorithm would behave well. During normal network conditions, when we can predict the number of nodes in the network, we are designing and aptly adjust network mask, i.e. sensor network, then it has very good success rate in terms of solving conflicts. Unfortunately, it starts to cause problems as the network fills in. It is difficult to assign an address which is not already occupied, and every trial generates traffic which grows to really great numbers.

The last of the tested solutions, Buddy algorithm, has completely different way of approaching the problem. Surprisingly, it has a really good ability to prevent conflicts. If we use this algorithm in our network address conflicts occur around 10 times less than in case of the remaining solutions. Of course, this does not come without cost. It happens way too often that connecting node is rejected due to the lack of free addresses in master node, even though there still available addresses in the network as a whole. If specifics of the network being designed demand low conflict occurrence rate, but at the same time, we can pay in node rejects, this solution is perfect. The Buddy algorithm is no perfect solution. It has one other important disadvantage. To prevent conflicts well, it has to store free address tables of every node in every node. It demands significant amount of memory, processing power and bandwidth to exchange all these data in case of any changes. The leaked addresses connected with leaving nodes may be recalculated and returned by the algorithm to the network if only we want to spend additional resources on it.

The choice between the three methods in terms of DTN is obvious only in case of Strong DAD. It simply cannot be used in such specific kind of network, as it fails to fulfill the most basic requirement. The remaining two may be switched between, accordingly to various conditions, such as the available infrastructure or network specifics. As always, there is no universal solution, but considering pros and cons of certain solutions in chosen environment, we are able to choose optimal one for our use.

IV. Conclusion

In this paper we have presented performance analysis of IP address auto-configuration solutions in DTN environment. We decided to create this work due to the lack of research respecting to MANET address assignment methods in Delay-Tolerant Networks environment. Our research allows us for estimating effectiveness of a chosen solution and, therefore to choose the most suitable one for specified conditions. For simulations we chose the popular solutions, such as: Strong DAD, Weak DAD and solution based on binary split of addressing pule. For this work we proposed simulation tool which was designed for address assignment performance analysis. Our simulation results allow us for estimating of the effectiveness of a chosen solution and, therefore, choose the most suitable one for specified conditions.

References

[1] S. Jain, K. Fall, and R. Patra, „Routing in a delay-tolerant network”, *In Proc. ACM SIGCOMM*, 2004

[2] D. Jea, A. Somasundara, and M. Srivastava, „Multiple Controlled Mobile Elements (Data Mules) for Data Collection in Sensor Networks”, *In Proc. IEEE/ACM International Conference on Distributed Computing in Sensor Systems (DCOSS)*, June 2005.

[3] R. Shah, S. Roy, J. Sushant, and W. Brunette, „Data MULEs: Modeling a Three-tier Architecture for Sparse Sensor Networks” *In Proc. IEEE SNPA Workshop*, May 2003.

[4] D. Johnson, D. Maltz, and J. Broch, „DSR: The Dynamic Source Routing Protocol for Multi-Hop Wireless Ad Hoc Networks”, *in Ad Hoc Networking*, chap. 5, pp. 139–172, Addison-Wesley, 2001

[5] C. E. Perkins and E. M. Royer, „Ad hoc on-demand distance vector routing”, *In The Second IEEE Workshop on Mobile Computing Systems and Applications*, February 1999

[6] T. Clausen, P. Jacquet, A. Laouiti, et al., „Optimized Link State Routing Protocol, *RFC 3626*, 2003

[7] T. Spyropoulos, K. Psounis, and C. Raghavendra, „Spray and wait: An efficient routing scheme for intermittently connected mobile networks”, *In WDTN 05: Proceeding of the 2005 ACM SIGCOMM workshop on Delay-tolerant networking*, 2005.

[8] T. Spyropoulos, K. Psounis, and C. Raghavendra, „Spray and focus: Efficient mobility-assisted routing for heterogeneous and correlated mobility”, *In Fifth Annual IEEE International Conference on Pervasive Computing and Communications Workshops*, 2007.

[9] A. Lindgren, A. Doria, and O. Scheln, „Probabilistic routing in intermittently connected networks”, *In Proceedings of the Fourth ACM International Symposium on Mobile Ad Hoc Networking and Computing (MobiHoc 2003)*, 2003

[10] J. Burgess, B. Gallagher, D. Jensen, and B. N. Levine, „MaxProp: Routing for vehicle-based disruption-tolerant networks", *In Proc. IEEE INFOCOM*, April 2006.

[11] P. Hui, J. Crowcroft, and E. Yoneki, „Bubble rap: Social-based forwarding in delay-tolerant networks", *IEEE Transactions on Mobile Computing* vol. 10, pp. 1576-1589.

[12] A. Balasubramanian, B. N. Levine, and A. Venkataramani, „DTN routing as a resource allocation problem", *In Proc. ACM SIGCOMM*, August 2007.

[13] R. Droms, „Dynamic Host Configuration Protocol, *Network Working Group RFC 2131*, March 1997

[14] S. Thomson and T. Narten, „IPv6 Stateless Address Autoconfiguration, *Network Working Group RFC 2462*, December 1998

[15] C. Perkins, J. Malinen, R. Wakikawa, E. Belding-Royer, and Y. Sun, „IP Address Autoconfiguration for Ad Hoc Networks, *Internet Draft*, Nov. 2001, http://tools.ietf.org/html/draft-perkins-manet-autoconf-01

[16] M. Mohsin, and R. Prakash, „IP Address Assignment in a Mobile Ad Hoc Network", *MILCOM 2002*, pp. 856-861, 2002

[17] K. Weniger and M. Zitterbart, „IPv6 Autoconfiguration in Large Scale Mobile Ad-Hoc Networks, *Proceedings of European Wireless 2002*, Florence, Italy, Feb. 2002

[18] N.H. Vaidya, „Weak Duplicate Address Detection in Mobile Ad Hoc Networks", *Proceedings of ACM MobiHoc 2002*, Lausanne, Switzerland. June 2002; pp. 206-216

[19] A. Misra, S. Das, A. McAuley, and S. K. Das, „Autoconfiguration, Registration, and Mobility Management for Pervasive Computing, *IEEE Personal Communication*, August 2001, pp 24-31

[20] A. Tayal, and L. Patnaik, „An address assignment for the automatic configuration of mobile ad hoc networks", *in Personal Ubiquitous Computing*, 2004.

[21] S. Nesargi and R. Prakash, „MANETconf: Configuration of Hosts in a Mobile Ad Hoc Network, *in proc. InfoCom 2002*, June 2002

[22] Bernardos C, Calderon M, H. Moustafa, „Survey of IP Address Autoconfiguration Mechanisms for MANETs", Nov, 2008. Internet Draft, http://tools.ietf.org/html/draft-bernardos-manet-autoconf-survey-04

[23] Zero Configuration Networking, http://www.ietf.org/html.charters/zeroconf-charter.html

[24] H. Moustafa, C. Bernardos, and M. Calderon, „Evaluation Considerations for IP Autoconfiguration Mechanisms in MANETs", draft-bernardosautoconf-evaluation-considerations-03(work in progress) November,2008.