J Netw Syst Manage (2011) 19:394–426
DOI 10.1007/s10922-010-9187-4

# SAAMAN: Scalable Address Autoconfiguration in Mobile Ad Hoc Networks

**Syed Rafiul Hussain · Subrata Saha · Ashikur Rahman**

Published online: 19 November 2010


**Abstract** Address autoconfiguration is one of the fundamental issues in mobile ad hoc networks (MANET). A node must need some form of identity before participating in any sort of communication. So each host in a MANET needs to be uniquely addressed so that the packets can be relayed hop-by-hop and delivered ultimately to the desired destination. Moreover, nodes in the MANET are free to move and organize themselves in an arbitrary fashion. Therefore any fixed infrastructure based solution for assigning identity (i.e. IP address) is not directly applicable to MANET. Under this infrastructureless and sporadic nature of the mobile nodes, several protocols of address autoconfiguration in the mobile ad hoc networks (MANET) have been proposed. Although some of these protocols perform decently in sparse and small networks, but exhibit poor performance (e.g., single point of failure, storage limitation, large protocol overhead and so on) when the network is either dense or very large. In this paper, we propose an efficient and scalable address autoconfiguration protocol that automatically configures a network

A preliminary version [1] of this paper has been published in International Conference on Ad-Hoc, Mobile and Wireless Networks (ADHOC-NOW 2009).

S. R. Hussain (✉) · S. Saha · A. Rahman
Department of Computer Science and Engineering, Bangladesh University of Engineering & Technology, Dhaka, Bangladesh
e-mail: rafiulhussain@csebuet.org

S. Saha
e-mail: subrata@csebuet.org

A. Rahman
e-mail: ashikur@cse.buet.ac.bd

S. R. Hussain
Ahsanullah University of Science & Technology, Dhaka, Bangladesh

S. Saha
Tiger IT Bangladesh Limited, Dhaka, Bangladesh

by assigning unique IP addresses to all nodes with a very low overhead and minimal cost. Evenly distributed Duplicate-IP address Detection Servers are used to ensure the uniqueness of an IP address during IP address assignment session. In contrast to some other solutions, the proposed protocol does not exhibit any problems pertaining to leader election or centralized server-based solutions. Furthermore, grid based hierarchy is used for efficient geographic forwarding as well as for selecting Duplicate-IP address Detection Servers. Through simulation results we demonstrate scalability, robustness, low latency, fault tolerance and some other important aspects of our protocol.

**Keywords** Duplicate address detection (DAD) · Duplicate-IP detection server (DDS) · DDS selection principle (DSP) · Geographic forwarding · IP address autoconfiguration · MANET

## 1 Introduction

A mobile ad hoc network (MANET) consists of a set of mobile transceivers communicating via single or multi-hop wireless links and functions without any predefined infrastructure. A node equipped with such a transceiver can join or leave the MANET at any time. In such infrastructureless environment, some of the notable challenging issues are routing, power consumption, security and network configuration. Among them network configuration includes IP address autoconfiguration, DNS server setup etc. IP address autoconfiguration is an inevitable issue not only in mobile ad hoc networks but also in all types of networks. Nevertheless, with the view of spreading quickly and easily (i.e., like a plug and play device) in situations like battlefields, disastrous areas etc., where there is no possibility and time to set up a fixed infrastructure, a very sophisticated issue like large scale *IP address autoconfiguration* in MANET should be focused with added emphasis.

The address autoconfiguration can be defined as a task of automatically assigning conflict-free unique IP address to every constituent node in the MANET without any manual intervention or without using any centralized DHCP [2] server. Upon arrival of a new node in a network, a unique conflict-free IP address must be assigned before it can participate in data communication. The IP address autoconfiguration protocol possess some of the characteristics given below:

1. The IP address assignment scheme must be distributed so that there is no single point of failure or bottleneck.
2. The assignment process should be quick enough since a node cannot participate in communication until it is configured with a unique IP address.
3. Autoconfiguration process should be scalable to accommodate a large number of nodes.
4. The protocol should be able to detect duplicate addresses when merging of two or more networks occurs and resolve quickly to avoid inappropriate routing.
5. It should handle some important aspects like address reclamation, address leakage, presence of multiple nodes at the time of network initialization etc.

Although there exists many solutions for IP management in traditional wired networks but those are not directly applicable in MANET environment. In wired network, a static IP address can be easily assigned by the network administrator to each host in the network, but in MANET, hosts join and leave the network dynamically making the manual IP address assignment inapplicable and impractical. Also a dynamic host configuration protocol (DHCP) can be used to dynamically assign IP addresses in traditional networks. However the caveat of this DHCP protocol is its dependency on some preconfigured hosts running as DHCP servers and assigning dynamic IP addresses to other hosts. MANET environment is assumed to work in a distributed fashion without any servers. Another protocol known as *Stateless Address Auto-configuration* (SAA) [3, 4] is used in the Internet for dynamically assigning IP addresses. Although in this protocol each host configures minimally required IP addresses in a distributed manner but the routers need to assign a globally *unique address prefix* within the same link of user nodes. SAA can not be directly applied to MANET because of its dependency on some special network elements such as router which is absent in MANET. Moreover, the topology changes frequently in MANET which makes it impossible to assign any globally unique prefix.

Prior works on autoconfiguration in MANET can be classified into two major groups: *stateless* and *stateful* address autoconfiguration. Nevertheless, a mixture of these two classes, called *hybrid* autoconfiguration, is also possible. In *stateless* autoconfiguration [5, 6], nodes in MANET do not store any IP address allocation information. A newly joined node randomly picks up an IP address and runs an algorithm called Duplicate Address Detection (DAD) algorithm [6] within the entire network to ensure that the chosen IP address is unique. If a conflict is found by a node with same IP address, the same process is repeated until a unique IP address is found. So, stateless autoconfiguration is also called *conflict-detection* address assignment scheme which adopts 'trial and error' policy to allocate a unique IP address.

On the contrary, in *stateful* autoconfiguration (e.g., [7] and [8]) a node acquires its unique IP address either from a centralized node or from one of the nodes from a set of distributed servers. The centralized nodes or the distributed servers are equipped with a pool of disjoint IP address blocks and keep records of all the IP addresses they have already allocated along with a list of available IP addresses. In this way, a unique IP address can be chosen without conducting any DAD and hence no conflict occurs. For this reason it is also called *conflict-free* autoconfiguration protocol. One major drawback of this class of protocols is nodes need to synchronize with each other in order to maintain the consistency of the IP address allocation information and to avoid duplicity.

Both stateful and stateless protocols provide limited *scalability*. In stateless protocols, every time a new node acquires an IP address, it floods the entire network with overwhelming DAD broadcast messages. With the increased network size, this kind of message broadcast is very costly and hinders scalability. One of the simple but naive non-scalable solutions for autoconfiguration is as follows: suppose there is a special node located in a well known position within the network which stores all the IP addresses assigned in the network. Let us call this special node a Central

Duplicate-IP Detection Server (CDDS). Any new node requiring an IP address picks up a random IP address and sends a query to this special node (i.e., CDDS) to verify whether this randomly chosen IP address is already chosen by some other node. If this CDDS replies positively (i.e., no node has chosen this IP address so far) then the node may safely assign this IP address to itself. Otherwise it discards this IP address, chooses another IP address randomly and repeats the same procedure. As this special node is located in a position which is previously known to all other nodes in the network, sending such query using geographic forwarding is easy. The problems with this approach are:

1. *A single point of failure:* Because there is only one special node which stores all the IP addresses, if this special node fails, so does the entire autoconfiguration scheme!
2. *Large database:* This single node needs to maintain a large database. With the increase of network sizes, the size of this IP address database also grows.
3. *Traffic volume:* This single special node would have to handle all Duplicate-IP address detection queries.
4. *Distant centralized database:* This single node cannot be *close to* all nodes. This will create a variable processing delay for different nodes located on different parts of the network.

Instead of such a naive centralized solution, we present a distributed approach of IP address autoconfiguration, dubbed as SAAMAN, to automatically configure a large scale MANET. The protocol works with the help of a special service offered by all the nodes called Duplicate-IP address Detection Service. Any node offering this service is called a Duplicate-IP address Detection Server (DDS in short). Instead of storing all the IP addresses in a central database belonging to a special node, here we distribute this database almost uniformly to all the nodes present in the network. Without any predesignation or pre-agreement, a node can act as a DDS for other nodes by keeping the information of position, speed and identity (i.e., IP address) of other nodes. Also, for a single node, a group of nodes simultaneously act as DDSs. With the help of a very simple principle (described in Sect. 3.3) the DDSs are efficiently selected for a particular node. This distributed duplicate-IP detection mechanism, not being centralized, eliminates the risk of single point of failure and yields duplicate-IP detection facility by copying the knowledge of a node at several DDSs. Also different subset of nodes become DDSs for different nodes which ensure load balancing effectively. Every node maintains a table called Duplicate-IP Detection Table (DDT). To facilitate scalability, a node's DDT contains information of only those nodes for which it acts as a DDS.

The paper is organized as follows. Section 2 describes the related works already done along this direction. Section 3 covers some preliminary concepts and definitions for better understanding of the proposed protocol. Section 4 describes the basic autoconfiguration scheme that we propose. A thorough extensive analysis of different critical problems and/or issues along with their tentative solutions are also presented in Sect. 5. Rigorous experimental results showing the effectiveness and robustness of the proposed autoconfiguration protocol are presented in Sect. 6. An analytical comparison of our proposed scheme with other DAD based protocols is

demonstrated in Sect. 7. Section 8 presents directions for future works and finally Sect. 9 concludes the paper by summarizing special features towards efficiency and scalability of the proposed protocol.

## 2 Related Works

Even though scalability is a much sophisticated issue for address autoconfiguration in MANET, a very few works e.g., [9] and [10] in the current MANET literature address this issue seriously. The first approach to IP address autoconfiguration protocol for MANET, proposed by Perkins et al. [6], belongs to the category of stateless autoconfiguration. In this protocol the address pool of 169.254/16 is used by the MANET and is divided into 2 sets: temporary address pool (169.254.0.1–169.254.7.255) and permanent address pool (169.254.8.0–169.254.255.254). A newly joined node first randomly chooses a temporary address from the temporary address pool and uses it as its source address to send a query for Duplicate Address Detection (DAD). The query includes a randomly picked IP address from the permanent address pool and is flooded throughout the network. If no other node in the network is assigned with the address mentioned in the query, then no reply comes back. This causes a timer at that newly joined node to expire and thereby that address can be assigned accurately to that node as a permanent address. On the other hand, if there is a conflict, a reply message from the conflicting node is sent. Then, the newly joined node has to rescind that chosen IP address and again randomly picks another IP address from the permanent address pool. The procedure is repeated again until the node is correctly assigned with an available IP address. However, there are some drawbacks of this protocol. This scheme does not describe what will happen if multiple nodes concurrently select the same temporary address from the temporary address pool (1–2,047) during assignment session. And most importantly, the duplicate address detection procedure described in their scheme does not scale well as the network size grows. Furthermore, this proposal does not handle network partitions and as a result it does not fit well for ad hoc networks.

However, Weak DAD based protocol proposed by Vaidya [11] ensures no erroneous routing in spite of having two nodes the same IP address. This is accomplished by adding a unique key to each address to ensure the uniqueness. The key is of arbitrary length and is chosen once by each node either randomly or based on a Universal Unique ID. It incorporates this key in the routing messages. A node detects conflict if it receives two packets of same address, but different keys. Though there is infinitesimal probability that two nodes pick up same address and the same key, still it may happen and hence conflicts occur. But, the probability of conflict decreases with increasing key length. Since key length is not constant, it incurs extra overhead in routing packets for including the information of key length in Packet Header. Thus there is a trade-off between routing protocol overhead and probability of detecting conflicts.

The solution of Zeroconf working group [5] also uses DAD algorithm. It assigns every node a unique link-local address within the range 169.254.1.0–169.254.254.255. After selecting an address, the host checks to determine if the

address is already used by any other node. This approach focuses on wired networks and ensures link-local uniqueness of the address. It is required that every node in the network to be within the communication range of every other node in the network. For this reason the solution is incompatible to MANET as MANET does support multi-hop communication along with single-hop. To extend the solution to MANETs, DAD message(s) will have to be flooded throughout the network. Park et al. [12] propose a solution using IPv6 site-local address which incorporates a problem regarding subnet IDs. It uses Neighbor Discovery Protocol (NDP) messages for DAD and floods this forwarded NDP messages within the entire network which obviously introduces handsome overhead, message loss, network congestion and so on.

IPv6 stateless autoconfiguration [4] uses IPv6 instead of IPv4 addressing mechanism. So, addressing process is different from that of IPv4 based stateless autoconfiguration protocol. An IPv6 node has several addresses per interface (e.g., link local addresses and global addresses) and IPv6 address has a prefix and a suffix. A router broadcasts its address prefix intermittently to all nodes in its subnets. So, when a new node joins the network it receives the prefix from its edge router. After receiving that prefix, that newly joined node forms a global address using this prefix and a suffix which is obtained from the MAC address. It then uses its link local addresses as its source address on temporary basis to conduct a DAD on the entire Local Area Network (LAN) to check the uniqueness of its formed global address. If any other node in that LAN is already configured with that requested global address, it will negatively respond to that newly joined node. On the other hand, if no negative reply comes before the timeout, the requesting node assigns that global address as its permanent address. However, one of the drawbacks of this solution is: it is not guaranteed that all nodes and interfaces contain unique IEEE MAC identifiers which can be changed easily. Hence, duplication of IP address may occur which seriously causes problems in routing. Moreover, flooding during DAD procedure makes the scheme non-scalable. To overcome this scalability issue, an extension is proposed in [9] where a hierarchical structure is created by special nodes (called *Leader nodes*) for flooding during the DAD procedure in bounded areas rather than entire network. Each leader selects a subnet ID and issues *Router Advertisement (RA)* to the nodes which are within their scope i.e, to the set of nodes which are within $r_s$ hops away from itself. *Router Advertisement (RA)* contain subnet ID (i.e., network prefix) and leader node's link-local address as source address which are used to create the site-local-address for the nodes within the scope. Then leader nodes perform the DAD procedure only within their scope in order to avoid duplicate Interface IDs. Since the subnet ID has to be unique for each leader node, so DAD has to be performed among the leader nodes within the entire ad hoc network. However, the leader election algorithm required in this solution obviously hinders scalability. Furthermore, the cost incurred in maintaining such a hierarchical structure for scalability may be too high.

The solution proposed by Nesargi et al. [7] considers that all nodes share a common address space and know the current IP-address pool state. Hence, a synchronization is required among all the nodes in the network for consistency. In this solution (MANETconf [7]), a new node (*Requester*) entering the MANET

requests for configuration to its neighbors. One of the configured neighbors, selected through leader election algorithm, acts as an *Initiator* for the *Requester* node. Each node belonging to the network stores all the used addresses (Allocated list), as well as the ones that are going to be assigned (Pending list) to the new elements. This allows each node to know the available addresses at any time. After receiving request from a *Requester* the *Initiator* selects a free address from its own table and sends a broadcast message to all other nodes in the MANET to check the uniqueness of that chosen address. If the *Initiator* receives a positive acknowledgment from all nodes of the MANET it assigns the address to the new node and again broadcasts the allocation complete message within the entire MANET. Then the new node will receive the address table in order to work as an *Initiator* for assigning addresses to subsequent new nodes. Here flooding the entire network for at least 2 times is an undeniable requirement for each newly joined node that causes the problems like high volume of traffic, timing delay, network congestion, high protocol overhead and some complexity for keeping the information updated in all the nodes of a network. Nevertheless, per node storage threatens the reliability for keeping several data structures by each node. The above facts therefore yield poor scalability.

Mohsin et al. [8] propose a buddy system which uses binary buddies. Here, all buddy sizes are of power of two and each size is divided into two equal parts. Thus, every node has a disjoint set of IP addresses that it can assign to a new node without consulting any other node in the network. When a newly joined node requests an IP address, the *Initiator* divides its IP address pool into two equal halves and offers one half to the requesting node. The new node assigns itself an IP address from the acquired pool of addresses, storing the rest of addresses to configure other nodes later. The new node is now configured and is considered as the Buddy of the *Initiator*. Nodes synchronize the IP address blocks which they store to keep track of the assigned IP addresses and to detect any IP address leakage. Each node broadcasts its IP address pool to all other nodes in the network and each node receiving an IP pool from another node records the received information in its IP address table. Through this approach, the available IP addresses are organized in the form of a binary tree with a division of two identical blocks (Buddies) per level among the nodes in the network. The proposed mechanism can also maintain the consistency in case of graceful and abrupt departures. This way, the IP address allocation has disjoint address pools, and the nodes can be sure that the allocated addresses are unique. It is evident that the advantage of this method is that the IP-address pool will be allocated quickly. However, the cost for sending a large amount of control messages through broadcasting within the entire network in order to invoke an IP address is high. In addition it cannot guarantee a uniform distribution of the IP-address pools in the MANET. Tayal et al. [13] propose a solution where a newly joined (*Requester*) node also has to contact the *Initiator*. If the *Initiator* has the address pool, it divides it into two parts and allocates one part to the *Requester*. On the other hand if it does not have the address pool, this search message is forwarded recursively by all the nodes which do not have an address pool. If a node replies with its address pool, it marks half of its addresses as under allocation and wait for a confirmation message from the *Initiator*. The *Initiator* sends confirmation message to the node whose address pool it received first, and allocates the received

address pool to the requesting node. But this protocol has a severe disadvantage as it requires flooding to search nodes with available address pool and requires a waiting for non-deterministic time in case of exhaustion of all address pools. Another stateful approach, Dynamic Address Assignment Protocol (DAP) [14], also allows an *Initiator* to subdivide its available IP address set with newly joined (*Requester*) node. However, when a node has an empty address set, it asks for an address set reallocation. This reallocation and the detection of a possible address leakage can cause a high control load in the network, depending on how the addresses are distributed among nodes. Furthermore, network identifiers used in partition merging is not suitable as these identifiers give no information about the current set of nodes in each partition. Another drawback is storage capacity because of maintaining the list of address set in each node. Another dynamic address assignment [15] based on the buddy system also handles node mobility during address assignment, message losses, network partition and merging. However, network-wide flooding is required for address reclamation in order to avoid address leakage which turns the solution non-scalable for large networks. Moreover, handling partitions and merging may also incur high overhead due to globalized nature of this protocol.

Unlike above approaches ([7, 8, 13]), Zhou et al. [10] introduce an approach which is different in a way that every node generates new addresses with the help of a special common distributed allocation function. This idea is also known as Prophet Address Allocation scheme. This scheme describes a function that produces addresses which are unique in the network, i.e., no other node generates the same address. Hence, disjoint address space is created by each node and thus requires no flooding which in turn implies scalability. But, since there is hardly any function that is strong enough to guarantee a unique address, there is some small probability of address duplicity which is a fundamental problem for any autoconfiguration protocol in MANET. Though the protocol gives solution of partitioning and merging for small networks, it introduces lots of message losses in case of large networks which hinders scalability. Unlike also MANETconf [7], which deploys full replication, the distributed IP address assignment scheme [16], proposed by Sheu et al., requires no data replication. Here instead of every node (as in [8]), only coordinators keep available IP address pool and are thereby responsible for configuring newly joined nodes. Coordinators in the network form a virtual C-tree in order to periodically update their address allocation information to C-root, which is the first node that entered the network. Though the solution incurs lower latency, lower storage usage and less control overhead to configure small number of nodes, it does not perform well as the network size grows. Moreover, detection of departed nodes and maintenance of the IP allocation table of the entire network relies on the C-root which becomes bottleneck and thereby demonstrates serious drawback. Tinghui et al. [17], in their Quorum Based Autoconfiguration, propose two-level hierarchy to configure the MANET. IP address block is replicated locally in adjacent cluster heads. A newly joined node acquires IP from a cluster head, called *Allocator*, through Quorum voting mechanism where only a partial decision is sufficient to have a valid IP address. Though it reduces overhead satisfactorily, it does not ensure data consistency. However, in our proposed approach, there is no

leader or cluster head selection or election or voting which causes large protocol overhead in order to ensure consistency.

Ancillotti et al. [18] propose an AH-DHCP protocol to assign a globally routable IPv4 address to the mobile nodes of a multi-hop WLAN using the DHCP-based mechanisms already implemented in the wired part of the network. The limitation of this solution is that it does not divide the network into subnets, thus can only work with routing protocols that spread routing information about every host. Nonetheless, the solution assumes that gateways, which interact with DHCP server established in the wired part, are the first nodes to join multi-hop WLAN and are configured beforehand which is impractical to MANET like environment. Moreover, it does not consider any solution for stand-alone MANET which is the fundamental concern for any autoconfiguration protocol. Automatic IP Address Configuration (AIPAC) [19] protocol, proposed by Fazio et al., provides a solution with *Initiator* and *Requester* as in [7] for solving the problem (associated with the solution [6]) of two nodes using the same temporary IP address while conducting DAD algorithm. The *Initiator* randomly selects an address and broadcasts a query message to check whether the address belongs to any other host. Any node receiving this broadcast message checks its routing table and sends a negative reply to the *Initiator* if a match is detected. Then the *Initiator* again chooses randomly another new IP address and repeats the process. On the contrary, if no reply is received for a given time interval, the *Initiator* broadcasts once more the same message to ensure no loss of reply messages due to possible errors in wireless channels. If neither reply arrives, it implies that the address is not used yet. Then the *Initiator* notifies the *Requester* with the *NetID* of the network and the resolved IP address. However, AIPAC does not specify how it handles the situation of more than one node requesting for the same IP address at the same time. Hence, uniqueness in this scheme is not guaranteed. Also like strong DAD [6], communication overhead increases as more nodes join the network since the number of DAD trials are likely to increase before a free IP address is obtained.

In the protocol PACMAN [20] (hybrid autoconfiguration), though node gathers state information from ongoing routing protocol packets to save bandwidth, a node assigns address to itself as in stateless protocols. Globally unique address is found on the basis of MANET-local address. Run Length Limited (RLL) encoding is used to compress the address as well as to limit bandwidth. A node must know about the compression scheme which is a drawback of this work. Scalability issue is also ignored. Yuan et al. [21] propose a three-level hierarchy to automatically configure the MANET. Though it uses stateful addressing scheme, there is no distributed server to efficiently obtain IP addresses. A DAD is also run in the entire network to ensure uniqueness. However, in our approach, instead of running DAD in the entire network, a node just sends query to some selected servers to test the uniqueness of the chosen IP address. Natalia et al. in their literature [22] propose a filter-base autoconfiguration protocol. Bloom filter is used to detect and resolve network partitioning and merging, and Sequence filter is used to take IP addresses. When a node takes an IP address based on Sequence filter, it requires to run a DAD in the whole network to update sequence filters of all other nodes residing in the network. But in our proposed protocol there is no need to run the DAD in the entire network.

Moreover, the solution does not address the issue of several nodes simultaneously choosing the same IP address at the same time. Again if a node gracefully departs the network it must notify the entire network to update the Sequence filters of all other nodes to prevent address leakage. But when a node departs without notifying because of software crushing or power failure, all the nodes require to reset their Sequence filters and return to the network initialization procedure after reaching a certain threshold value of the Sequence filter. Moreover, to detect and resolve network partitioning and merging, all the nodes in the same network must have the same bloom filters. This costs extra overhead in the network. Whenever two network merges, one of the networks must reset bloom filters based on low priority partition.

## 3 Preliminaries

In this paper we introduce a concept called *Duplicate-IP Detection Service* to ensure uniqueness of an IP address in entire network. But before going into the deep, it is necessary to describe some preliminary issues involved in this protocol

### 3.1 Geographic Forwarding

Geographic forwarding is used in our protocol as the basis of routing packets from one node to another. In geographic forwarding, a node knows its position i.e., altitude, latitude and longitude from GPS which gives almost correct measurement. Every node then periodically informs its existence to all of its neighbors by broadcasting HELLO messages within one hop. A neighbor node, upon reception of the HELLO message (Fig. 1), allocates an entry for the source of the HELLO message into its *Neighbor Allocation Table* (NAT, hereafter) along with the source's IP address, position, velocity and time of the HELLO message received. Now consider a scenario where node *A* wants to communicate with another node *C* and has the location information of node *C* with the help of any location service (e.g., GLS [23]). Before sending a message to node *C*, node *A* appends *C*'s IP address and *C*'s current geographic position in the packet header. Then node *A* looks up its *Neighbor Allocation Table* to find a node *B* which is geographically closest to node *C*. If the node *C* and node *B* are the same node, then node *A* sends the packet to node *C* directly. Otherwise node *A* forwards the packet to the intermediate node *B*. This process is then repeated in node *B* and in all subsequent nodes until the packet is received by node *C*.

**Fig. 1** HELLO message body

| HELLO |
| --- |
| Source IP address |
| Source Position |
| Source Velocity |

### 3.2 The Architecture

To automatically organize *Duplicate-IP address Detection Servers* (DDSs), we exploit the architecture proposed for *Grid Location Service* (GLS) [23]. In this architecture the entire network topology is divided into several hierarchical grid structures. The grids are organized with squares of increasing sizes. The smallest grid is referred to as an Order-1 square. Four such Order-1 squares form an Order-2 square. Similarly, four Order-2 squares make up an Order-3 square and so on. In brief, the *Grid Formulation Rule* is: *Any Order-n* ($n \geq 2$) *square is composed of four Order-*($n-1$) *squares and any Order-n* ($n \geq 1$) *square is constituent part of one and only one Order-*($n+i$) *squares, where* $i = 1, 2, 3$, *to ensure no overlap. The rule followed by an Order-n square is that its lower left coordinates must be of the form* $(a \cdot 2^{n-1}, b \cdot 2^{n-1})$ *for integers a and b*. Figure 2 shows a sample grid hierarchy that follows the above rule.

Figure 2 depicts the network area up to Order-4 square. Hence, there are four Order-3 squares, each of which in turn contains four Order-2 squares. Again each of such four Order-2 squares contains four Order-1 squares. So, the above mentioned network has 64 Order-1 squares which are numbered from 1 to 64. Among these 64 Order-1 squares, the *1st, 2nd, 9th and 10th* squares form an Order-2 square which is named as *A* (as listed in Table 1). Thus there are total 16 Order-2 squares which are numbered from *A to P* as inscribed in Table 1. Among these 16 Order-2 squares *A, B, E and F* constitute an Order-3 square α (as shown in Table 1). Finally, 4 such

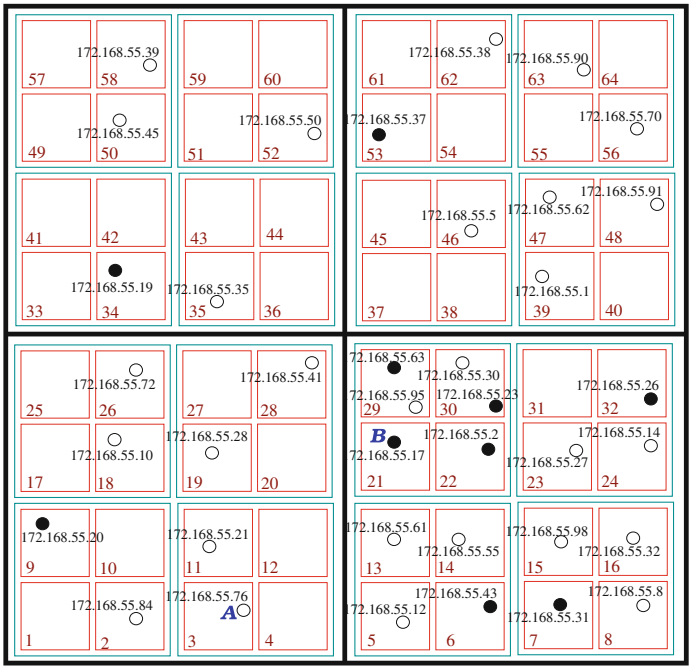


**Fig. 2** A DDS example

**Table 1** Grid hierarchy

| Hierarchy of squares | | | |
|---|---|---|---|
| Order-1 square | Constituent parts | Order-1 square | Constituent parts |
| 1 | NULL | 5 | NULL |
| 2 | NULL | . | NULL |
| 3 | NULL | . | NULL |
| 4 | NULL | 64 | NULL |
| Order-2 square | Constituent parts | Order-2 square | Constituent parts |
| A | 1, 2, 9, 10 | I | 33, 34, 41, 42 |
| B | 3, 4, 11, 12 | J | 35, 36, 43, 44 |
| C | 5, 6, 13, 14 | K | 37, 38, 45, 46 |
| D | 7, 8, 15, 16 | L | 39, 40, 47, 48 |
| E | 17, 18, 25, 26 | M | 49, 50, 57, 58 |
| F | 19, 20, 27, 28 | N | 51, 52, 59, 60 |
| G | 21, 22, 29, 30 | O | 53, 54, 61, 62 |
| H | 23, 24, 31, 32 | P | 55, 56, 63, 64 |
| Order-3 square | Constituent parts | Order-3 square | Constituent parts |
| $\alpha$ | A, B, E, F | $\gamma$ | I, J, M, N |
| $\beta$ | C, D, G, H | $\delta$ | K, L, O, P |
| Order-4 square | Constituent parts | | |
| $\xi$ | $\alpha$, $\beta$, $\gamma$, $\delta$ | | |

Order-3 squares $\alpha$, $\beta$, $\gamma$ and $\delta$ jointly complete the Order-4 square $\xi$. It is also to be noted that *28th, 29th, 36th and 37st* Order-1 squares or *F, G, J and K* Order-2 squares cannot form any higher Order-2 and Order-3 square respectively. So, any of such combination of lower Order squares cannot make any higher Order square which violates the *Grid Formulation Rule* given above (Fig. 3). The composition of entire network using different Order squares are shown in Fig. 4 as a hierarchy of grids.

### 3.3 Selection Process of Duplicate-IP Address Detection Server

Selection process of DDSs of a node is based on its current IP address and the predetermined grid hierarchy. Here we first describe which nodes are selected as DDSs for a particular node and then how they are selected through HELLO and UPDATE messages. For the grid hierarchy of Fig. 2, at most 10 DDSs can be selected for a node in different Order squares. Of these 10 DDSs, 1 DDS is from the node's own Order-1 square, 3 DDSs are from the node's three peer Order-1 squares, 3 DDSs are from the node's three peer Order-2 squares and 3 DDSs are from the

**Fig. 3** QUERY, NACK and UPDATE message body

| QUERY |
|---|
| Source IP Address |
| Source Position |
| Destination IP |
| Destination Square |
| Timeout Count |

| NACK |
|---|
| Source IP Address |
| Source Position |
| Destination IP |
| Destination Square |
| Timeout Count |

| UPDATE |
|---|
| Source IP Address |
| Source Position |
| Destination Square |
| Time Stamp |
| Timeout Count |

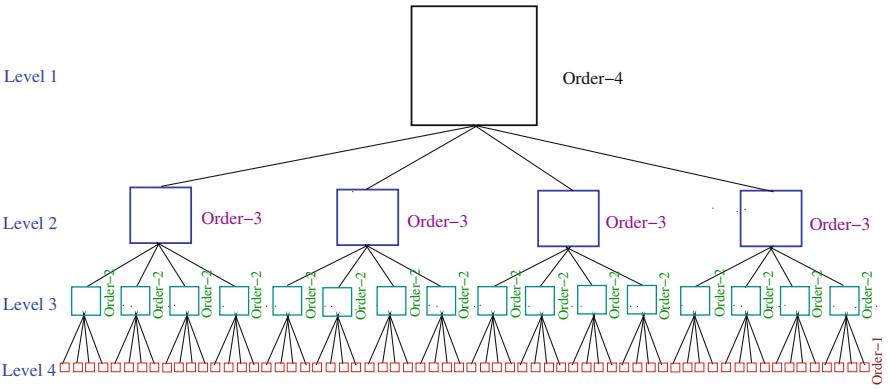


**Fig. 4** The composition of entire network as hierarchy of grids

node's three peer Order-3 squares. The different squares from which 10 DDSs (shown as filled circles in Fig. 2) are selected for node *B* are: *21st* Order-1 square; *29th, 30th* and *22nd* peer Order-1 squares; *C, D* and *H* peer Order-2 squares and $\boldsymbol{\alpha}, \boldsymbol{\gamma}$ and $\boldsymbol{\delta}$ peer Order-3 squares. Note also that, if no node is present in a square, then obviously no node in that square is selected as DDS for other nodes. So for node *A* (in *3rd* Order-1 square) not all 10 DDSs are selected since 2 peer Order-1 squares (*4th and 12th* Order-1 squares) are empty. Now which node in a square is selected as DDS for a particular node follows the principle called *DDS Selection Principle (DSP)*. The principle has 3 cases:

***Case*(*a*) :** *In an Order square, that node is selected as DDS whose IP address is least but greater than the IP address of a particular node for which a DDS is going to be selected. If no such node is present in that square go to Case (b).*
***Case*(*b*) :** *In an Order square, for a particular node one is selected as DDS whose IP address is absolutely least in that square. If no such node is present in that square go to Case (c).*
***Case*(*c*) :** *In an Order square, the node itself is selected as its own DDS.*

For the purpose of clarification the above three cases of *DSP* are explained with three specific examples in Fig. 5. The *Case (a)* (shown in Fig. 5a) arises when node *X* with IP address 195.125.70.10 has three neighbors with IP addresses 195.125.70.8, 195.125.70.11 and 195.125.70.12 in its Order-1 square. So, according

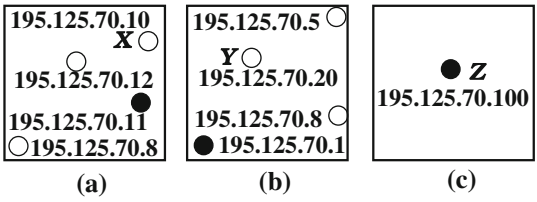


**Fig. 5** Examples of DDS selection principle (*DSP*)

to *Case (a)* node with IP address 195.125.70.11 is selected as *X's* DDS (shown as filled circle) since 195.125.70.11 is the least IP address between 195.125.70.11 and 195.125.70.12 which are greater than node *X*'s IP address 195.125.70.10. The *Case (b)* (in Fig. 5b) arises if node *Y* having IP address 195.125.70.20 has neighbors with IP addresses 195.125.70.8, 195.125.70.5 and 195.125.70.1 in its Order-1 square. In this scenario *Case (a)* is failed and according to *Case (b)* node with IP address 195.125.70.1 is selected as DDS for node *Y* since 195.125.70.1 is the least IP address among all other nodes' IP addresses (i.e., 195.125.70.8, 195.125.70.5 and 195.125.70.1). Finally, the *Case (c)* (in Fig. 5c) arises when the node *Z* with IP address 195.125.70.100 is the only node present in its own Order-1 square. Hence, the node *Z* acts as its own DDS in its Order-1 square. This above principle is also similarly applied to higher Order-n ($n = 1, 2, 3, \ldots$) squares to setup DDSs.

Figure 2 shows the selected DDSs of node *B*. As there is no other node in its own Order-1 square i.e, in *21st* Order-1 square, node *B* itself is selected (according to *Case (c)* of DSP) as its own DDS in that square. Then, three other DDSs in its three peer Order-1 squares (*29th, 30th and 22nd* Order-1 squares) are also chosen according to the DSP. Therefore, the node *B* itself and 172.168.55.63 (according to *Case (a)* of DSP) from *29th* Order-1 square, 172.168.55.23 (according to *Case (a)*) from *30th* Order-1 square and 172.168.55.2 (according to *Case (b)*) from *22nd* Order-1 square are selected as DDSs in its Order-2 square. Next, 172.168.55.43, 172.168.55.31 and 172.168.55.26 are also chosen as DDSs respectively from *C, D and H* peer Order-2 squares of node *B*'s Order-2 square. In peer Order-2 square *C* there are 4 nodes with IP addresses 172.168.55.12, 172.168.55.43, 172.168.55.55 and 172.168.55.61. Under *Case (a)* of DSP 172.168.55.43 is selected as DDS. Similarly the other nodes 172.168.55.31 and 172.168.55.26 are selected as DDSs respectively in *D and H* Order-2 squares under *Case (a)*. Again, 172.168.55.20, 172.168.55.19 and 172.168.55.37 are also picked up (according to *Case (a)*) as DDSs in $\alpha$, $\gamma$ and $\delta$ Order-3 squares respectively. Similar concept can be extended to determine DDSs at higher Order squares. For illustration and clarity purpose, we show the Fig. 2 only up to Order-4 square.

The above description of this subsection demonstrates the selection procedure of *Duplicate-IP Detection Servers* (DDSs) for a particular node. Now we describe how DDSs are selected efficiently through HELLO messages and geographic forwarding of UPDATE messages (message body is shown in Fig. 3) with the help of Figs. 2 and 6a. In our protocol, only the HELLO message is sufficient to select a DDS in a node's own Order-1 square. But, except the DDS in a node's own Order-1 square, all DDSs from other squares are selected dynamically only when UPDATE messages reach those squares. The most important requirement for node *B* to distribute its current information to the appropriate DDSs in an Order-n square is: *The nodes contained in that square have already distributed their current information throughout that square. As soon as the Order-n DDSs are operating, there is sufficient capability for geographic routing to set up the Order-*$(n + 1)$ *DDSs*.

The size of the smallest Order square (Order-1 square) in the grid hierarchy is deliberately chosen in such a way that all the nodes in that square are within their mutual transmission range, i.e., all nodes are able to know all other nodes in their

**Table 2** Partial content stored in DDT of the nodes in *G* Order-2 square

| Node | Content of DDT | Node | Content of DDT |
|---|---|---|---|
| 172.168.55.17 | 172.168.55.17 | 172.168.55.95 | 172.168.55.63 |
| 172.168.55.2 | 172.168.55.2 | 172.168.55.23 | 172.168.55.30 |
| 172.168.55.63 | 172.168.55.95 | 172.168.55.30 | 172.168.55.23 |

Order-1 square through the periodic HELLO beacons. In Fig. 2 and Fig. 6a, only node *B* is present in its own Order-1 square (i.e., 21st Order-1 square). As a result, no HELLO message is received by node *B* from any node in that 21st Order-1 square. Hence, node *B* has no information about any other node in that square and thus selects itself as DDS for itself in its Order-1 square. In the mean time, the nodes in other 3 Order-1 peer squares (*29th, 30th and 22nd* squares) also have already known their respective neighbors. Therefore, their DDSs also have been selected there with the help of each others' HELLO messages. DDT of these DDSs will also be updated. At that moment, the partial content of the DDT of each node in the Order-2 square *G* is shown in Table 2.

Under this circumstance, nodes in each Order-1 square of *G* Order-2 square have already disseminated their current information within their respective Order-1 squares. So after a little while of sending the first few HELLO messages, all nodes send 3 UPDATE messages to their 3 Order-1 peer squares. Node *B* sends UPDATE messages to *29th, 30th and 22nd* Order-1 squares using geographic forwarding as shown in Fig. 6a. We call it *Grid Forwarding* because rather than location and IP address of the destination node, only location of the destined square's midpoint is written in the destination field of the UPDATE message's packet header. The UPDATE message destined to *29th* peer Order-1 square is first caught by node 172.168.55.95 in that square. Then 172.168.55.95 checks whether it can act as DDS for node *B*. So it compares node *B*'s IP address with its own IP address and also with IP addresses stored in its DDT. It finds that 172.168.55.63 in its DDT is least IP address greater than 172.168.55.17 in its (i.e., *29th*) Order-1 square. So it determines 172.168.55.63 is worthwhile (according to *Case (a)* of DSP) to act as DDS for node *B* and forwards the UPDATE message of node *B* to 172.168.55.63. After receiving the UPDATE message, 172.168.55.63 also checks its DDT and ensures with its explored knowledge that no other node in its Order-1 square is further least node greater than 172.168.55.17 to become a DDS for node *B*. Hence, it is selected as the DDS and does not further forward this UPDATE message. On the contrary, node

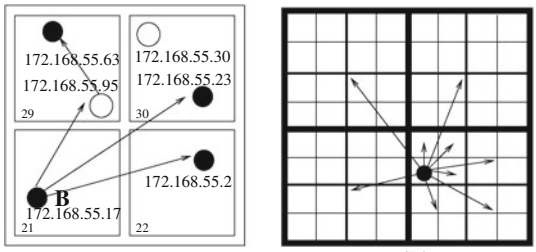


**Fig. 6** **a** Node *B's* UPDATE messages to its peer Order-1 squares. **b** 9 UPDATE messages to 9 different Order squares

**Table 3** Partial content stored in DDT of the nodes in *G* Order-2 square

| Node | Content of DDT |
|---|---|
| 172.168.55.17 | 172.168.55.17, 172.168.55.2, 172.168.55.95, 172.168.55.63, 172.168.55.30, 172.168.55.23 |
| 172.168.55.2 | 172.168.55.2, 172.168.55.17, 172.168.55.95, 172.168.55.63, 172.168.55.30, 172.168.55.23 |
| 172.168.55.63 | 172.168.55.95, 172.168.55.2, 172.168.55.17, 172.168.55.30, 172.168.55.23 |
| 172.168.55.95 | 172.168.55.63 |
| 172.168.55.23 | 172.168.55.30, 172.168.55.17, 172.168.55.2, 172.168.55.63, 172.168.55.95 |
| 172.168.55.30 | 172.168.55.23 |

with IP address 172.168.55.23 first catches the UPDATE message of node *B* destined to the *30th* Order-1 square and finds that it is the least node greater than 172.168.55.17. Hence, it acts as a DDS for node *B*. No further forwarding is also required and it stores node *B's* information (i.e., IP address, geographic position etc) in its DDT. UPDATE message transmitted for the *22nd* Order-1 square is received by 172.168.55.2 and it selects itself as DDS of node *B* as no other node is present there. In the same way, each node in the *29th, 30th and 22nd* Order-1 squares also send 3 UPDATE messages to their respective 3 Order-1 peer squares and thus DDSs for them in those squares are also properly selected. So DDT of all nodes in that Order-2 square are updated regularly through the periodic UPDATE messages. The current partial content of DDT of each node in Order-2 square *G* after sending of the UPDATE messages is shown in Table 3.

Table 3 shows that all nodes have already distributed their current information throughout the Order-2 square *G*. So when all nodes in an Order-1 square send UPDATE messages to their peer Order-1 squares, all nodes in those squares are able to know the most eligible node for acting as DDS in their Order-2 square (for any other node). In similar way nodes in other Order-2 squares also distribute their current information within their respective squares. However, node *B* then sends 3 UPDATE messages to its three peer Order-2 squares and subsequently three peer Order-3 squares. Thus DDSs are selected from those squares under the same procedure described above and therefore contents of DDT of them are also updated. So, like node *B* every node sends total 9 UPDATE messages to 9 different Order squares and hence, 9 DDSs are selected. These maximum 9 DDSs and 1 DDS in own Order-1 square sum up 10 DDSs for each node. A scenario of sending 9 UPDATE messages of a node is depicted in Fig. 6b.

## 4 Autoconfiguration Protocol

In the proposed address autoconfiguration protocol a node at first randomly chooses an IP address which we call its real IP address. This randomly chosen IP address becomes its final IP address when the node becomes sure that no other node in the network is currently using the same IP address i.e. there is no duplicity. However, to check for duplicity, we provide an intelligent mechanism. If the same IP address is

already assigned currently to another node, then there must exists several DDSs in the network for that node. By consulting only those DDSs the possibility of duplicity can be avoided. A node unicasts query messages to all of these (possible) DDSs to check for duplication. But the paradox is to conduct this query and to get back a reply successfully, the node needs an IP address! Therefore, at first we assign temporary IP address to this newly joined node. This temporary IP address is used just for the query-response purpose and later on this temporary IP address is discarded and reused by some other newly joined node within the same grid. This temporary IP address is guaranteed to be conflict-free. These two steps are described below in details.

### 4.1 Temporary IP Address Assignment

At the very first when the MANET is not initialized, we assume that several nodes simultaneously enter the network and they are connected, i.e., there exists at least one communication path among the nodes. Each Order-1 square is allocated with a predefined disjoint block of IP addresses which we call *temporary IP address pool*. Two Order-1 squares do not have any common IP address in their temporary IP address pools. We also assume that all nodes have prior knowledge of all temporary IP address pools before joining the network. The reasonable account of this assumption is as follows: if a MANET has 64 Order-1 squares and its total range of temporary IP address is from 1 to 2,048, then every single Order-1 square can use (2048/64 =) 32 temporary IP addresses. Hence, the *1st* Order-1 square's temporary IP addresses ranges from 1 to 32, *2nd* Order-1 square's temporary IP addresses ranges from 33 to 64 and so on. Since a node in the network knows its geographic position through GPS, it can easily determine the particular Order-1 square on which it resides. Therefore, if a node is able to know its residing Order-1 square, it can also determine the disjoint temporary IP address block (which is a part of total temporary IP address pool for the whole network) that is allotted for that specific square. Thus every node has prior knowledge about this disjoint block of temporary IP address pool in order to get a temporary IP address before assigning a real IP address.

A newly joined node determines its temporary IP address with minimal overhead as described as follows. When a node joins in the MANET, at first it identifies its position using GPS. From its position it can easily calculate the Order-1 square within which it is located. Then it chooses a conflict-free temporary IP address from the temporary IP address pool reserved for that Order-1 square. As every node knows all other nodes within its own Order-1 square (the size of an Order-1 square is such that all nodes in that Order-1 square are within their mutual transmission range), choosing of such conflict-free temporary IP address is easy. Therefore, a node randomly picks up an IP address from the temporary IP address pool reserved for that square and observes the *Neighbor Allocation Table*. It repeats the same process in case of conflicts. But conflicts in determining unique temporary IP address may still arise when several newly joined nodes choose the same temporary IP address simultaneously. To prevent such conflicts, every node runs a DAD algorithm within its Order-1 square after choosing temporary IP address. It is done

by one-hop broadcasting of DAD message within its own square. If a node finds a DAD message containing the address same as its chosen temporary IP address, it sends a NACK message. Receiving NACK message, that newly joined node gives up that chosen temporary IP address and randomly chooses another temporary IP address after a random amount of time. Thus a newly joined node gets a temporary address from the allocated temporary IP address pool for the smallest square where the node resides in.

The paradox is if the conflict-free temporary IP address can be achieved, why it is necessary to assign again the real IP address! Assigning real IP address following temporary IP address assignment is necessary for scalability purpose. If we assign disjoint blocks of real IP address to every Order-1 square then there might be a situation where the number of joining nodes in an Order-1 square is greater than the pre-assigned IP address pool for that Order-1 square. Hence, some of the joining nodes to that Order-1 square never get real IP address. Moreover, a newly joining node in an Order-1 square may not be able to avail any IP address even though that particular Order-1 square is currently empty. This may happen when that Order-1 square is left with no spare IP address as all IP addresses reserved for that square may be used up by the past/early joining nodes and these nodes now may have changed their locations to other squares. That is why our mechanism requires a disjoint blocks of predefined temporary IP addresses to every Order-1 square that are to be used by newly joining nodes at first place. Once a node acquires a conflict-free real IP address, it releases its temporary IP address. Then its released temporary IP address can be reused by some other newly joined nodes. This reusability helps in situations where the number of newly joining nodes in an Order-1 square is larger than the number of temporary IP addresses assigned for that particular square. In worst case, some joining nodes may need to wait until one of the nodes of its Order-1 square releases its temporary IP address.

### 4.2 Real IP Address Assignment

After resolving temporary IP address, a node randomly chooses a tentative (real) IP address. It then makes queries through QUERY messages to the best nodes (i.e., to DDSs) for the chosen real IP address in Order-n ($n = 1, 2, 3$) peer squares. If an entry is found in the *Duplicate-IP address Detection Table* (DDT) of any of those DDSs, the corresponding DDS immediately informs the node using NACK message. The node then chooses another tentative IP address randomly and the same process is repeated again after a random amount of time. The QUERY messages are sent iteratively. At first, the node sends queries to DDSs in peer Order-1 squares. If IP address conflict is detected in any Order-1 square, there is no need to send queries in peer Order-2 squares. In general, when an IP address conflict is detected in Order-n square, there is no need to send any further query to Order-$(n + 1)$ square or higher Order peer squares. If no conflict is detected in any of the DDSs at any Order, no reply is sent to the requesting node. Therefore, if the node receives no NACK message within a timeout interval, it assumes that the tentative IP address is conflict-free and finalizes this IP address as its real IP address.

How a query is accomplished is described here with an illustrative example. Suppose a node *A* in Fig. 2 with temporary IP address 172.168.55.76 randomly chooses 172.168.55.17 as its tentative (real) IP address. Note that, node *B* has already assigned this IP address 172.168.55.17 to itself and updated all its DDSs, but node *A* is not aware of this situation yet. After choosing tentative IP address, node *A* sends QUERY message to its own Order-1 square with the same principle for choosing DDSs as described in Sect. 3.3. If no NACK message is received within a predefined time interval, it sends three query messages to its three peer Order-1 squares (i.e., *4th, 11th and 12th* Order-1 squares) and the process is repeated again for higher Orders. In our example, no node in the *3rd, 4th, 11th and 12th* Order-1 squares is currently acting as a DDS for node *B* and hence, there is no chance of receiving NACK from any node in these squares. After predefined amount of time node *A* again sends three queries to its three peer Order-2 squares (i.e., *A, E and F* Order-2 squares). At this point, a node with IP address 172.168.55.20, currently acting as a DDS of node *B*, sends a NACK message to node *A* using geographic forwarding. If no NACK message is received from the highest Order squares, the node finalizes its chosen tentative IP address as its real IP address. For better understanding of the sequence of our protocol, a flow chart is given in Fig. 7.

### 4.3 How Many DDSs are Reasonably Needed

Within the proposed scheme every node in the network acts as DDS server for some other nodes. However, a single node has a fixed number of (multiple) DDSs which depends on the number of grids in the deployment area. The number of grids also implicitly determine the number of levels in the grid hierarchy (see Table 1 and Fig. 4). For example, a $2 \times 2$ grid has in total 4 small grids where we can form 2 levels of hierarchy, whereas a $4 \times 4$ grid has 16 grids in 3 levels of hierarchy, an $8 \times 8$ grid has 64 grids organized in 4 levels of hierarchy and a $16 \times 16$ grid has 256 grids in 5 levels of hierarchy. After determining the number of levels within the grid hierarchy it is pretty straightforward to compute the number of DDSs needed for each node. If there are exactly 2 levels then total DDSs for every node is 4, for 3 levels of hierarchy it is 7, for 4 levels it is 10. The general formula is, if there is exactly $n$ levels in grid hierarchy then there are exactly $1 + (n - 1) \times 3$ DDSs for each node.

## 5 Some Critical Issues

Various critical events may arise when a node wants to obtain an IP address. These events may occur when same tentative IP address may be chosen simultaneously by several nodes or there occurs frequent switching events from temporary IP address to real IP address of a node. These critical issues and their proposed solutions are described below with illustrative examples.

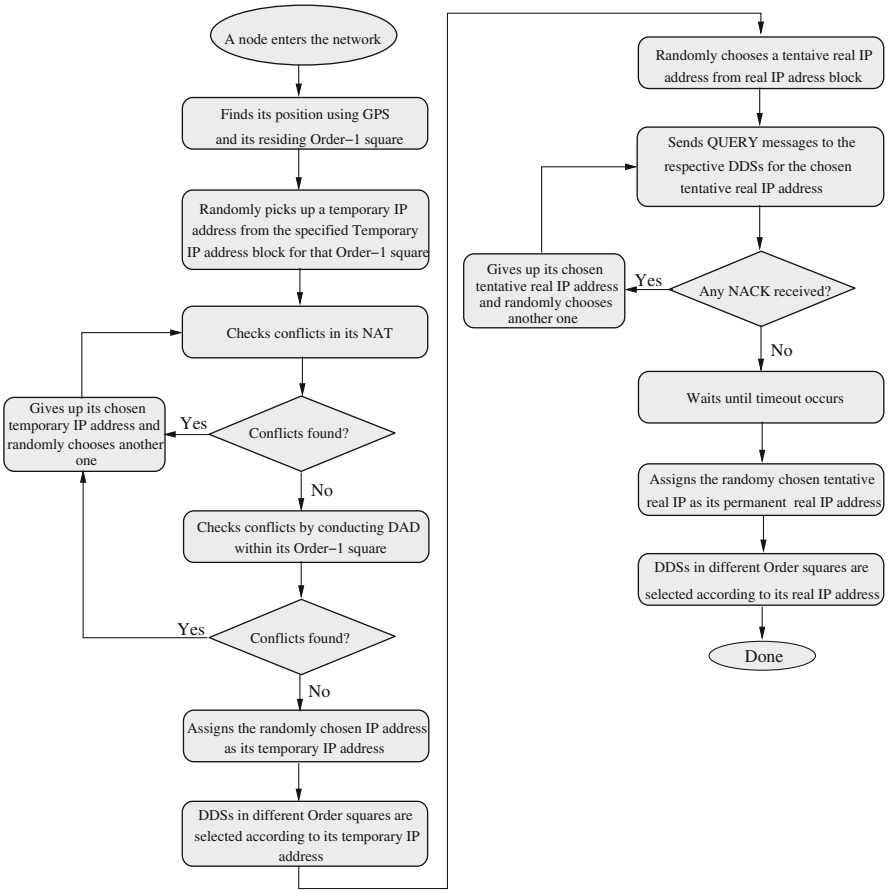


**Fig. 7** Flow chart of SAAMAN

## 5.1 Concurrent Initialization

Same tentative address may be randomly chosen by several nodes simultaneously or within a *transitive period*. Transitive period is the interval between the time when a node starts its tentative real IP address selection process and the time when it finishes updating all of its DDSs in entire MANET with its assigned real IP address. Obviously, if there is no mechanism in DDSs to distinguish the same requesting IP address from different nodes within this critical time, there must be duplicity of IP addresses in the network. This unwanted event is easily overcome by creating a REQUEST_QUEUE in every DDS. When a query message for an IP address comes to that IP address's DDS, it makes an entry in its REQUEST_QUEUE with ⟨ tentative IP address, temporary IP address, Timeout ⟩ tuple. If the server finds an entry already present in its queue with the same tentative but different temporary IP address pair, it immediately sends a NACK message to the requesting node and thus

avoids duplicate IP address assignment. Furthermore the entry of the REQUEST_QUEUE is deleted when an UPDATE packet is received or the timeout occurs.

### 5.2 Dynamically Changing DDSs

With the change of node's identity from temporary IP address to real IP address, evenly distributed DDSs of this node are also changed dynamically. As a result, query message may not hit the desired DDSs at that point of time. This event is now discussed elaborately by an illustrative example. Suppose, node *A* (in Fig. 2) with temporary IP address 172.168.55.76 randomly chooses its real IP address as 172.168.55.17 which is already taken by node *B*. Now, node *A* at first sends four query messages in its Order-2 square, i.e, in *4th, 11th and 12th* Order-1 square along with its own Order-1 square (i.e., *3rd* Order-1 square). It is evident from the figure that there is no node in its Order-2 square currently representing itself as a DDS of node *B*. So, there is no incoming NACK message to prevent the node *A* from obtaining real IP address. After a predefined amount of time, node *A* sends another three query messages to its Order-3 square (i.e, in *A, E and F* Order-2 square). As there is no DDSs in *E and F* Order-2 squares, there is no NACK message from these Order-2 squares. But in *A* Order-2 square, a node with IP address 172.168.55.20 is now currently acting as a DDS of node *B*. So, when the node *A* sends a query message to *A* Order-2 square, the first node with IP address 172.168.55.19 (node with temporary IP address 172.168.55.80 recently takes the real IP address 172.168.55.19 and does not get sufficient time to be updated by node *B* with UPDATE messages) catches the query message. As it is the best node of that Order-2 square, it does not retransmit the query message to the real DDS of node *B*—which is node with IP address 172.168.55.20. This problem is solved by preventing a node obtaining a real IP address recently from processing any query messages although it is the best node for the requesting tentative IP address for that square. So, it just sends a NACK message to the requested node. After some predefined amount of time it comes back to capable state and starts processing any incoming query.

### 5.3 Node(s) Having No Neighbor(s)

When a node enters the MANET and finds no neighbor, it only takes temporary IP address randomly. There is no need to take real IP address. When the network begins to converge (i.e., when the node slowly discovers its neighbors), it then chooses a real IP address. This scenario is described above. In Fig. 8a, there is only one node in the entire grid and in Fig. 8b, there are only two nodes but have no communication link/path between them. So, until they have enough neighbors to communicate, two nodes may continue their normal operation with their chosen temporary IP addresses.

### 5.4 Departure of Nodes

A node can depart from the network either gracefully or abruptly (i.e. due to mobility or sudden software crash or power failure). So, there is a chance of *"IP*

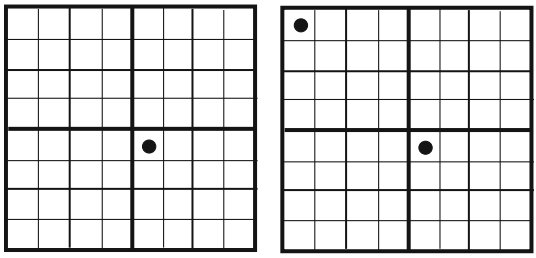


**Fig. 8** Node(s) having no neighbor

*address leakage"*. But in our approach it is resolved efficiently without requiring any extra cost. As part of entry update procedure, the UPDATE and HELLO messages are periodically sent by every node in the network. If no HELLO and UPDATE messages are received repeatedly after some predefined time interval from a node, all the neighbor nodes and the DDSs of that particular node remove the entries corresponding to that node. This IP address then can be automatically reused by another newly joined node.

### 5.5 Moving From One Square to Another Square

In this case, depending on a node's new position, a new set of DDSs can be chosen or old DDSs can be updated with its current location and speed for efficient geographic forwarding. There is no need to change the chosen IP address. If there is no communication link between the two places, network partitioning occurs. After a predefined time when there is no UPDATE and HELLO messages to update and validate the entry of its DDSs and neighbors respectively, the corresponding entry is automatically deleted and used to assign the IP address to another node requesting the same IP address.

### 5.6 Message Losses

Message losses are frequent in wireless networks. Because of network congestion, limited memory capacity of the nodes, corruption of message header, message can be discarded or lost. To accommodate message losses, messages are transmitted periodically.

### 5.7 Network Partitioning and Merging

Partitioning occurs when there is no communication link between two or several parts of the MANET. Each part then acts as a separate MANET. Partitioning does not affect in overall IP address assignment. But when merging of two or several MANETs occurs, it affects. Let us consider this scenario: Here a node joins the network by appearing within a grid, takes a temporary IP address, resolves the temporary IP address and finally takes the real IP address. After that it goes to another grid apart from the previous grid having no communication links or paths

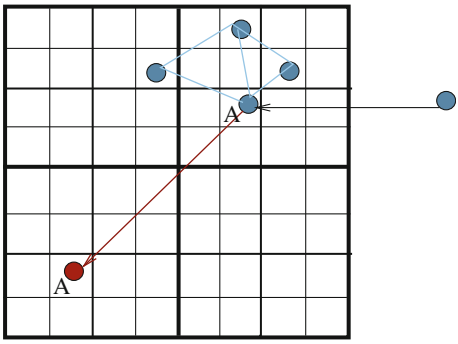


**Fig. 9** Node A moves from one square to another square having no communication link

between the grids (Fig. 9 shows partially this scenario). Now suppose another node joins the network in the previous grid and takes the same real IP address. As there is no communication path between these two grids, there is no way to know for the second node of the above scenario about this conflict in IP address. Now, when enough nodes join the MANET and there is at least one path among the nodes and hence merging of those separated networks occur. As a regular task, the DDSs for all nodes are updated or reselected according to the *DSP* with the help of UPDATE messages. When a DDS finds duplicate permanent IP addresses, it detects the merging and sends requests to those duplicate nodes to leave their IP addresses and retake a new IP address using the same autoconfiguration scheme described above. Under this circumstance, any packets destined to those old duplicate IP address holding nodes are discarded as the nodes are in a transition state of changing from old IP address to new IP address. Thus, with leaving old duplicates IP address and successfully retaking new conflict free IP address, the merging problem can be solved efficiently.

## 6 Simulation

### 6.1 Simulation Setup

Through simulation we evaluate the performance of our protocol both for static and mobile ad hoc networks. Between 100 and 600 nodes are randomly deployed in a fixed area of $1,360 \times 1,360\,\mathrm{m}^2$. The size of an Order-1 square is assumed to be $170 \times 170\,\mathrm{m}^2$. For mobile networks each node moves randomly at average velocities of $15, 20$ and $25\,\mathrm{ms}^{-1}$ without any pause. The transmission range and data rate of a node is 250 m and 2 MBps respectively. The joining time of all nodes within the network are randomly chosen from 0 to 30 s. Each simulation run ends when all nodes are assigned with real IP address. In a 32-bit IP address, the first 8-bit is unique in the network and the rest 24-bits are populated randomly (In Fig. 2 we assume first 24 bits of an IP address as the unique network ID just for simplicity). Under this restriction, first 2048 IP addresses are allocated for temporary IP address pool and the rest are used for real IP addresses.

### 6.2 Performance Metrics

We analyze the performance of our proposed protocol using the following performance metrics:

a) *Number of Conflicts*: When a node randomly chooses a real IP address, that address may conflict with an already allocated IP address to another node in the network. We define this situation as a *conflict* and count total number of such situations. Note that this conflict is ultimately resolved with the help of DDSs.
b) *Average latency*: *Latency* is the time interval between the moment when a node joins in the network and the moment when it acquires a non-conflicting real IP address. We sum up such latency for all nodes and find the average.
c) *Average DDT length*: We keep track of the number of entries in each node's *Duplicate-IP address Detection Table* (DDT). All nodes' DDT size are summed together to get an average length. For a scalable protocol, the size of DDT should grow very little with the increase in total number of nodes.
d) *Protocol Overhead*: It is defined as a ratio of total size of overhead packets in kilobytes to total number of nodes. For any scalable protocol, this number should be a bounded constant.
e) *Average Packet loss*: Any UPDATE/QUERY packets may get lost due to the limitation of geographic forwarding (i.e. loop-hole) and/or during *transient period* of a node. We count all those losses and take an average.
f) *Average REQUEST_QUEUE length*: This is defined as average number of entries in REQUEST_QUEUE of nodes.

### 6.3 Simulation Results

At first we show average number of conflicts in Fig. 10 for both static and mobile networks. On the average (roughly) only one conflicting situation occurs per node. The number of conflicts increases very slowly as the network size grows. Mobile networks have slightly more conflicting situations than static networks due to mobility.

Figure 11 demonstrates the average latency. On the average, a node needs 8–9.5 s to acquire a real IP address after joining the network. Also average latency increments very slowly as the number of nodes increases. Dynamic networks exhibits more latency than static networks. Moreover, the average latency of the proposed protocol is also examined under the circumstances of abrupt departure of some of the nodes in the network. 25% of existing nodes are chosen randomly to depart the network abruptly. Figure 11 also shows that the latency (for moving nodes of velocity 25 $ms^{-1}$) under this stipulated condition (i.e., abrupt departure of nodes) is slightly higher than those of the regular cases. This is due to the message losses and retransmission for the abruptly leaving nodes.

Average DDT length is shown in Fig. 12. From the figure it is clearly evident that on the average, a node keeps 7–9 entries. In other words, a node acts as DDS for roughly 7–9 other nodes in the network. Also the size does not grow too much with

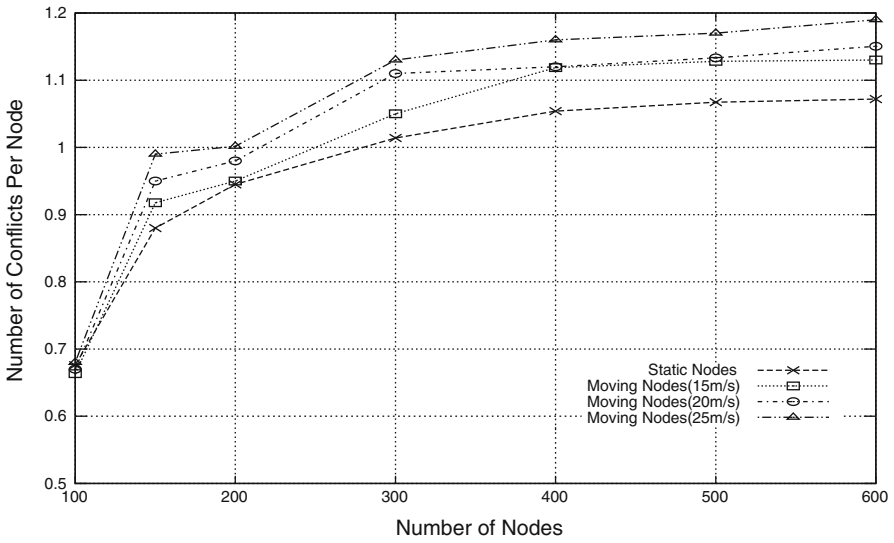


**Fig. 10** Average conflicts

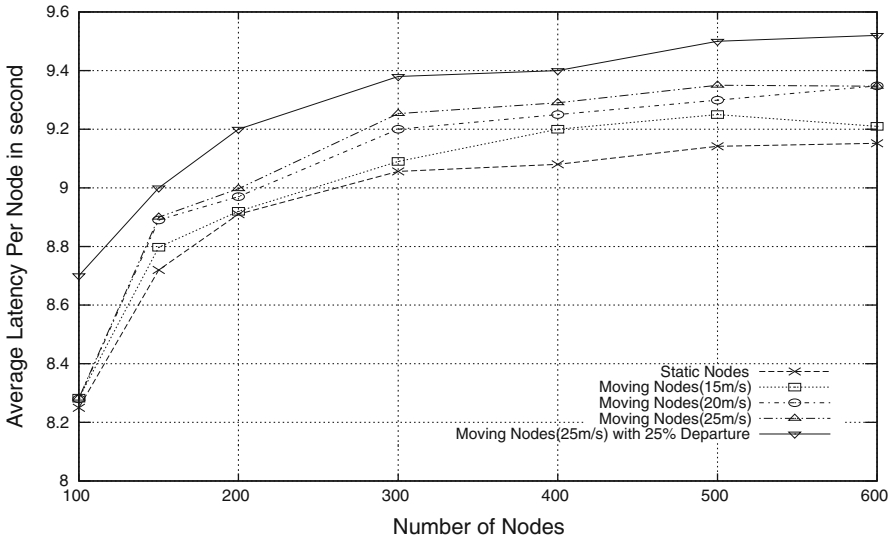


**Fig. 11** Configuration latency

the increase in network size. For static networks average DDT length remains almost constant at 8.5 and grows very slowly in dynamic networks. As the protocol is truly scalable such result is quite obvious.

Per node storage for REQUEST_QUEUE is shown in Fig. 13. The figure indicates highly scalable behavior in memory utilization. The storage requirement to maintain the protocol is almost same with the increased network size.

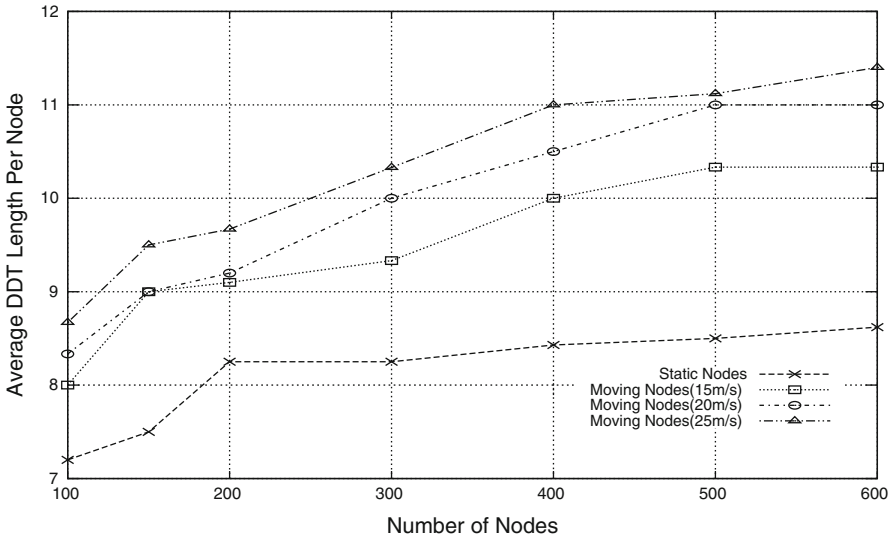


**Fig. 12** Dup-IP address detection table (*DDT*) length

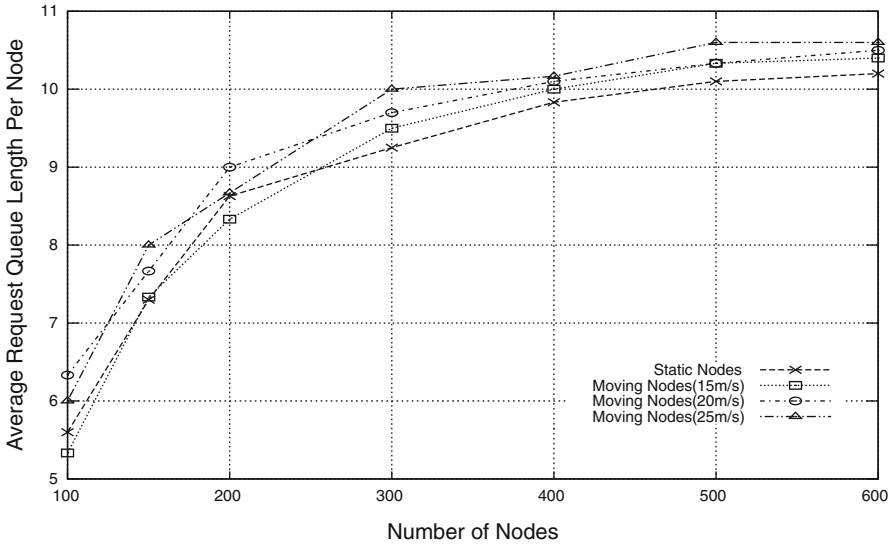


**Fig. 13** REQUEST_QUEUE length

Average packet loss per node occurs due to changing identity of nodes and failure of geographic forwarding because of loop hole. As shown in Fig. 14, average packet loss at the very beginning decrements rapidly as the network is becoming more dense than the previous one. For further growth of the network, packet losses per node remains pretty constant and have very little impact on the protocol since the network is now dense enough to prevent loop holes.

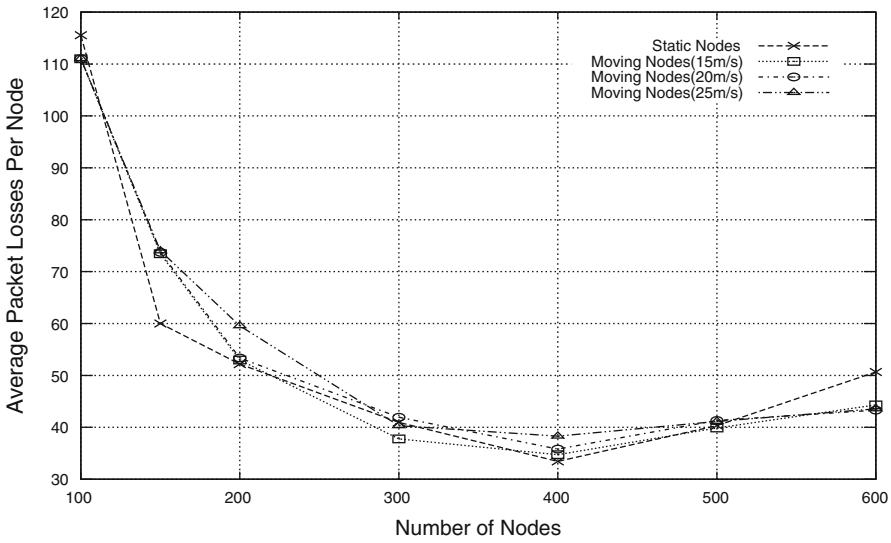


**Fig. 14** Packet loss

Figure 15 shows protocol overhead in KB per node. Protocol overhead grows very slowly with the network size. In particular, when the number of nodes vary from 300 to 600 nodes (i.e. a 100% increase), the protocol overhead increases only by 20%.

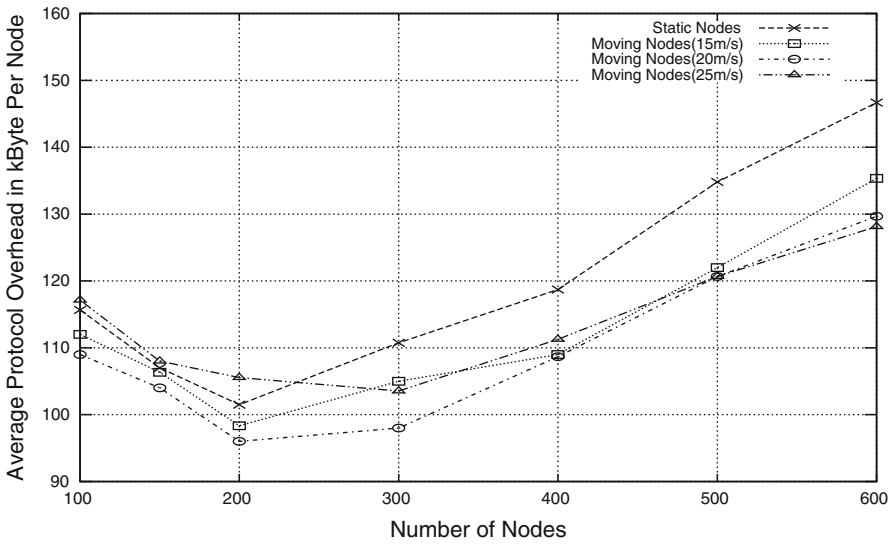


**Fig. 15** Protocol overhead

## 7 Analytical Comparison

We analyze our protocol theoretically and compare with the DAD [6] based algorithms (that allows flooding throughout the entire network) since DAD is the basis of most of the stateless autoconfiguration protocols. The analytical results show that our protocol (SAAMAN) outperforms the DAD based protocols.

In DAD based autoconfiguration protocols (as described earlier in Sect. 2), every new node joins with a temporary IP address and randomly chooses an IP address from permanent IP address pool. It then broadcasts a query message, containing that chosen IP address, to check whether the chosen IP address is already picked up by any other node in the network. Hence, every newly joined node has to flood this query message in the whole network at least once. If there is a conflict then the node chooses another random IP address and repeats the flooding. Therefore the number flooding varies with the number of times IP conflicts occurred. If a network (in this analysis, the network is considered to be static) consists of $n$ nodes, then a single broadcast requires $n$ forwarding if blind flooding is used. Therefore, $n$ nodes incur at least $n \times n = n^2$ forwarding. Hence, if every node on an average faces $p$ conflicts before assigning a unique IP address, then each node has to broadcast query messages $(p + 1)$ times on the average (i.e., no conflict is found in last trial after consecutive $p$ trials). Therefore, if every node has $p$ times conflict on an average and the total number of nodes in a network is $n$ (each of which runs the autoconfiguration protocol to join the network), then the total number of forwarding for all nodes, $F_{d_t}$, can be expressed as follows:

$$F_{d_t} = (p + 1) \times n^2$$

Hence, average number of forwarding per node, $F_{d_{avg}}$, is

$$F_{d_{avg}} = (p + 1) \times n \tag{1}$$

Equation (1) clearly shows that the protocol overhead increases in a great extent due to flooding for checking IP duplication as the number of nodes in the network increases. However, the number of forwarding in our proposed protocol (SAAMAN) depends on the network size. The detail analysis of the required number of flooding is shown below with the help of Fig. 2 where the network is of size Order-4.

In our protocol a newly joined node sends a single query message (through one-hop broadcast) in its own Order-1 square, 3 query messages to its 3 peer Order-1 square DDSs, 3 query messages to its 3 peer Order-2 square DDSs and 3 query messages to its 3 peer Order-3 square DDSs. Therefore, in total 9 query messages are sent to 9 different Order peer squares. In fact, each of these query messages is a single unicast message destined to respective DDS. For the 'worst case', we consider that 2, 4 and 8 forwarding (in our protocol forwarding means hop counts required for a unicast message to reach the desired destination) are required respectively to reach an Order-1 query message to its any one peer Order-1 square DDS, an Order-2 query message to its any one peer Order-2 square DDS and an Order-3 query message to its any one peer Order-3 square DDS. Hence, under above consideration, the total number of forwarding for a single (newly joined) node to

resolve IP conflicts (or to check IP duplication) in order to assign a unique IP address is: $1 + 3 \times 2 + 3 \times 4 + 3 \times 8 = 43$. In general, the total number of forwarding can be further broken down according to the following formulae:

$$\begin{aligned}
&\textit{Number of forwarding required to reach own Order-1 square}, F_s(0) = 1\\
&\textit{Number of forwarding required to reach a peer Order-1 square}, F_s(1) = 2 \times F_s(0)\\
&\textit{Number of forwarding required to reach a peer Order-2 square}, F_s(2) = 2 \times F_s(1)\\
&\textit{Number of forwarding required to reach a peer Order-3 square}, F_s(3) = 2 \times F_s(2)
\end{aligned}$$

Therefore, total number of forwarding, $F_s(4)$, that is required for a single node to resolve IP conflicts in a network of size Order-4 is:

$$\begin{aligned}
F_s(4) &= F_s(0) + 3 \times F_s(1) + 3 \times F_s(2) + 3 \times F_s(3)\\
&= F_s(0) + 3 \times 2 \times F_s(0) + 3 \times 2 \times F_s(1) + 3 \times 2 \times F_s(2)\\
&= F_s(0) + 3 \times 2 \times F_s(0) + 3 \times 2^2 \times F_s(0) + 3 \times 2^2 \times F_s(1)\\
&= F_s(0) + 3 \times 2 \times F_s(0) + 3 \times 2^2 \times F_s(0) + 3 \times 2^3 \times F_s(0)\\
&= F_s(0) \times (1 + 3 \times 2 + 3 \times 2^2 + 3 \times 2^3)\\
&= 1 \times (1 + 3 \times 2 \times (1 + 2 + 2^2))
\end{aligned}$$

Hence, for the network of size Order-$N$, the total number of forwarding, $F_s(N)$, needed for a single node is:

$$\begin{aligned}
F_s(N) &= 1 \times (1 + 3 \times 2 \times (1 + 2 + 2^2 + \cdots + 2^{N-2}))\\
&= 1 + 6 \times \frac{2^{N-2+1} - 1}{2 - 1}\\
&= 1 + 6 \times (2^{N-1} - 1)
\end{aligned}$$

If there are $n$ nodes in a network where each of them sends query messages to their respective DDSs and each node has $p$ conflicts on the average, then the total number of forwarding for all nodes, $F_{s_t}$, is:

$$\begin{aligned}
F_{s_t} &= (p + 1) \times n \times F_s(N)\\
&= (p + 1) \times n \times (1 + 6 \times (2^{N-1} - 1)) \quad [Here, N << n]
\end{aligned}$$

Hence, average number of forwarding per node, $F_{s_{avg}}$, is:

$$F_{s_{avg}} = (p + 1) \times (1 + 6 \times (2^{N-1} - 1)) \quad [Here, N << n] \tag{2}$$

Since $N << n$, Eqs. (1) and (2) conspicuously show that $F_{s_{avg}}$ is much smaller than $F_{d_{avg}}$. Therefore the protocol overhead of SAAMAN satisfactorily meager in comparison to that of full-flooding DAD based autoconfiguration protocols. Moreover, few scenarios comparing the performance are shown in Table 4.

Even our protocol carries out better results than other DAD based autoconfiguration protocols that use *Dominant Pruning (DP)* [24] or *Partial Dominant Pruning (PDP)* [25] as opposed to blind flooding for broadcasting DAD messages. To examine it we also conduct simulations and measure average number of forwarding required to broadcast packets throughout the entire network using DP

**Table 4** Average number of forwarding of query messages

**(a)** Static Environment, Network of size Order-3, Area: $680 \times 680m^2$, Area of an Order-1 square: $170 \times 170m^2$, Transmission Range: $250m$

| | $p = 0$ | | $p = 1$ | | $p = 2$ | | $p = 3$ | |
|---|---|---|---|---|---|---|---|---|
| Number of nodes | $F_{d_{avg}}$ | $F_{s_{avg}}$ | $F_{d_{avg}}$ | $F_{s_{avg}}$ | $F_{d_{avg}}$ | $F_{s_{avg}}$ | $F_{d_{avg}}$ | $F_{s_{avg}}$ |
| 100 | 100 | 19 | 200 | 38 | 300 | 57 | 400 | 76 |
| 200 | 200 | 19 | 400 | 38 | 600 | 57 | 800 | 76 |
| 300 | 300 | 19 | 600 | 38 | 900 | 57 | 1200 | 76 |

**(b)** Static Environment, Network of size Order-4, Area: $1360 \times 1360m^2$, Area of an Order-1 square: $170 \times 170m^2$, Transmission Range: $250m$

| | $p = 0$ | | $p = 1$ | | $p = 2$ | | $p = 3$ | |
|---|---|---|---|---|---|---|---|---|
| Number of nodes | $F_{d_{avg}}$ | $F_{s_{avg}}$ | $F_{d_{avg}}$ | $F_{s_{avg}}$ | $F_{d_{avg}}$ | $F_{s_{avg}}$ | $F_{d_{avg}}$ | $F_{s_{avg}}$ |
| 100 | 100 | 43 | 200 | 86 | 300 | 129 | 400 | 172 |
| 200 | 200 | 43 | 400 | 86 | 600 | 129 | 800 | 172 |
| 300 | 300 | 43 | 600 | 86 | 900 | 129 | 1200 | 172 |
| 400 | 400 | 43 | 800 | 86 | 1200 | 129 | 1600 | 172 |
| 500 | 500 | 43 | 1000 | 86 | 1500 | 129 | 2000 | 172 |
| 600 | 600 | 43 | 1200 | 86 | 1800 | 129 | 2400 | 172 |

**Table 5** Average number of forwarding of query messages for DP and PDP based DAD

| | $p = 0$ | | $p = 1$ | | $p = 2$ | | $p = 3$ | |
|---|---|---|---|---|---|---|---|---|
| Number of nodes | DP | PDP | DP | PDP | DP | PDP | DP | PDP |
| 100 | 70 | 36 | 140 | 72 | 210 | 108 | 280 | 144 |
| 200 | 148 | 78 | 296 | 156 | 444 | 234 | 592 | 312 |
| 300 | 207 | 108 | 414 | 216 | 621 | 324 | 828 | 432 |

Simulation setup: Static environment, area: 680 × 680 $m^2$, transmission range: 250 m. Each node in the network generates a broadcast packet in turn. The number of forwarding required to broadcast is measured for each node and then averaged over all nodes. All data points are the mean of those averages processed over 5 simulation runs

and PDP. The results are shown in Table 5 which demonstrates average number of forwarding required by a single node to broadcast in the entire network.

Tables 4 and 5 manifest that the average number of forwarding required by our proposed protocol SAAMAN is much less than that of DP and PDP incorporated DAD based autoconfiguration protocols. Hence, our protocol incurs less protocol overheads, network congestions and message losses.

## 8 Future Works

In the proposed protocol, SAAMAN, less attention has been given to security problems. Security problem and its solutions are an emerging area in *Address Autoconfiguration in Mobile Ad Hoc Networks*. Various types of security problems can arise in MANETs while performing autoconfiguration. These are mainly due to

different behavior of nodes. Behaviorally these unconventional nodes in MANETs can be divided into three categories. They are *Malicious Nodes, Selfish Nodes* and *Helpless Nodes*.

Malicious nodes are those nodes which can refrain a requesting node from obtaining an IP address by sending frequent NACK messages. It can also capture a block the IP addresses by periodically updating the DDSs with fake UPDATE messages with different unused IP addresses as if those IP addresses are currently being used.

A selfish node can deny to route the packet by simply ignoring it. As a result, frequent message losses and network partitioning may happen. There is no first-hand solution to this problem. Periodically monitoring the behavior of the neighbors may solve this problem.

Helpless nodes are those nodes that have less battery power to transmit or forward packets to its neighbor. As their power signal is low, message losses and corruption of messages frequently occur in the network. This problem can be solved by inserting a new attribute in the *Neighbor Allocation Table*, 1 i.e., in NAT representing the current strength of the power signal of the neighbors of a node. A node can periodically broadcast HELLO messages containing the current power level. By checking the NAT, a node can heuristically transmit or forward messages to the best node.

A detail analysis of all possible misbehavior of unconventional nodes in untrusted environment and their concrete solutions while assigning conflict-free unique IP address is the primary concern of our future work.

## 9 Conclusion

This paper presents a new, efficient and scalable autoconfiguration protocol for MANET where topology can be changed frequently. Distributed Duplicate-IP Detection Servers (DDSs) are used to ensure the uniqueness of chosen IP addresses. DAD algorithm is run for only one hop to acquire unique temporary IP address. Grid based quad tree hierarchy is used to distribute DDSs evenly across the MANET. So, there is no leader election and allocation of IP address state information.

The special characteristics of the proposed protocol are summarized as follows:

1. The protocol handles problems of turning on more than one node concurrently during bootstrapping which is described in Sect. 4.1.
2. The protocol is highly scalable. The causes of its scalability are:
    a. No node is a single point of failure or bottleneck–the workload related to address assignment and duplicate IP address detection service is distributed evenly over all the nodes in the network.
    b. The storage and communication cost of address assignment scheme as well as the DDSs size grow as a small valued function of the total number of nodes. The correctness of this characteristic also has been verified in Sect. 6.3.

3. The problem of two or more nodes choosing conflicting IP address is also intelligently resolved as described in Sect. 4.
4. The proposed protocol is consistent with frequent topology changes.

Last but not the least, the proposed autoconfiguration protocol may be extended in future to cope with un-trusted ad hoc environment, since the present solution is vulnerable to the attack like *Denial of Services* (DoS) [26] and others by malicious nodes.

**Acknowledgments** The authors would like to thank the anonymous reviewers and the editor for their helpful suggestions and recommendations.

## References

1. Hussain, S.R., Saha, S., Rahman, A.: An efficient and scalable address autoconfiguration in mobile ad hoc networks. In: Proceedings of the 8th International Conference on Ad-Hoc, Mobile and Wireless Networks (ADHOC-NOW'09), Spain, LNCS 5793, pp. 152–165. Springer (2009)
2. Droms, R.: Dynamic Host Configuration Protocol. http://www.ietf.org/rfc/rfc2131.txt (1997)
3. Narten, T., Nordmark, E., Simpson, W.: Neighbor discovery for IP version 6 (IPv6). In: Network Working Group RFC 2461 (1999)
4. Thomson, S., Narten, T.: IPv6 stateless address autoconfiguration, December 1998. IETF RFC 2462—Standards Track (1998)
5. Guttman, E., Cheshire, S.: Zero Configuration Networking Group. http://www.ietf.org/html.charters/zeroconf-charter.html. Cited 21 (2003)
6. Perkins, C.E., Malinen, J.T., Wakikawa, R., Royer, E.M., Sun, Y.: IP address autoconfiguration for ad hoc networks. Draft-ietf-manet-autoconf-01.txt (2001)
7. Nesargi, S., Prakash, R.: MANETConf: configuration of a host in mobile ad hoc network. In: Proceedings of IEEE INFOCOM, New York, USA (2002)
8. Mohsin, M., Prakash, R.: IP address assignment in a mobile ad hoc network. In: Proceedings of IEEE MILCOM, Anaheim, USA (2002)
9. Weniger, K., Zitterbart, M.: IPv6 autoconfiguration in large-scale mobile ad-hoc networks. In: Proceedings of European Wireless, Italy (2002)
10. Zhou, H., Ni, L.M., Mutka, M.W.: Prophet address allocation for large scale MANETs. In: Proceedings of IEEE INFOCOM, USA (2003)
11. Vaidya, N.H.: Weak duplicate address detection in mobile ad hoc networks. In: Proceedings of the Third ACM International Symposium on Mobile Ad Hoc Networking and Computing, pp. 206–216. ACM Press (2002)
12. Park, J., Kim, Y., Park, S.: Stateless address autoconfiguration in mobile ad hoc networks using site-local address, 2001, internet draft: draft-park-zeroconf-manetipv6-00.txt (2001)
13. Tayal, A., Patnaik, L.: An address assignment for the automatic configuration of mobile ad hoc networks. Pers. Ubiquitous Comput **8**(1), 47–54 (2004) (Springer)
14. Kim, H., Kim, S. C., Yu, M., Song, J. K., Mah, P.: DAP: dynamic address assignment protocol in mobile ad-hoc networks. In: Proceedings of IEEE International Symposium on Consumer Electronics (ISCE'07), pp. 1–6 (2007)
15. Thoppian, M., Prakash, R.: A distributed protocol for dynamic address assignment in mobile ad hoc netorks. IEEE Trans. Mobile Comput. **5**(7), 4–19 (2006)
16. Sheu, J.P., Tu, S.H., Chan, L.H.: A distributed IP address assignment scheme for ad hoc networks. In: Proceedings of the 11th International Conference on Parallel and Distributed Systems (ICPADS'05), vol 1, pp. 439–445 (2005)
17. Xu, T., Wu, J.: Quorum based IP address autoconfiguration in mobile ad hoc networks. In: Proceedings of International Conference on Distributed Computing Systems Workshops (ICDCSW'07) (2007)

18. Ancillotti, E., Bruno, R., Conti, M., Pinizzotto, A.: Dynamic address autoconfiguration in hybrid ad hoc networks, vol. 5, pp. 300–317. Pervasive and Mobile Computing, Elsevier. doi:10.1016/j.pmcj.2008.09.008) (2009)
19. Fazio, M., Villari, M., Puliafito, A.: IP address autoconfiguration in ad hoc networks: design, implementation and measurements. Comput. Netw. **50**(7), 898–920 (2006) (Elsevier)
20. Weniger K.: PACMAN: passive autoconfiguration for mobile ad hoc networks. IEEE J. Selected Areas Commun. (JSAC) **23**(3), 507–519 (2005)
21. Sun, Y., Royer, E.M.: Dynamic address configuration in mobile ad hoc networks. In: Proceedings of Wireless Communications & Mobile Computing, pp. 315–329 (2004)
22. Fernandes, N.C., Moreira, M.D.D., Duarte, O.C.M.B.: An efficient filter-based addressing protocol for autoconfiguration of mobile ad hoc networks. In: Proceedings of IEEE INFOCOM. Rio de Janeiro, Brasil
23. Li, J., Jannotti, J., Cuoto, D.D., Karger, D., Morris, R.: A scalable location service for geographic ad hoc routing. In: Proceedings of the ACM/IEEE International Conference of Mobile Computing and Networking (MOBICOM'00), pp. 120–130 (2000)
24. Lim, H., Kim, C.: Multicast tree construction and flooding in wireless ad hoc networks. In: Proceedings of ACM MSWiM, USA, pp. 61–68 (2000)
25. Lou, W., Wu, J.: On reducing broadcast redundancy in ad hoc wireless networks. IEEE Trans. Mobile Comput. 1(2):111–123 (2002)
26. Zhang, Y., Lee, W.: Intrusion detection in wireless ad hoc networks. In: Proceedings of the ACM/IEEE International Conference of Mobile Computing and Networking (MOBICOM'00), pp. 275–283. Boston, MA (2000).

## Author Biographies

**Syed Rafiul Hussain** received his B. Sc. in Computer Science and Engineering from the department of Computer Science and Engineering (CSE), Bangladesh University of Engineering and Technology (BUET), Dhaka, Bangladesh in 2009. Currently he is working as a full time faculty member in the Department of CSE, Ahsanullah University of Science and Technology (AUST) and as a visiting faculty member in the Department of CSE, BUET. His research interests include mobile ad-hoc and sensor networks, peer-to-peer computing, distributed systems, network security and data mining.

**Subrata Saha** received his B. Sc. in Computer Science and Engineering from the department of Computer Science and Engineering (CSE), Bangladesh University of Engineering and Technology (BUET), Dhaka, Bangladesh in 2009. He is working as a Software Engineer at Tiger IT Bangladesh Limited. His research interests include mobile ad-hoc and sensor networks, cloud computing, computer systems and software engineering.

**Ashikur Rahman** received his B. Sc. and M. Sc. degrees in Computer Science and Engineering from the department of Computer Science and Engineering (CSE), Bangladesh University of Engineering and Technology (BUET), Dhaka, Bangladesh in 1998 and 2001, respectively. He received his PhD in 2006 from the Department of Computing Science, University of Alberta, Canada. In the year of 2007, after finishing his PhD, he worked as a postdoctoral researcher at the Simon Fraser University, Canada. From the year of 2008 to 2010, he worked as a faculty member in the Department of CSE, BUET. Currently he is working as a postdoctoral researcher at the department of Computer Science, University of Calgary, Canada. His research interests include ad-hoc and sensor networks, social networks, peer-to-peer computing, swarm intelligence, back-end compiler optimization and neural networks.