A Distributed Protocol for Dynamic Address Assignment in Mobile Ad Hoc Networks [1]

Mansi Thoppian and Ravi Prakash

Abstract

A Mobile Ad hoc NETwork (MANET) is a group of mobile nodes that form a multi-hop wireless network. The topology of the network can change randomly due to unpredictable mobility of nodes and propagation characteristics. Previously, it was assumed that the nodes in the network were assigned IP addresses *a priori*. This may not be feasible as nodes can enter and leave the network dynamically. A dynamic IP address assignment protocol like DHCP requires centralized servers that may not be present in MANETs. Hence, we propose a distributed protocol for dynamic IP address assignment to nodes in MANETs. The proposed solution guarantees unique IP address assignment under a variety of network conditions including message losses, network partitioning and merging. Simulation results show that the protocol incurs low latency and communication overhead for an IP address assignment.

Index Terms

MANET, address allocation, IP-networks.

I. Introduction

MOBILE ad hoc networks (MANETs) are multi-hop wireless networks of mobile nodes without any fixed, or pre-existing infrastructure. Nodes within the wireless range of each other can communicate directly. Nodes outside each other's wireless range must communicate indirectly, using a multi-hop route through other nodes in the network. This multi-hop route may change if the network topology changes. Several routing protocols like DSR [5], CBR [6], TORA [7], ZRP [8], DSDV [9], AODV [10], etc. have been proposed for MANETs.

This work is supported in part by NSF CAREER Grant # CCR-9796331

The authors are with the Department of Computer Science at The University of Texas at Dallas, Richardson, TX 75083. Email: mansi, ravip@utdallas.edu

MANETs may operate in a stand-alone mode, or may have gateways to interconnect to a fixed network. In the stand-alone mode, the network is spontaneously formed by nodes gathering at a remote location with no network infrastructure. Such a network can also be formed when the gateways to the external world fail or when all the existing network infrastructure goes down due to natural/man-made disasters. In the presence of a gateway, a MANET is envisioned to operate as a stub network connected to a fixed internetwork.

In most networks, including MANETs, each node needs a unique identifier to communicate. It may be argued that the MAC address or the home IP address of the node should be sufficient for this purpose. However, use of the MAC address as a unique identifier has the following limitations:

1) MANET nodes are not restricted to using network interface cards (NICs) with a 48-bit IEEE-assigned unique MAC address. In fact, the TCP/IP protocol stack should work on a variety of data-link layer implementations. So, if this approach were to be employed, specific implementations would be required for each type of hardware.
2) The uniqueness of a MAC address cannot always be guaranteed, as it is possible to change the MAC address using commands like *ifconfig*.
3) There are known instances of multiple NIC cards from the same vendor having the same MAC address [11] [12].

The home IP address of the mobile node may not be usable as a unique identifier at all times. The home IP address may not be permanent, for example when the node acquires an IP address during boot up through DHCP and releases it when it leaves the network. It is possible that two nodes belonging to the same home network, at different times, may join the MANET with the same home IP address. Moreover, even if a node owns a unique home IP address, it needs a unique care-of IP address in the MANET if it is to be addressable from the Internet.

Static IP address assignment for MANET nodes is difficult as it needs to be done manually with prior knowledge about the MANET's current network configuration. Dynamic configuration protocols like Dynamic Host Configuration Protocol (DHCP) [1] require the presence of centralized servers. MANETs may not have such dedicated servers. Hence, centralized protocols cannot be used to configure nodes in MANETs. In this paper we present a distributed protocol for dynamic IP address assignment. The proposed solution is targeted towards the stand-alone mode

of operation. It may also be used in situations where the gateway only provides connectivity to external network(s) with no support for IP address assignment. While we limit our examples to IPv4, the proposed protocol is applicable to both IPv4 and IPv6 networks.

The remainder of the paper is organized as follows. Section II presents the related work. Section III provides the basic idea of the protocol. The protocol messages and timers are discussed in Sections IV and V, respectively. In Section VI, a detailed description of the protocol is provided followed by a discussion of its correctness in Section VII. Simulation experiments and results are presented in Section VIII, followed by conclusion in Section IX.

II. Related Work

Cheshire *et al.* [13] describe a method for auto-configuration of the host by randomly choosing a link-local address within the range 169.254.1.0 to 169.254.254.255. After selecting an address, the host tests to see if the address is already in use 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 be within the communication range of every other node in the network, which is not always possible in the case of MANETs. To extend the solution to MANETs, conflict detection message(s) will have to be flooded throughout the network.

Perkins *et al.* [14] propose a solution for address auto-configuration in ad hoc networks. An address is randomly chosen within the range 2048 to 65534 from the 169.254/16 address block. A node *floods* Route Requests (RREQs) for the selected IP address. If no Route Reply (RREP) is received within a timeout period, the node retries for RREQ_RETRIES times. At the end of all the retries, if no response is received, the chosen IP address is assumed to be free. The node assigns itself that IP address. Here the latency is the timeout value multiplied by RREQ_RETRIES. This approach requires the routing protocol to have a "route discovery" phase. It does not address the network partitioning issues.

IPv6 Stateless Autoconfiguration [15] specifies the steps a node takes to configure its interfaces in IPv6. The steps include construction of link-local address, *Duplicate Address Detection*, and construction of a site-local address. During *Duplicate Address Detection* for MANETs flooding is required, thus making the approach unscalable. To overcome this scalability issue, an extension is proposed in [16] by building a hierarchical structure. But the cost incurred in maintaining such a hierarchical structure may be high.

In MANETconf [12], a new node entering the MANET requests for configuration information from its neighbors. One of these neighbors initiates the IP address allocation process for the new node. For each IP address assignment, this approach requires a network-wide broadcast leading to scalability problems. However, this approach handles network partitioning and subsequent merging.

Weak Duplicate Address Detection (DAD) protocol [17] requires each node in the network to have a unique key. Weak DAD requires that packets "meant for" one node must not be routed to another node, even if the two nodes have chosen the same address. This is achieved by using the key information for duplicate address detection. In this approach, the routing protocol related control packets need to be modified to carry the key information. In the weak DAD scheme, the packet can still be misrouted in the interval between the occurrence of duplicate IP addresses in the network and their actual detection. Enhanced Weak DAD [17] was proposed to eliminate the above shortcoming by using sequence numbers and some bookkeeping.

Passive Duplicate Address Detection (PDAD) presented in [23] tries to detect duplicate addresses without disseminating additional control information. Based on classic link state routing, the following three schemes are proposed:

- PDAD based on sequence numbers (PDAD-SN): Detects duplicate addresses by using sequence numbers and link state information.
- PDAD based on locality principle (PDAD-LP): Exploits the fact that nodes move with limited speed.
- PDAD based on neighborhood (PDAD-NH): Exploits the property that a node knows its own neighborhood and the neighborhood of the originator of a link state packet.

The proposed PDAD schemes use random source IDs to detect duplicate addresses within two hop neighborhood. This approach requires the use of a link state routing protocol.

The stateless address autoconfiguration scheme presented in [24] consists of three phases: (1) selection of random address, (2) verification of the uniqueness of the address and, (3) assignment of the address to the network interface. The verification of uniqueness is done by a hybrid DAD scheme consisting of two phases: (a) strong DAD phase and, (b) weak DAD phase. Within a connected ad hoc network, a node configures itself with an IP address using strong DAD. During strong DAD, the node chooses a tentative address and checks for any address duplication by

broadcasting AREQ message with the chosen tentative address for a fixed number of times. If no response is received for the AREQ message, the node configures itself with the tentative address. Weak DAD uses a “key” in addition to the IP address to detect duplicate addresses during ad hoc routing. This scheme ensures that during resolution of an address conflict, the sessions using conflicting addresses are maintained until the sessions are closed.

Dynamic Registration and Configuration Protocol (DRCP) [18] [19] extends DHCP for wireless networks. In this protocol, each node acts as both server and client and owns an address pool. The address pool distribution is done using Dynamic Address Allocation Protocol (DAAP) [18] [19]. Each node obtains the address pool by requesting half of the addresses from the address pool of a neighboring node. The protocol does not discuss the network partitioning issues and the impact of message loss.

Dynamic Address Allocation Protocol proposed in [20] requires that new nodes approach the leader of the network to obtain an IP address. The leader is the node with the highest IP address in the network. Network partitioning and merger are considered. The unique identifier used for identifying the network is the MAC address of the initiator which is the first node that comes up in the network. This could lead to a problem of multiple networks having the same identifier when the initiator itself moves out of the network and forms another network. The impact of message losses is not considered.

Zhou *et al.* [21] propose a solution which is derived from a sequence generation scheme whereby a sequence consisting of numbers within a range R is generated using a function *f(n)*. The function *f(n)* is chosen such that in the sequences generated by *f(n)*, the interval between two occurrences of the same number is very large and the probability of more than one occurrence of the same number in a limited number of different sequences initiated by different seeds during some interval is extremely low. The protocol requires the first node to choose a random number from the range R as its IP address and use a random state value as the seed for function *f(n)*. When a new node joins the network, the configured node generates another integer and a new state value using *f(n)*. The new node obtains these values and configures itself. In this approach, the needed block of IP addresses may be significantly bigger than the number of nodes in the network. Even if the minimal interval between two occurrences of the same address in the sequence generated is extremely large, it is still possible for two nodes to have the same IP

address if the nodes keep moving in and out of the network at a high rate.

The solutions mentioned above have made significant contributions to our understanding of the problem. However, we believe that all of these approaches handle only a subset of the network conditions listed below:

1) Topology changes: Nodes in the network can move arbitrarily and can enter and leave the network dynamically.
2) Message losses and node crashes: Message loss can be quite prevalent and can lead to duplicate IP address assignment if not handled effectively. Nodes can leave the network abruptly either due to link failure or crash.
3) Partitioning and subsequent merging: During the course of MANET operation, the network can split into multiple networks and subsequently merge into one. During network merging, it is possible to have duplicate IP addresses in the merged network.
4) Concurrent address requests: Multiple nodes can join the network simultaneously.
5) Limited energy and bandwidth: Nodes in MANET are energy limited and links are bandwidth limited. Hence, the communication overhead incurred should be low.

In this paper, we propose a solution similar to DAAP [18], [19] that guarantees unique IP address assignment under the above network conditions. In our approach, most of the address assignments require local communication leading to low communication overhead and latency (see Section VIII for details).

III. Basic Idea

The objective of the proposed protocol is to assign a unique IP address to a new node joining the MANET. The new node joining the network is called a *requester*. The configured node responsible for assigning an IP address to the *requester* is called an *allocator*. We assume that the MANET starts with a single node. We call this node the *Initiator* of the network and the configuration of this first node as MANET *initialization*. For simplicity, it is assumed that at least the first node in the MANET knows the IP address block from which the IP addresses are to be assigned to the nodes in the MANET. For example, this block can be the IPv4 private address block: 10.0.0.1 - 10.255.255.255, 172.16.0.0 - 172.31.255.255 or 192.168.0.0 - 192.168.255.255. The proposed protocol is equally applicable for IPv6 address space. The address

block information can be propagated to other nodes joining the network during the assignment process. The term "broadcast" in this paper stands for local broadcast, unless specified otherwise.

When the *Initiator* starts operation in MANET mode, it broadcasts a message requesting an IP address. As there are no other MANET nodes in the neighborhood, the *Initiator* will not receive any response. The *Initiator* re-broadcasts its request message for a constant number of times after which it assigns itself the first IP address from the IP address block and forms its *free_ip* set from the remaining addresses. The *free_ip* set is an ordered set containing addresses that are not in use by any node in the network.

After MANET initialization, every time a new node (*requester*) requests an IP address, one of the existing MANET nodes (*allocator*) within communication range of the *requester* initiates address allocation process for the *requester*. If the *allocator* has a non-empty *free_ip* set, it allots the second half of the addresses from its *free_ip* set to the *requester* (this approach is similar to the buddy system for memory management [2], [3]). Otherwise, it performs an expanding ring search whereby it propagates the request through the network. If the *allocator* finds a node (say node A) with a non-empty *free_ip* set during the expanding ring search, it allots half of the addresses from node A's *free_ip* set to the *requester*. The *requester* configures itself with the first address from the alloted address block and forms its *free_ip* set with the remaining addresses in the block.

During the expanding ring search, the *allocator* might fail to find a free IP address either because: (i) all IP addresses have been assigned to nodes currently in the MANET or (ii) some nodes have left the MANET without releasing their IP address and/or *free_ip* set (*leaked-addresses*). In the first scenario, no new node can be admitted (without expanding the address block range) as the MANET has reached its maximum size. In the second case, the *leaked-addresses* have to be reclaimed and assigned to new nodes joining the network. Address reclamation is done as follows: the *allocator* determines the addresses of the nodes that failed to respond during the expanding ring search (call this set the *missing_addresses* set). The *allocator* performs a network-wide broadcast targeted towards these addresses. The nodes in the network, on hearing this broadcast message, respond with a message indicating a conflict if their IP address is among the broadcast addresses. If the *allocator* receives messages indicating conflict, it removes the corresponding IP address from the *missing_addresses* set. The addresses in the resultant *missing_addresses* set

are then declared as free IP addresses. One of these addresses is assigned to the *requester*. The rest of the free IP addresses are re-distributed among the existing nodes in the network to form their *free_ip* sets. Concurrent reclamation operations are serialized based on the priorities of the *allocators* (see Section VI-E for details).

During the course of MANET operation, nodes can split from the network and form/join different networks. These networks can later merge into a single network. Each network is assigned a unique identifier (*network_id*). To detect network-merging, nodes periodically broadcast "Hello" messages to their neighbors. These "Hello" messages contain the IP address and the *network_id* of the sending node. Whenever a node (say A) receives a "Hello" message from its neighboring node (say B) containing a different *network_id* than its own, node A detects merging of networks and replies. When node B receives a reply from A, it detects network-merging as well. Furthermore, if node A has higher IP address than node B (ties are broken using *network_ids*), then node B is responsible for re-configuring the merged network in terms of re-distributing free IP addresses and removing duplicate address assignments. We call node B the *merge_agent*. Node B initiates a network-wide flood to collect the network configuration information, i.e., the IP addresses assigned to the nodes and their corresponding *free_ip* sets within its network. Node B asks node A to initiate similar network-wide flood in node A's network[1]. We call node A as *co-merge_agent*. Node A sends the collected network-wide information to node B. After collecting the IP address and *free_ip* set information from all the responses, node B detects duplicate IP address assignments among nodes in the merged network. Node B invalidates the IP address of the nodes with conflicting IP addresses and assigns them with new IP address. After invalidation, node B performs re-distribution of free addresses among nodes in the merged network.

The protocol is described in detail in Section VI.

IV. Protocol Messages

The following messages are exchanged during IP address assignment:

- *IPAddressRequest*: A local broadcast message from the *requester* to neighboring nodes in the network requesting an IP address.

[1]By A's and B's network we mean the networks they belonged to prior to merging.

- *IPAddressAvail*: Unicast[2] responses to *IPAddressRequest* from the neighboring nodes of the *requester*. This message contains the replying node's IP address and proposed block of IP addresses, if any.
- *AllocatorChosen*: Unicast messages from the *requester* to all its neighbors on electing an *allocator* from among the neighbors that sent *IPAddressAvail* messages. This message contains the IP address of the chosen *allocator*.
- *IPAddressAssign*: Unicast message from the *allocator* to the *requester* assigning an IP address. This message contains an IP address block for the *requester*, IP address range from which the addresses can be assigned to nodes, and the *network_id*.
- *IPAddressUpdate*: Network-wide message exchanged between nodes to update the network configuration information. This message is sent to all nodes in the network: (i) by the *allocator* responsible for reclamation of IP addresses during IP address allocation process, and (ii) by the *merge_agent* during network-merging process.
- *WaitPeriod*: Unicast message from the *allocator* informing the *requester* to extend its *assign_ip* timer (explained in Section V).

The following four messages are exchanged during the expanding ring search:

- *IPAddressInfoRequest*: Unicast messages sent from the *allocator*, during expanding ring search, to nodes in the network requesting network configuration information namely the IP address and *free_ip* set.
- *IPAddressInfoReply*: Unicast message in response to *IPAddressInfoRequest* from a node to the *allocator* containing its IP address, *free_ip* set and the neighbor list information.
- *IPAddressChosen*: Unicast message sent by the *allocator* to a node in the network informing the node that addresses from its *free_ip* set are chosen for assignment.
- *IPAddressConfirm*: Unicast response to *IPAddressChosen* message confirming the receipt of *IPAddressChosen* message. This message contains the IP address block for the *requester*.

The following five messages are exchanged during the reclamation operation :

- *TentativeFreeAddresses*: Network-wide flood message sent by the *allocator* during the reclamation of IP addresses. This message contains the *missing_addresses* set.

[2]Unicast messages directed to *requester* or originating from *requester* use MAC address for communication.

- *ConflictNotification*: Unicast message sent by a node in response to a *TentativeFreeAddresses* message indicating that the node's IP address is in conflict with an address in the *TentativeFreeAddresses* message.
- *NoConflict*: Unicast message sent by a node in response to a *TentativeFreeAddresses* message indicating that the node's IP address is not in conflict with the addresses in the *TentativeFreeAddresses* message.
- *Defer*: Unicast messages sent by the nodes/*co-merge_agents* on receiving multiple *TentativeFreeAddresses*/*PartitionMergeQuery* messages from different *allocators*/ *merge_agents*. This is sent to all the *allocators*/*merge_agents* except the *allocator*/*merge_agent* with the lowest IP address. On receipt of this message, the receiver (*allocator*/*merge_agent*) suspends the address allocation/network-merge process.
- *Resume*: Unicast message sent by a node that had previously sent a *Defer* message. A node on receiving this message resumes the suspended address allocation/network-merge process.

The following message is exchanged during migration of a *requester*:

- *IPAddressForward*: When a *requester* moves away from its *allocator* before acquiring an IP address, the *requester* chooses a new *allocator*. The new *allocator* sends this message to the old *allocator*.

The following four messages are used during partition handling:

- *Hello*: Local broadcast messages exchanged periodically between neighbors to detect neighborhood changes and merging of networks. Each node sends this message containing its *network_id* to its neighbors.
- *PartitionMergeQuery*: Network-wide flood message sent by the *merge_agent*/*co-merge_agent* to collect the network-wide IP address information. This message contains the *network_ids* of the merging networks and the IP address of *merge_agent* .
- *PartitionMergeResponse*: Unicast message sent in response to the *PartitionMergeQuery* message by a node to a *merge_agent*/*co-merge_agent*. This message contains the responding node's IP address, *free_ip* set and *pending_ip* set (explained in SectionVI).
- *IPAddressInvalidate*: Network-wide broadcast message sent from the *merge_agent* to invalidate conflicting IP addresses.

The following message is for graceful departure of nodes:

- *HandOver*: A node departing the network sends this unicast message, containing its IP address, *free_ip* set and *pending_ip* set, to one of its neighbors.

V. TIMERS

Timers are used to ensure that the protocol is deadlock-free and works correctly in the event of message losses or node crashes. Let *t* indicate the time it takes for a message to reach a node one hop away from the sender, and let *p* indicate the upper bound on the time required to process a message (including the queuing delay) at a node. The following are the various timers[3] used in the protocol.

- *offer_ip* timer: The *requester* sets this timer after sending *IPAddressRequest* message. If the *requester* does not receive any reply before the timer expires, it retries for *requester_retry* times, where *requester_retry* is the maximum number of attempts by a *requester* to acquire an IP address. The timer value should at least be (2t + p).
- *allocation_pending* timer: A node in the network sets this timer after sending an *IPAddressAvail* message and creating an entry in the *pending_ip* set. If no *AllocatorChosen* message is received before the timer expires, the proposed address block is withdrawn from the *pending_ip* set and appended to the *free_ip* set of the node. The timer value should at least be (2t + p).
- *assign_ip* timer: This timer is set by the *requester* after choosing an *allocator*. If no address is assigned before the timer expires, the *requester* retries *assign_retry* times, where *assign_retry* is the maximum number of times a *requester* tries to choose a new *allocator*. The timer value should at least be (d + 1)×(2t + p), where the value of the hop count, *d*, is provided by the *allocator* through a *WaitPeriod* message (refer to Section VI for details on hop count).
- *confirm_ip* timer: The *allocator* sets this timer after sending an *IPAddressChosen* message to a node with largest *free_ip* set. If the timer expires before the receipt of an *IPAddressConfirm* message, the *allocator* chooses the next node with largest *free_ip* set. The timer value should at least be (d×(2t + p)).
- *state_collection* timer: The *allocator* sets this timer before sending an *IPAddressInfoRequest*/*TentativeFreeAddresses* message. The timer should at least be (d×(2t + p)).

[3]Here we have provided the minimum values for the timers. It is possible to choose higher values for the timers to make the protocol less aggressive while increasing its latency, without compromising its correctness.

- *hello* timer: This timer is used to detect network-merging. It is set after sending the *Hello* messages. The timer value corresponds to the interval between consecutive *Hello* messages.
- *partition* timer: This timer is set by the *merge_agent*. It is set after sending the network-wide flood of *PartitionMergeQuery* message. If D_1 and D_2 are the average diameters of the two networks, the timer value should be at least (2 (max(D_1, D_2)(t + p)))).

VI. Detailed Description of the Algorithm

In MANETs, the following scenarios could occur due to dynamic topology changes and the dynamic arrival/departure of nodes in the network: (i) MANET initialization, (ii) new nodes joining the network, (iii) graceful departure of nodes, (iv) abrupt departure of nodes, (v) concurrent address requests, (vi) migration of the *requester*, (vii) message losses, and (viii) partitioning of network and subsequent merging. We address each of these scenarios in this section:

A. MANET Initialization

The *Initiator* of a MANET broadcasts *IPAddressRequest* message and waits for an *IPAddressAvail* message until the *offer_ip* timer expires. If it does not receive any *IPAddressAvail* message, it re-broadcasts the *IPAddressRequest* message. This continues for *requester_retry* times. This is to ensure that, if there are other configured nodes in the network, in the event of message losses the node does not assume itself to be the *Initiator* (details on message losses given in Section VII-A). After all the failed retries, the node concludes that it is the only node in the network. It then assigns itself the first IP address from the IP address block, initializes the *free_ip* set to the rest of the IP address block, and sets its *network_id* to the following 4 tuple *<Initiator's MAC address, Initiator's IP address, timestamp, random number>*.

B. New node joining the network

After MANET initialization, any new node (*requester*) appearing in the neighborhood of existing node(s) broadcasts an *IPAddressRequest* message and starts the *offer_ip* timer. Each address allocation is assigned a unique *transaction_id*. The *transaction_id* consists of the *requester's* MAC address and timestamp.

Let $free\_ip_{(X)}$ set at a configured node X in the neighborhood be $\{A_1, A_2, ..., A_n\}$. On receiving the *IPAddressRequest* message, node X computes k = $\lceil n/2 \rceil$.

- If k > 0, node X sends *IPAddressAvail* message to the *requester* with the addresses $\{A_k,...,A_n\}$ as the proposed address block (*proposed_block*)[4]. It updates its *free_ip* set to : $free\_ip_{(X)} = \{A_1, A_2, ..., A_{k-1}\}$. Node X also creates a {*transaction_id, proposed_block* } entry in its *pending_ip* set and starts the *allocation_pending* timer.
- Otherwise, node X sends a NULL *IPAddressAvail* message to the *requester*.

When the *offer_ip* timer expires[5], *requester* sorts all the received non-NULL *IPAddressAvail* replies based on the size of the *proposed_block*. The *requester* selects the neighbor with the largest *proposed_block*[6] as the *allocator*. The *requester* sends an *AllocatorChosen* message with the selected *allocator*'s IP address and corresponding *transaction_id* to all its neighbors. If all the received *IPAddressAvail* messages contain NULL responses, the *requester* randomly chooses one of the nodes as its *allocator*.

On receiving an *AllocatorChosen* message, each neighboring node that is not chosen as the *allocator*, removes the *proposed_block* from its *pending_ip* set for the corresponding *transaction_id* and appends it to its *free_ip* set. Thus, it is ensured that nodes not chosen as the *allocator* get back their offered address blocks.

On receiving the *AllocatorChosen* message, the node chosen as the *allocator* creates an entry in the *transaction_info* set for the *transaction_id*. The *transaction_info* is a set of ordered pairs. The first field of the ordered pair is the *transaction_id* and the second field is the *allocator*'s IP address. The *allocator*, then retrieves the *proposed_block* from its *pending_ip* set for the corresponding *transaction_id.*

1) **If proposed_block is not NULL:** The *allocator* cancels the *allocation_pending* timer and sends *IPAddressAssign* message with the *proposed_block* to the *requester*. It then removes the entry for the *transaction_id* from its *pending_ip* set.

2) **If proposed_block is NULL:** The *allocator* initiates an *expanding ring* search. The hop count *d* is initialized to 1. Each node in the network maintains a neighbor list. The neighbor list is updated by the exchange of *Hello* messages used for partition handling. The *allocator* performs the following sequence of steps during the expanding ring search until it finds either

[4]If the *proposed_block* is contiguous only the first and the last addresses are sent. A field can be added to the protocol header to indicate whether the message contains the range or individual addresses of the block.

[5]The first *IPAddressAvail* message received need not contain the largest *proposed_block*, hence we wait for a time-out to collect more *IPAddressAvail* messages.

[6]In case of a tie, one of the neighbors is chosen randomly.

a free IP address or all the nodes in the network are visited:

1) The *allocator* sends an *IPAddressInfoRequest* message containing its IP address, *transaction_id*, and hop count (*d*) to every node in its d-hop neighborhood and to all the nodes that did not respond during previous iterations. It also starts the *statecollection* timer and sends a *WaitPeriod* message to the *requester* with an estimated delay of the allocation process. The *requester* on receiving the *WaitPeriod* message extends its *assign_ip* timer.
2) On receiving an *IPAddressInfoReply* response, if the received *IPAddressInfoReply* message has a non-empty *free_ip* set, the *allocator* goes to step 3. Otherwise, it takes a union of all the received neighbor lists to check if the resultant union set is same as the set obtained during the previous iteration. If so, there are no nodes left to be visited. Thus, if the *IPAddressInfoReply* message responses do not have any free IP addresses and if all nodes in the network are visited, the *allocator* performs reclamation of IP address (explained in next subsection). Otherwise it goes to step 5.
3) The *allocator* sorts all the received non-empty *free_ip* sets based on their cardinality. It sends an *IPAddressChosen* message to the node with highest cardinality *free_ip* set. It also sends a *WaitPeriod* message to the *requester* with current hop count. It then starts the *confirm_ip* timer and waits for *IPAddressConfirm* message. If the timer expires before it receives an *IPAddressConfirm* message, it chooses the next node in the sorted list and repeats this step until either an *IPAddressConfirm* response is received or all the nodes in the list are visited. If all nodes in the list are visited and no free IP address is found, it goes to step 5.
4) If the *allocator* receives *IPAddressConfirm* message before *confirm_ip* timer expires, it cancels the *confirm_ip* timer and sends an *IPAddressAssign* message to the *requester* with the *proposed_block* received in the *IPAddressConfirm* message. This ends the IP address allocation process.
5) The *allocator* increments hop count *d* by an integer constant and goes to step 1.

During the expanding ring search, an intermediate node does the following:

- On receiving an *IPAddressInfoRequest* message, it responds with an *IPAddressInfoReply* message containing its *free_ip* set and neighbor list information.
- On receiving *IPAddressChosen* message, it performs actions similar to the ones performed

by the neighbors of the *requester* when they receive an *IPAddressRequest* message. It sends an *IPAddressConfirm* message with half of the addresses from its *free_ip* set as the *proposed_block* to the *allocator*. In case the *IPAddressConfirm* message is lost, the *allocator* eventually times out as explained in step 3 of the expanding ring search. The addresses of the *proposed_block* would be unavailable for allocation in future as they are assumed to be allocated and are removed from the *free_ip* set of the node. These addresses would be reclaimed during subsequent reclamation process (Section VI-B.3).

3) **Reclamation of IP addresses:** Suppose, no free IP addresses are found during the *expanding ring* search, it does not necessarily mean that there are no free IP addresses in the network. It is possible that some nodes may have left the network abruptly. Hence, reclamation of addresses needs to be done whereby the IP addresses of nodes that may have left abruptly are reclaimed. Reclamation of addresses is done by the *allocator* as follows:

- It finds the addresses that are not in use and forms the *missing_addresses* set where:
  missing_addresses = { *x:x* ∈ { *IP address block - alive_addresses* } }
  alive_addresses = { *IP addresses of nodes that responded during expanding ring search* }
- It floods a *TentativeFreeAddresses* message in the network. This message contains the *missing_addresses* set. Nodes that receive this message respond either with a *ConflictNotification* message (if the node's IP address is in the *missing_addresses* set) or with a *NoConflict* message. If the *allocator* receives a *ConflictNotification* message from any node, it removes the corresponding IP address from the *missing_addresses* set. If addresses in a node's *free_ip* set are in the *missing_addresses* set, then the node removes the corresponding addresses from its *free_ip* set. The resulting *missing_addresses* set contains addresses that are free.
- It then sends an *IPAddressAssign* message with the first address from these free IP addresses and a new *network_id* (<*allocator's MAC address, its IP address, timestamp, random number*>) to the *requester*. It then re-distributes the remaining free IP addresses among the nodes in the network by sending a network-wide *IPAddressUpdate* message. The addresses could be re-distributed randomly or evenly among the nodes. Instead of distributing reclaimed IP addresses randomly, they can also be allocated such that nodes finally have contiguous block of addresses. The latter approach would reduce the problem of address block fragmentation. The *IPAddressUpdate* message also carries the new *network_id*.

The nodes receiving the *IPAddressUpdate* message update their *network_id* and *free_ip* set. If nodes that had left the network later rejoin the network, their *network_id* would be different from the new *network_id*. Thus, by changing the *network_id* during reclamation it is ensured that rejoining of nodes that left the network is detected by the partition handling process (Section VI-G).

The *requester*, on receiving the *IPAddressAssign* message, cancels the *assign_ip* timer and configures itself with the first address from the *proposed_block*. It also sets its *free_ip* set to the remaining addresses in the *proposed_block* . If it does not receive an *IPAddressAssign* message before *assign_ip* timer expires, it retries *assign_retry* times. The likelihood of all the *assign_retry* attempts being unsuccessful due to message losses is very low (refer to Section VII-A for details).

C. Migration of Requester

Let a *requester* (say node X) move away from the *allocator* (say node Y) before Y could assign an IP address to X. Node X sends node Y's IP address to the new *allocator* (say node Z). Node Z sends *IPAddressForward* message to Y. Node Y, on receiving the *IPAddressForward* message, updates the entry for the corresponding *transaction_id* in the *transaction_info* set with the IP address of Z. Node Y then sends the *IPAddressAssign* message to node X via node Z. Alternately, the old *allocator* (Y) could abort the IP address allocation process when the *requester* migrates and the new *allocator* (Z) could start the allocation process afresh. In the former approach, the communication overhead is reduced by maintaining state information at the nodes. The latter approach does not maintain such state information but requires the address allocation process to be started all over again. Our simulations use the first approach.

D. Departure of a node:

The nodes in the network can either depart abruptly or gracefully from the network. The IP addresses and the *free_ip* sets of nodes that abruptly depart the network are reclaimed during subsequent IP address allocation processes. A node that wishes to gracefully depart the network sends a *HandOver* message with its IP address, *free_ip* set, and *pending_ip* set to one of its neighbors before leaving the network. The neighbor on receiving the *HandOver* message, appends the received IP address, *free_ip* set, and *pending_ip* set to its own *free_ip* set.

E. Concurrent Address Requests

If a node receives concurrent IP address requests, and if it has a non-empty *free_ip* set, then it allots disjoint IP address blocks from its *free_ip* set to the requesting nodes. If two *allocators* perform IP address reclamation concurrently, one of them suspends the allocation process as follows. Let the two *allocators* be P and Q with P's IP address higher than Q's. Both nodes flood the network with *TentativeFreeAddresses* message. A node that receives the *TentativeFreeAddresses* message from Q before receiving that from P, sends a *NoConflict*/*ConflictNotification* message (based on whether there is conflict between its address and addresses in the broadcast message) to Q and a *Defer* message to P. A node that receives the *TentativeFreeAddresses* message from P before receiving that from Q, responds with a *NoConflict*/*ConflictNotification* message to both the *allocators*. P suspends the IP address allocation process on receiving a *Defer* message. While Q continues the IP address allocation process. After completing the address allocation process, Q sends the *IPAddressUpdate* message to all nodes in the network. Nodes that had sent the *Defer* messages to P, now send *Resume* messages to P. Node P, on receiving the *Resume* message, re-broadcasts *TentativeFreeAddresses* message and continues the address reclamation process. After the reclamation process at Q, the *free_ip* sets at nodes in the network might have changed. Hence, it is possible that the addresses that were in *missing_addresses* set of P are either in use or are re-distributed among nodes to form their *free_ip* set. By re-broadcasting *TentativeFreeAddresses* after suspension, it is ensured that P has the updated network information. The proposed approach is similar to the one described in [12] and Ricart-Agrawala algorithm [25] for mutual exclusion. Unlike the Ricart-Agrawala algorithm, in our proposed scheme, explicit *Defer* messages are used to suspend the allocation process. In the absence of such *Defer* messages, the timer at the *allocator* would have expired and the allocation process at multiple *allocators* would have continued leading to duplicate IP address assignment. The approach described above avoids such duplication.

F. Message Losses

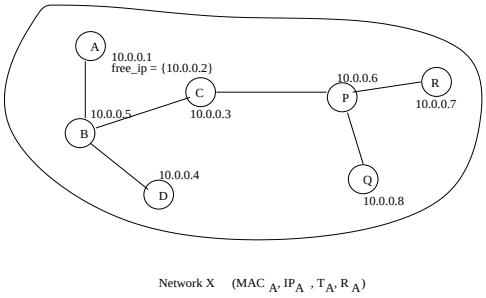
We propose to use UDP for communication. Hence, it is important to handle message losses as they can lead to duplicate IP address assignments. In this protocol, message losses are handled using appropriate timers and confirmation messages (refer to Section VII-A).

G. Partition Handling

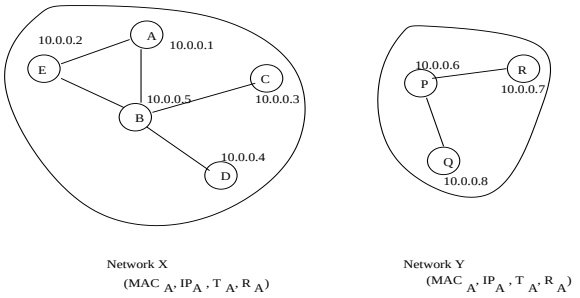
During the course of MANET operation, nodes can split from a network and form/join other networks. These networks can later merge into one. To detect merging of networks, each network needs a unique identifier (*network_id*). The *network_id* of the network is initially the following 4-tuple: <*Initiator's MAC address, Initiator's IP address, timestamp, random number*>, where *Initiator* is the first node that forms the network. The probability of two nodes having the same MAC address is low. Furthermore the probability of nodes having the same MAC address getting assigned identical IP addresses with equal timestamps is even lower. Adding the random number field to the 4-tuple, makes the probability of duplicate *network_ids* negligible. Thus, for all practical purposes, *network_ids* can be considered to be unique.

The *network_id* of a network is changed every time an address reclamation is performed. By changing the *network_id* it is ensured that networks have unique *network_ids*. The inclusion of timestamp as part of the *network_id* ensures that even if the same *allocator* performs IP address reclamation more than once, the resulting networks would still have different *network_ids*. If a MANET splits into multiple MANETs at time t_1 and the networks merge at a later time t_2 such that no reclamation is performed between time t_1 and t_2, then the *network_ids* will not change. This does not violate the correctness of the protocol because there are no conflicts among the networks merging.

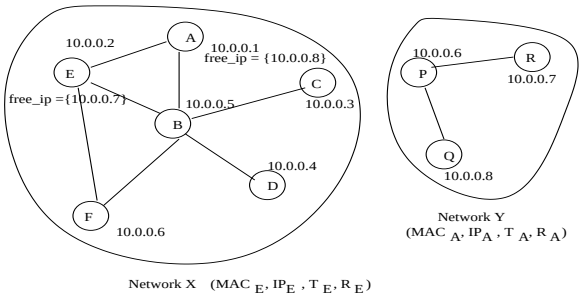
Figure 1 describes a sample network partitioning and merging process. Suppose *network_id* of network X is $<MAC_A, IP_A, T_A, R_A>$. Let *free_ip* sets of all the nodes except node A be empty (Figure 1(a)). Nodes P, Q, and R split from this network and form a new network Y. The nodes in networks X and Y have unique IP addresses, disjoint *free_ip* sets, and same *network_ids* after partitioning (Figure 1(b)). New nodes joining either networks are assigned unique IP addresses as long as the two networks do not initiate an IP address reclamation process. Now, suppose new node F joins the network X and let node E be the *allocator* (Figure 1(c)). Since *free_ip* set of node E is empty, it initiates an expanding ring search and finds that none of the active nodes in network X have a non-empty *free_ip* set. At this point, node E performs address reclamation whereby it finds that nodes P, Q, and R have left the network (the protocol does not distinguish between nodes leaving the network and abrupt crashing of nodes). Node E reclaims the IP addresses and the *free_ip* sets (in this example, they are empty) of P, Q, and R. Node E then propagates new



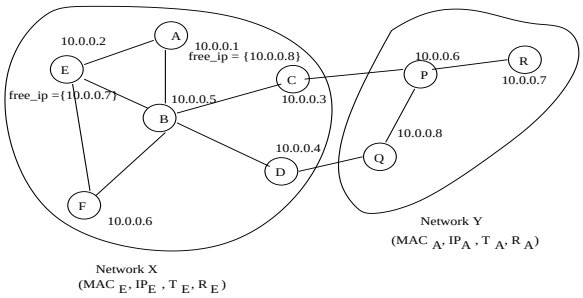
(a) Network X before Partitioning



(b) Network after Partitioning. Node E joins the network X after partitioning



(c) Network X performs reclamation of IP address



(d) Network X and Network Y merge

Fig. 1. Network Partitioning and Merging

network_id, <Node E's MAC address, Node E's IP address, timestamp, random number>, to all the nodes in network X. Thus, at the end of the reclamation process, the *network_ids* of X and Y are different (Figure 1(c)).

Detection of network merging is done as follows. Each node in the network locally broadcasts a periodic *Hello* message with its *network_id* and IP address to its neighbors. Whenever a node (say A) receives *Hello* message from a neighboring node (say B) containing a *network_id* different from its own, A detects merging of networks and responds to B. When B receives such a response from A, B detects merging of networks as well. Furthermore, if A has higher IP address than B (ties are broken using *network_ids*), then B (*merge_agent*) initiates a network-wide flood of *PartitionMergeQuery* message to collect the network configuration information in terms of IP addresses assigned to the nodes and their respective *free_ip* sets within its network. Node B asks node A (*co-merge_agent*) to do the same within A's network[7]. On receiving a

[7]By A's and B's network we mean the networks they belonged to prior to merging.

PartitionMergeQuery message, a node responds with a *PartitionMergeResponse* message, if its *network_id* matches with sender's *network_id*. The *PartitionMergeResponse* message contains IP address, *free_ip* set, and *network_id* of the node sending the message. Nodes receiving a *PartitionMergeQuery* message store the *network_ids* of networks being merged. Node A sends all the collected IP address and *free_ip* set information to node B. The assigned IP addresses and free IP addresses that are present in both the networks being merged are called as conflicting addresses and conflicting *free_ip* sets, respectively. By aggregating all the information obtained from the *PartitionMergeResponse* messages and information collected from A, node B learns of all the non-conflicting IP addresses and the number of nodes with conflicting IP addresses. For each conflicting IP address, node B further knows those nodes that own the IP address and the corresponding *network_id*. Based on all these information, node B performs a network-wide broadcast of an *IPAddressInvalidate* message containing three tuples of the following form: <x, y, z>. The first element, x, is a conflicting IP address and y is a *network_id*. While multiple nodes may have the same IP address, x, they can be differentiated by their *network_ids*. The first two elements (x and y) can uniquely identify the nodes whose address needs to be invalidated and the third element z[8] is the new IP address assigned to that particular node. Nodes receiving the *IPAddressInvalidate* message with their IP address and *network_id* contained in one of the 3-tuples, change their IP address to the new IP address[9] given in that three tuple. After the invalidation process, node B computes all the addresses not in use. It re-distributes these free IP addresses among nodes and propagates the new *network_id* to all the nodes in the merged network by sending an *IPAddressUpdate* message. The free IP addresses can be re-distributed such that the nodes finally have contiguous block of addresses. A node on receiving the *IPAddressUpdate* message updates its *network_id* and its *free_ip* set. Any node that misses the *IPAddressUpdate* message remains as part of the old network. Later, when the nodes in the network exchange *Hello* messages, this node gets merged into the network.

When more than one node initiates the network-wide flood of *PartitionMergeQuery* message simultaneously on detecting the same merger between networks, the *merge_agent* with lower IP address has precedence over the *merge_agent* with higher IP address. If a node receives *PartitionMergeQuery* messages from multiple nodes that detected the same merger concurrently,

[8]If there are no free IP addresses, then z is set to null.

[9]If the new IP address field is null, then the node invalidates its IP address and retries to acquire a new IP address later.

then: (i) If the node has already sent a response to one flood request, it ignores the *PartitionMergeQuery* message from the other nodes if the *merge_agent's* address in the received *PartitionMergeQuery* message is higher than the *merge_agent's* address in the previous flood message or if the *network_id* of the sender is different from its own *network_id*, else it responds; (ii) If the node itself is a *merge_agent* and receives a flood message from another *merge_agent* with lower IP address, the node aborts the merging process that it had initiated and responds to the received flood message. Thus, the *merge_agent* with the lowest IP address among all the *merge_agents* succeeds. Figure 1(d) illustrates this situation. Nodes C-P and D-Q detect the merging of networks X and Y simultaneously. Node C has a lower IP address than node P. So, C initiates network-wide flood within X and P initiates flood within Y on behalf of C. Both the flood messages contain node C's address as *merge_agent's* address. Similarly, node D (*merge_agent*) initiates the network-wide flood in X and Q initiates the network-wide flood on D's behalf in network Y. Later, when D learns about node C's merge process, node D aborts its merge process.

When more than two networks merge simultaneously, the merge requests are serialized based on the priorities of the *merge_agents*. If a node receives *PartitionMergeQuery* messages from multiple nodes that detected different mergers simultaneously, then: (i) If the node is a *merge_agent* and receives a flood message from another *merge_agent* with lower IP address, then the node suspends the merging process that it had initiated and responds to the flood message, (ii) If the node is a *co-merge_agent* and receives a flood message from a *merge_agent* with lower IP address, then the *co-merge_agent* sends a *Defer* message to the *merge_agent* with higher IP address. The suspended merge process is resumed on completion of ongoing merge process. This approach is quite similar to the one described in Section VI-E.

VII. Discussion

In this section we discuss the correctness of the address assignment process and the partition handling process. Theoretically, it is impossible to guarantee correctness in the event of message losses and node failures [4]. As message delays are non-deterministic and timers are best guesses, it is possible that timers may always expire prematurely and all the retry attempts fail all the time. In this section we prove the correctness based on the assumption that not all the retry attempts fail.

A. Correctness of IP address assignment

Given: Prior to an address assignment all the nodes have unique IP addresses and disjoint *free_ip* sets.

Assertion: Following the IP address assignment all the nodes have unique IP addresses and disjoint *free_ip* sets.

Proof: We prove the assertion in two parts. First, we prove the correctness of address assignment when there are no message losses. Then we relax our assumption and prove the correctness of address assignment in the event of message losses.

Part 1: Correctness under reliable communication: When an IP address request arrives, one of the following possibilities exist.

Possibility 1: The *allocator* has a non-empty *free_ip* set: The *allocator* allots the second half of its *free_ip* set to the *requester* and removes the corresponding addresses from its own *free_ip* set. The *requester* configures itself with the first address from the alloted set and forms its *free_ip* set with the remaining addresses in the alloted set (refer to Section VI-B.1 for details). Thus, following the IP address assignment, the IP address at both the *allocator* and the *requester* would be different and their *free_ip* sets disjoint. The IP addresses and *free_ip* sets at the other nodes in the network remain unaffected.

Possibility 2: The *allocator* has an empty *free_ip* set and finds a non-empty *free_ip* set in the network during expanding ring search: During expanding ring search, the *allocator* chooses one of the nodes (say X) with non-empty *free_ip* set and allots the second half of the addresses from X's *free_ip* set to the *requester*. X deletes the alloted addresses from its *free_ip* set (see Section VI-B.2). Thus, all the nodes in the the network have unique IP addresses and disjoint *free_ip* sets following the IP address assignment.

Possibility 3: The *allocator* does not find any free IP address during the expanding ring search: It does reclamation of addresses. As described in Section VI-B.3, after reclamation, the addresses in the *missing_addresses* set are free. One of the addresses from the resultant *missing_addresses* set is assigned to the *requester*. The remaining addresses are re-distributed among other nodes in the network.

Possibility 4: During concurrent IP address requests, a node X receives multiple IP address requests and has a non-empty *free_ip* set: X assigns different IP addresses and disjoint *free_ip*

sets to the *requesters* (Section VI-E).

Possibility 5: Multiple allocators perform reclamations concurrently: The *allocator(s)* with lower priority receives *Defer* message(s) and suspends the allocation process while the *allocator* with highest priority continues its allocation process. After the highest priority *allocator* completes the allocation process, the next highest priority *allocator* resumes its allocation process. Thus, concurrent IP address requests are serialized based on the priorities of the *allocator*s.

Therefore, when there are no message losses, all the five possibilities result in unique IP address assignment and disjoint *free_ip* set allotment.

Part 2: Correctness of address assignment in the event of message losses: Suppose the IP address assignment results in duplicate addresses in the network in the event of message losses. The following messages are exchanged during the address allocation process.

- If *IPAddressRequest* and/or *IPAddressAvail* messages are lost, when *offer_ip* timer expires, the *requester* retries by sending *IPAddressRequest* message for a maximum of *requester_retry* times. If it does not receive any response, it assigns itself an IP address, a *free_ip* set and a *network_id*. If *requester* is the only node in the network, then there are no duplicate addresses. If that is not the case, the *requester* assigns itself an IP address which may already be in use in the network. This amounts to a single node MANET (composed of the *requester*) co-located with another MANET consisting of all the other nodes. However, the *network_id* assigned to the *requester* would be different from the *network_id* of other nodes in the network. Eventually, when the MANETs are able to communicate with each other by network-merging process the duplicate address assignments would be detected and removed (Section VI-G). Thus, there would be no duplicate addresses in the network.
- If *AllocatorChosen* message is lost and the *assign_ip* timer at the *requester* expires, the *requester* retries by sending *IPAddressRequest* message and this IP address allocation process is started all over. This is done for a maximum of *assign_retry* times after which the *requester* is either successful in obtaining a unique IP address or it fails to acquire one.
- If *IPAddressInfoRequest* and/or *IPAddressInfoReply* messages from nodes get lost during expanding ring search, then during reclamation of addresses, these nodes would respond with *ConflictNotification* messages. If *ConflictNotification* message is also lost then either: (i) Node receives *IPAddressUpdate* message, invalidates its IP address and tries to acquire

a new IP address, or (ii) Node does not receive the *IPAddressUpdate* message, hence it would not receive the new *network_id*. This node remains as a part of the old network and eventually when it communicates with any node in the current network, it gets merged into the current network (Section VI-G).

- If *IPAddressChosen* and/or *IPAddressConfirm* messages are lost, the *allocator* on *confirm_ip* timer expiry sends *IPAddressChosen* message to another node with non-empty *free_ip* set and continues the allocation process. If the *IPAddressConfirm* message is lost, the proposed free block of IP addresses would no longer be available for allocation because they are assumed to be already allocated. Thus, it is possible that some addresses are unavailable temporarily. This does not violate the correctness requirement and those addresses would be available after subsequent address reclamation.

Thus, none of the message losses lead to duplicate address assignment. Therefore, we prove that IP address assignment process works correctly in the event of message losses.

B. Correctness of Partition Handling process

Given: Prior to network-merging all the nodes in the network have unique IP addresses and disjoint *free_ip* sets.

Assertion: Following network-merging, all the nodes in the merged network have unique IP addresses and disjoint *free_ip* sets.

Proof: The *network_id* assigned to each network is unique (see Section VI-G for details on uniqueness of *network_id*). The *Hello* messages are exchanged periodically to detect merging of networks. Thus, in finite time, the merging of networks would be detected. When two networks merge, there could be address conflicts. In the proposed scheme, the node responsible for configuring the merged network (*merge_agent*) collects the network-wide information in terms of IP addresses, *free_ip* sets, and *pending_ip* sets. If there are address conflicts, the *merge_agent* would send address invalidation message with new IP addresses for nodes with conflicting IP addresses. The *merge_agent* would also send a flood message to re-distribute the free IP addresses among the nodes in the merged network and update the *network_id* of the merged network. As all the conflicting addresses are removed and disjoint *free_ip* sets are distributed, all nodes in the merged network would have unique IP addresses and disjoint *free_ip* sets.

VIII. Simulation Experiments

Simulation experiments were performed using the network simulator ns-2 (ver 2.1b9a) [18] with CMU extensions to support MANETs to evaluate the performance of the protocol in terms of message complexity and latency.

A. Simulation scenario and parameters

The random waypoint mobility model was used. The speed of the nodes in the network was 5 meters/second and the pause time was 10 seconds. The simulation duration was 4500 seconds. The routing protocol used was AODV, although any routing protocol can be used. The *hello* timer period was 3 seconds, and the *requester_retry* threshold and *assign_retry* threshold were set to 3 and 2, respectively. The hop count *d* was incremented by 1 during the expanding ring search. Following terms are used in subsequent discussions:

1) Network density: the number of nodes per unit area.

2) Degree of a node: the number of nodes in the neighborhood of the node.

Two kinds of simulation experiments were carried out:

1) In the first kind, performance of the protocol was evaluated in terms of latency and communication overhead. The following three sets of simulation were performed:

a) Varying Node Population: Networks with *n* nodes were simulated where *n* varied from 50 nodes to 300 nodes. The area of the network was 408m $\times$ 408m, 578m $\times$ 578m, 816m $\times$ 816m and 1001m $\times$ 1001m for the 50, 100, 200 and 300 node networks, respectively (ensuring the network density of around 300 nodes/km^2). The size of the IP address block was 2 $\times$ *n*. Simulation started with zero configured nodes. The network was initialized with a single node. The inter-arrival time of new nodes in the MANET was exponentially distributed with mean of 0.2 node arrivals/second. The arriving node could appear anywhere within the area served by the network. Once the population reached *n*, departure of nodes began. The inter-departure time of nodes was exponentially distributed with a mean of 0.24 node departures/second. The arrivals and departures continued independently until the number of IP address allocations reached (n + 500) requests. Simulation results were collected for the last 500 address allocations. From these experiments the behavior of protocol for different node populations was analyzed.

b) Varying Address block range: Networks with 50 and 300 nodes, respectively with varying address block size were simulated. The area of network was set to 578m $\times$ 578m and 1415m $\times$ 1415m for the 50 node and 300 node systems, respectively. The address block size was varied from twice the node population to 10 times the node population. The arrival/departure pattern was the same as in the previous case. These experiments were used to analyze the effect of varying address block size on address allocation latency and communication overhead.

c) Varying Network Density: Networks with 50 and 300 nodes, respectively with varying network density were simulated. The address block size was three times the node population. The arrival/departure pattern was the same as in the previous case. The goal of these experiments was to study the address allocation latency and communication overhead with varying network density (which also affects the degree of each node).

2) In the second kind, the effect of the protocol on application traffic was analyzed: We used 50 and 300 node networks with 25% of node population generating CBR traffic. For every source there was a corresponding destination. The source and destination pairs were randomly selected. Each source injected packets with rate varying from 4 packets/second to 20 packets/second. The packet size was 512 bytes. The effect of the control traffic induced by our protocol on the end-to-end latency of application traffic was studied. This required us to run two sets of experiments for a given network scenario: (i) Using the proposed protocol for IP address assignment with its associated messages competing with application traffic for network bandwidth, and (ii) Using static IP address assignment with no resultant address assignment traffic. The area of the network was 578m $\times$ 578m and 1415m $\times$ 1415m for the 50 and 300 node systems, respectively.

Following statistics were gathered:

- Number of unicast messages: This includes the various unicast messages required namely *IPAddressAvail*, *AllocatorChosen*, *IPAddressInfoRequest*, *IPAddressInfoReply*, *IPAddressChosen*, *IPAddressConfirm*, *NoConflict*, *ConflictNotification*, *Defer*, *Resume* and *WaitPeriod* message.
- Latency: The total time taken by the protocol, from the instant the node enters the network and requests for an IP address until the node is assigned an IP address.
- Number of address reclamations: Number of times address reclamations were needed during the entire simulation. Two network-wide floods are required during each address reclama-

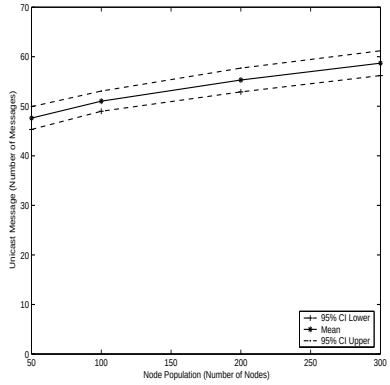
tion: (i) *TentativeFreeAddresses* sent by the *allocator* when none of the active nodes in the network have a non-empty *free_ip* set, and (ii) *IPAddressUpdate* message sent to update the *network_id* at nodes in the network.

- *Free_ip* set distribution: This gives the mean number of addresses in the *free_ip* set of the nodes in the network and the mean size of largest contiguous block of addresses in the *free_ip* set of the nodes in the network.
- Partition Handling overhead: This indicates the communication overhead incurred during network-merging, including the number of network-wide flood messages and number of unicast messages. The network-wide flood messages include the following: (i) *Partition-MergeQuery* flooded by the node detecting the network-merging, (ii) network-wide flood message with IP addresses to be invalidated, sent by the node responsible for configuring the merged network, and (iii) network-wide *IPAddressUpdate* message sent to all the nodes in the network. The unicast messages include the *PartitionMergeResponse* messages sent in response by the nodes in network for the *PartitionMergeQuery* message.
- Vulnerability Period: This is the total time taken by the protocol, from the instant network-merging is detected to the instant when the merged network is re-configured.
- End-to-end latency for application data: The time taken for application data to reach the destination node from the source node.

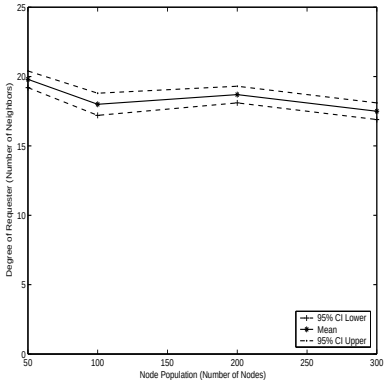
B. Simulation Results

1) Message overhead:

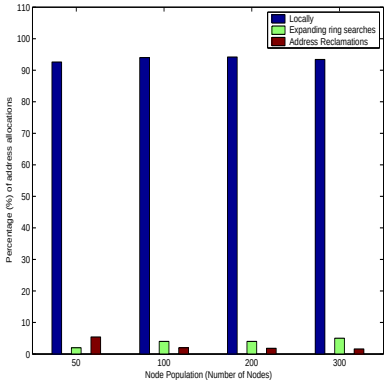
a) Impact of varying node population on number of unicast messages: The 95% confidence interval (CI) for mean number of unicast messages for varying node population over 500 address allocations is plotted in Figure 2(a). Figure 2(b) gives 95% confidence interval for mean degree of *requester* in the network. It was observed that, the mean degree of *requester* was about 20 and the mean number of unicast messages varied from around 50 to 60 messages. Every address allocation needs at least ((2 $\times$ degree of *requester*) + 1) unicast messages. Hence, the number of unicast messages is between 2.5 to 3 times the degree of *requester*. This is because most of the addresses were allocated locally. Only for few allocations, expanding ring search and reclamations were needed as can be seen from Figure 2(c). Thus, for the same network density,



(a) Node Population vs Number of Unicast Messages



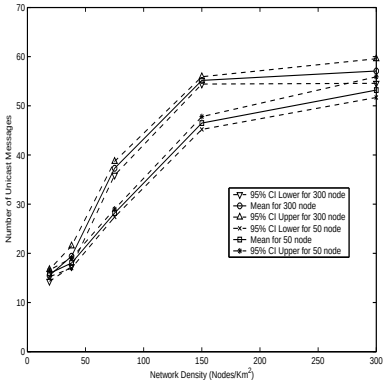
(b) Node Population vs Degree of *requester*



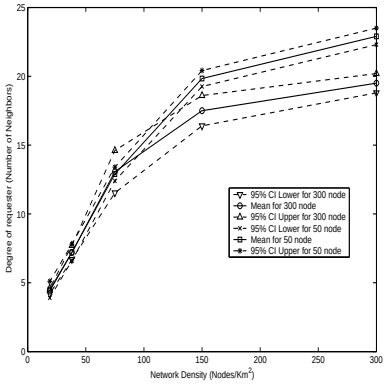
(c) Node Population vs Percentage of Expanding ring searches and Reclamations

Fig. 2. Communication overhead for varying Node population (address block size = 2 x node population, network density = 300 nodes/km^2)

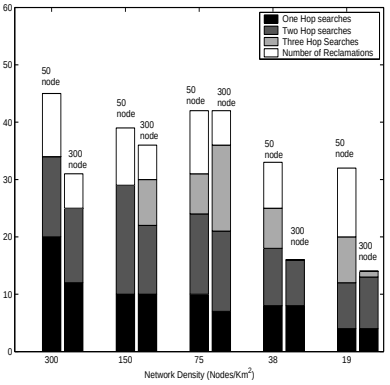
the number of unicast messages required increases sub-linearly with increase in network size.



(a) Network Density (nodes/km^2) vs Number of Unicast Messages



(b) Network Density (nodes/km^2) vs Degree of *requester*

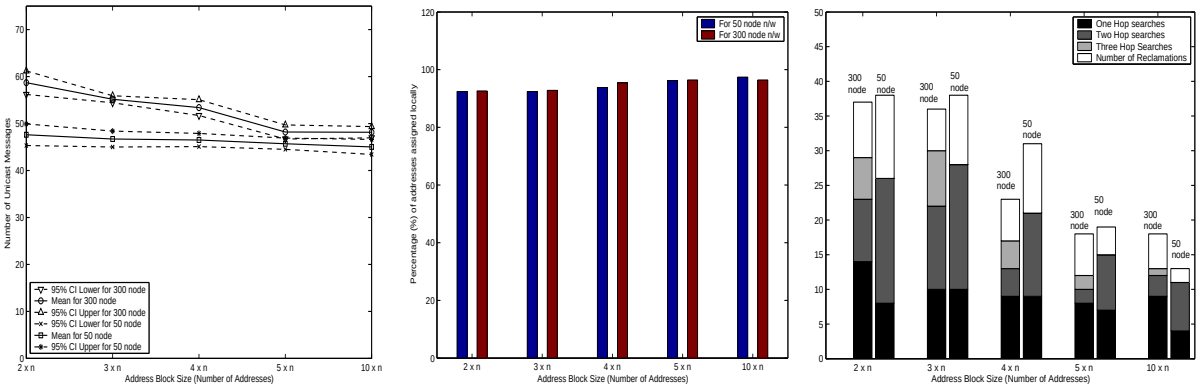


(c) Network Density (nodes/km^2) vs Expanding ring searches and reclamations

Fig. 3. Communication overhead for varying network density (node population = 50 and 300 nodes, Address Block Size = 3 x node population)

b) Impact of varying network density on number of unicast messages: Figure 3(a) shows the 95% confidence interval for mean number of unicast messages for 50 node and 300 node systems with varying network density. It was observed that the mean number of unicast messages exchanged increases with the increase in network density. In networks with high density, there are more nodes in the neighborhood of the *requester*, leading to a higher number of message

exchanges. Thus, we can say that the number of message exchanges required is a function of degree of nodes in the network. This can further be confirmed by Figure 3(b) which shows the degree of the *requester*. The number of unicast messages exchanged is 2 to 3 times the degree of *requester*. The same reason given in previous section applies. Figure 3(c) shows the number of expanding ring searches required along with their radius and the number of address reclamations done during the entire simulation duration. From Figure 3(c), we can infer that with the increase in area there was need to search farther in the network, with expanding ring searches upto a distance of 3 hops from the *allocator*. For networks with network density less than 75 nodes/km^2, we observed partitioning of network into smaller networks, which in turn lead to lower number of expanding ring searches and reclamations.



(a) Address Block Size vs Number of Unicast Messages

(b) Address Block Size vs Percentage of address allocations done locally

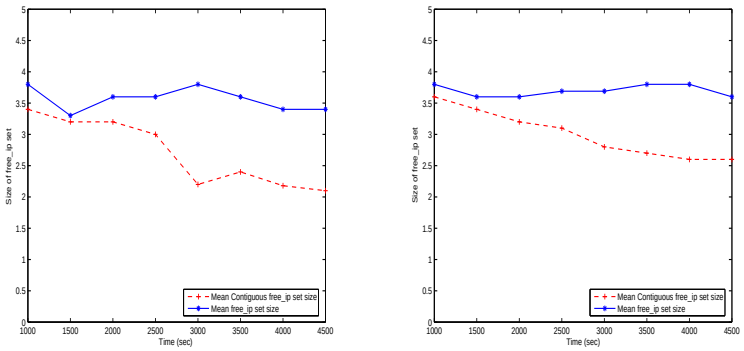
(c) Address Block Size vs Expanding ring searches and reclamations

Fig. 4. Communication overhead for varying address block size for 50 and 300 node network (network density = 150 nodes/km^2)

c) Impact of varying address block size on number of unicast messages: The 95% confidence interval for mean number of unicast messages over 500 address allocations for 50 node network and 300 node network is plotted in figure 4(a). From the plot, it can be seen that the mean number of unicast messages exchanged decreases as the size of address block increases. We observed around 17% decrease in number of unicast messages for a 300 node system when the address block size was increased from 2n to 10n. When the IP address block size is small, the probability of *requester* finding a free IP address within its neighborhood is small. Hence, the request needs to be propagated farther through the network, leading to a higher number

of message exchanges. Figure 4(b) shows the fraction of address requests that were completed locally by the *allocator*. It was observed that around 95% of addresses were allocated locally without requiring expanding ring search or reclamation of addresses. Figure 4(c) shows the number of address allocations that required expanding ring search and address reclamation to obtain an IP address. In our simulation experiments we observed that the number of expanding ring searches and the number of address reclamations decreased with increase in the address block size. Most of them were one or two hop searches.

d) Free_ip set distribution: Free_ip set distribution for 50 and 300 node systems is plotted in Figure 5. We collected snapshots at different stages of the simulation. In our simulations, the addresses are re-distributed randomly among nodes during reclamation and network-merging processes. During graceful departure of a node, the addresses given by the departing node to one of its neighbors need not be contiguous with the addresses in the neighbor's *free_ip* set. To find a node with *free_ip* set contiguous to addresses in *free_ip* set of the departing node would require significant communication overhead. We observed that the difference between mean size of *free_ip* sets and the mean size of contiguous block in *free_ip* sets is not much (around 1 to 2 addresses). As the simulation progresses, the mean size of contiguous block decreases, but again the decrease is not significant. Hence, there is little fragmentation of the *free_ip* set and not much reason for re-distribution of addresses to yield contiguous *free_ip* set at nodes.



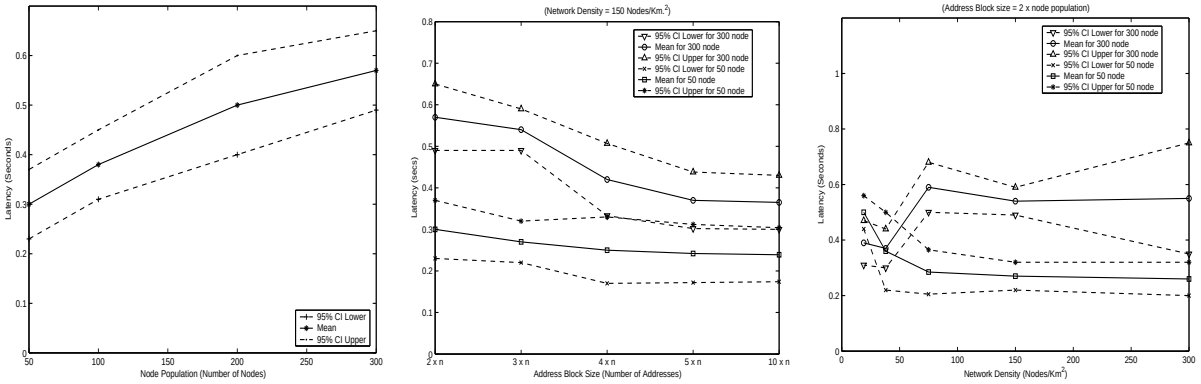
(a) Time vs Size of *free_ip* set for 50 node network

(b) Time vs Size of *free_ip* set for 300 node network

Fig. 5. Distribution of IP address blocks at different time instances for 50 and 300 node network (network density = 150 nodes/km^2, address block size = 5 x node population)

2) Latency:

a) Impact of varying node population on latency: Figure 6(a) shows the 95% confidence interval for mean latency for 50, 100, 200 and 300 node network scenarios. It was observed that around 95% of address allocations were completed within 0.19 seconds as the *requester* had neighbors with non-empty *free_ip* sets. A small fraction of the allocations did require as much as 4 seconds in a 300 node network. This was when the *allocator* had to perform a network-wide search for an IP address. Figure 6(a) shows an increase in latency with increase in node population. This was because as node population increased, for the same network density, the network diameter also increased. But again increase in latency is sub-linear with respect to the increase in node population. This is because of the same reasons provided in Section VIII-B.1.a.



(a) Node Population vs Latency (seconds)

(b) Address Block Size vs Latency (seconds)

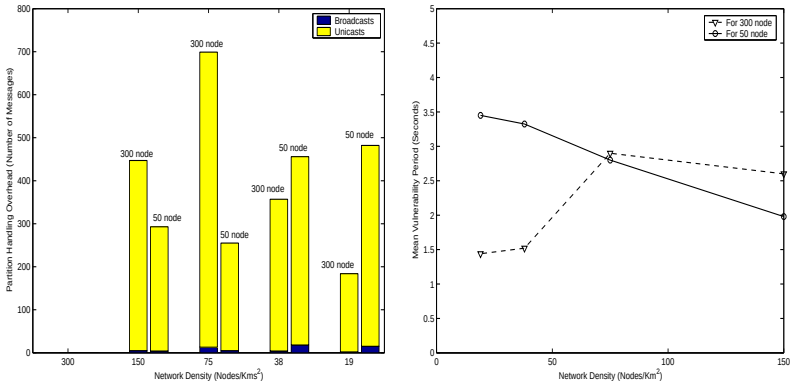
(c) Network Density (nodes/km^2) vs Latency (seconds)

Fig. 6. Latency for 50 and 300 node network with varying node population, varying address block size and varying network density

b) Impact of varying address block size on latency: The 95% confidence interval for mean latency for 50 and 300 node network scenarios is plotted in Figure 6(b) with varying address block size. It was observed that mean latency decreased with an increase in IP address block size. We observed a 33% decrease in latency when the address block size was increased from 2n to 10n in the 300 node system. Larger the address block size, greater is the probability of the *requester* finding a free IP address from its immediate neighbors. Hence, the number of expanding ring searches/address reclamations required decreases leading to lower latency (Figure 4(c)). Again the decrease in latency is sub-linear with increase in address block size.

c) Impact of varying network density on latency: The 95% confidence interval for mean latency for 50 node and 300 node network scenarios with varying network density is plotted

in Figure 6(c). It was observed that as the network density increased, mean latency decreased. At higher network density, there are more nodes in the neighborhood of the requester. Hence, the probability of a requester finding a free IP address from its immediate neighbors increases. Thus, the number of expanding ring searches/reclamations required is low. Even if there is need for expanding ring search, a free IP address is found within a few number of hops. At lower network density, more expanding ring searches are required leading to higher latency. In a 300 node network the latency increases upto a point, after which the networks have partitions and each partition in itself is a smaller network. Hence, we see a decrease in latency.



(a) Network Density (nodes/km^2) vs Partition Messages (Number of Messages)

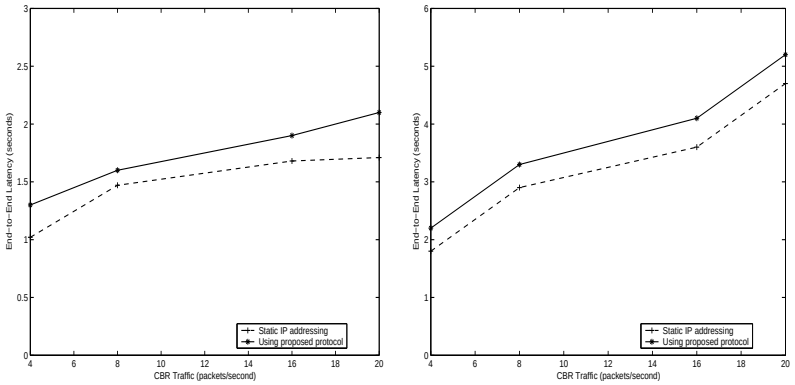
(b) Network Density (nodes/km^2) vs Vulnerability Period (seconds)

Fig. 7. Partition Handling overhead for varying network density

3) Overhead due to Partitioning:

a) Impact of varying network density on partition handling: Figure 7(a) shows the communication overhead incurred due to partition merging in 50 and 300 node networks with varying network density. The lower the network density, the greater the probability of network-partitioning. In our simulations we do not consider sparse networks which could experience disconnections and merger very often. In a 50 node system, we observed that the network had partitions but by the end of the simulation all the partitions had merged to form one single network. For a 300 node system, we observed that the network had partitions until the end of the simulation when the network density was low. Figure 7(b) shows the mean vulnerability period for 50 and 300 node networks with varying network density. During the vulnerability period, there could be duplicate addresses and it is possible that the traffic destined for one node

might get mis-routed to some other node. However, we observed that this period was always less than 4 seconds.



(a) CBR traffi c (packets/second) vs End-to-End Latency (seconds) for 50 node network

(b) CBR traffi c (packets/second) vs End-to-End Latency (seconds) for 300 node network

Fig. 8. Effect of protocol on application data (Address Block Size = 2 x node population)

4) Effect of protocol on application data: The mean end-to-end latency for CBR flows with and without the traffic generated by the proposed protocol is shown in Figure 8. We observed that there is a slight increase in end-to-end latency in the presence of control traffic generated by address assignment protocol. This is due to contention between the CBR traffic and protocol messages for communication medium. From Figure 8 we see that the application traffic is not adversely affected by the control traffic induced by our IP address assignment protocol.

From our simulation experiments, we can see that the communication overhead and latency is low when compared to the solution presented in [12]. This is because regardless of where a new node appears in the network, most IP address requests are satisfied locally incurring low latency and communication overhead. All the simulations were conducted with 2Mbps wireless channels. With current IEEE 802.11b and IEEE 802.11a hardware support for 11Mbps and 54Mbps channels respectively, latency would be less than what is shown in Figures 6 and 8.

IX. Conclusion

We presented a distributed protocol for dynamic configuration of nodes in MANETs. We have addressed the issue of unique IP address assignment to nodes in MANETs in the absence of any static configuration or centralized servers. The proposed protocol is based on the buddy system

[2], [3] used for memory management. The basic idea is to dynamically distribute the IP address block among the nodes in the network. Our approach guarantees unique IP address assignment under all network conditions including message losses, node crashes, network partitioning and merging.

We have presented the results obtained from simulation experiments. Simulation results show that most of the address allocations were done locally and required around 0.19 seconds. We observed that the latency and communication overhead increased in a sub-linear manner with increase in node population. We also observed that decrease in latency and communication overhead was sub-linear with increase in the address block size. Thus, the proposed protocol incurs low latency and communication overhead.

X. Acknowledgements

The authors wish to thank the anonymous reviewers for their useful suggestions that helped in improving the quality of this paper. The authors also wish to thank Mansoor Mohsin for helpful suggestions and discussions.

References

[1] R. Droms, "Dynamic Host Configuration Protocol," Network Working Group, RFC 2131, Mar 1997.

[2] K. Knowlton, "Fast Storage Allocator," Communications of the ACM, Oct 1965.

[3] J. L. Peterson and T. A. Norman, "Buddy Systems," Communications of the ACM, Jun 1977.

[4] N. A. Lynch, "A hundred impossibility proofs for distributed computing," Proceedings of 8th Annual Symposium on Principles of Distributed Computing, pp. 1-28, Aug 1989.

[5] D. B. Johnson, D. A. Maltz, Y. Hu and J. Jetcheva, "DSR: The Dynamic Source Routing for Multi-hop wireless Ad-hoc Networks,"*Ad Hoc Networking*, pp.139 - 172, Addison-Wesley, 2001.

[6] M. Jiang, J. Li and Y.C. Tay, "Cluster Based Routing Protocol (CBRP) Function Specification," Internet Draft, Aug 1999, work in progress, http://www.math.nus.edu.sg/~mattyc/cbrp.txt.

[7] V. D. Park and M. S. Corson, "A Highly Adaptive Distributed Routing Algorithm for Mobile Wireless Networks," *IEEE Conference on Computer Communications (Infocom '97)*, 1997.

[8] M. R. Pearlman and Z. J. Haas, "Determining the Optimal Configuration for the Zone Routing Protocol," *IEEE Journal on Selected Areas in Communications*, vol. 17, issue 8, pp. 1395 - 1414, Aug 1999.

[9] C. Perkins and P. Bhagwat, "Routing over Multihop Wireless Network in Mobile Computers,"*SIGCOMM'94 Computer Communications Review*, Oct 1994.

[10] C. Perkins and E. Royer, "Ad Hoc On-Demand Distance Vector Routing," *2nd IEEE Workshop on Selected Area in Communications*, pp. 90-100, Feb 1999.

[11] Cisco, "Duplicate MAC addresses on Cisco 3600 series," May 1997, http://www.cisco.com/warp/public/770/7.html.

[12] S. Nesargi and R. Prakash, "MANETconf: Configuration of Hosts in a Mobile Ad Hoc Network," *Proceedings of INFOCOM 2002*, 2002.

[13] S. Cheshire, B. Aboba and E. Guttman, "Dynamic Configuration of IPv4 Link-Local Addresses," Internet Draft, Jul 2004, work in progress, "http://files.zeroconf.org/draft-ietf-zeroconf-ipv4-linklocal.txt".

[14] C. Perkins, J. Malinen, R. Wakikawa, E. Royer and Y. Sun, "IP Address Autoconfiguration for Ad Hoc Networks," Internet Draft, Nov 2001, work in progress, http://people.nokia.net/~charliep/txt/aodvid/autoconf.txt.

[15] S. Thomson and T. Narten, "IPv6 Stateless Address Autoconfiguration," RFC 2462, Dec 1998.

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

[17] N. Vaidya, "Weak Duplicate Address Detection in Mobile Ad Hoc Networks," ACM International Symposium on Mobile Ad Hoc Networking and Computing (MobiHoc), Jun 2002.

[18] A. J. McAuley and K. Manousakis, "Self-Configuring Networks," *21st Century Military Communications Conference Proceedings*, Oct 2000, vol.1, pp. 315 - 319.

[19] A. Misra, S. Das, A. McAuley and S.K. Das, "Autoconfiguration, registration, and mobility management for pervasive computing," *IEEE Personal Communications*, pp. 24 - 31, Aug 2001.

[20] P. Patchipulusu, "Dynamic Address Allocation Protocols For Mobile Ad Hoc Networks," Master's Thesis, Texas A&M University, Aug 2001.

[21] H. Zhou, L. Ni, and M. Mutka, "Prophet Address Allocation for Large Scale MANETs," Proceedings of IEEE INFOCOM 2003, Mar 2003.

[22] K. Fall and K. Varadhan, "The ns Manual,", http://www.isi.edu/nsnam/ns/ns-documentation.html.

[23] K. Weniger, "Passive Duplicate Address Detection in Mobile Ad Hoc Networks," Proceedings of IEEE WCNC 2003, Mar 2003.

[24] J. Jeong, J. Park, H. Kim and D. Kim, "Ad Hoc IP Address Autoconfiguration," Internet Draft, draft-jeong-adhoc-ip-addr-autoconf-03.txt, work in progress, Jul 2004.

[25] G. Ricart and A. K. Agrawala, "An Optimal Algorithm for Mutual Exclusion in Computer Networks," Communications of the ACM, vol. 24, no.1, pp.9-17, Jan 1981.