AIPAC: Automatic IP Address Configuration in Mobile Ad Hoc Networks

M. Fazio, M. Villari, A. Puliafito
Università di Messina, Dipartimento di Matematica
Contrada Papardo - Salita Sperone, 31
98166 Messina, Italy.
e-mail: apulia{mfazio, mvillari}@ingegneria.unime.it

Abstract

We present an innovative protocol for the automatic configuration of the IP address in ad-hoc networks. This protocol assigns unique IP address to each node and manages possible duplicate addresses due to the mobility of nodes in the network. It avoids the storage of large amounts of data and makes use of procedures that minimize the number of exchanged packets. The protocol also provides a new mechanism, called Gradual Merging of networks, that causes the merging process of networks according to their evolution. Ns2 simulations are carried out to validate and characterize the performance of our proposed approach.

Keywords: ad hoc network, MANET, IP address, autoconfiguration, network configuration

I. INTRODUCTION

Ad hoc networks represent a dynamic communication systems independent from any fixed infrastructure. The benefits of ad hoc networks should combine with the easy configuration of networks devices. Manual identifiers configuration cannot be allowed, because users have to communicate with no previous agreements. The dynamic configuration mechanism used in the networks with a fixed infrastructure is the Dynamic Host Configuration Protocol (DHCP [1]). This technique is not suitable in ad hoc networks for some reasons: 1) nodes working as DHCP servers would not always remain active or connected while the network exists, 2) since energy resources of devices are limited, the configuration protocol should not overload some specific nodes for
managing addresses and 3) the bandwidth of wireless communication links is limited, so the configuration of nodes (especially in large networks) should take place with distributed approaches.

Thus, an important issue for ad hoc networks is the automatic configuration of network parameters and, in particular, of the IP Address [11].

An additional question concerns the merging and the partitioning of networks. In fact, nodes in two different networks can get the same IP address (IPx). If these two networks merge, a duplication of IPx address takes place, and the routing path toward IPx destination is not unique any more. An effective solution must therefore be found for the issue of address duplication. This solution must be related with the dynamics of network topology.

Some solutions proposed in literature use additional identifiers of several bits that travel in the network together with the IP address. This type of approach does not need to manage the network merging, but it causes an increase of traffic for data exchange because of the size of nodes identifier or can not guarantee the uniqueness of address. So they are not very suitable for large networks. Other solutions use network identifiers, and detect when the merging occurs. Then they reconfigure nodes in order to make a new single network. But merging and partitioning are sometimes so frequent (very dynamic networks or low density of nodes) that the immediate reorganization of the addresses for all the nodes might overload the devices.

In this paper we propose an innovative protocol, called AIPAC, to configure nodes in a MANET (Mobile Ad Hoc Network). Our purpose is to watch over the limited resources of devices in the ad hoc networks. In fact, in a spontaneous network users are very likely to make use of mobile and wireless devices like PDA and cellular phone. We think of a system that manages the address duplication reactively. This means working only when a node wishes to communicate with another one that has a duplicate address. AIPAC does not need to assure that no duplicate address is present, but that packets are correctly routed in the network. AIPAC makes use of network identifiers, but allows different network to coexist. It avoids to overload nodes and communication channels
whenever two networks merge. The reaction to the merging is quite slow in order to follow the evolution of the network topology. If two networks overlap for a long time, the system tends to take up only a network identifier instead of two.

We used ns2 simulator to test the AIPAC behavior and to evaluate its performance.

II. RELATED WORKS

Some solutions were presented in literature to solve the problem of the IP address automatic configuration in an ad hoc network. In [3] the authors propose that every mobile node obtains an unused IP address randomly on its own. At the beginning of allocation, a generic node chooses the seed for the whole MANET and the sequence of addresses may be computed locally. Therefore, that node is a prophet in the MANET, which means it knows in advance which addresses are going to be allocated. This is the reason why the authors call this algorithm prophet allocation. The uniqueness of the IP address is guaranteed by a very long sequence of random numbers generated by a specific seed. However, this mechanism does not exclude the possibility of generating duplicate addresses. In fact, they say that when an old address is assigned to a node, the previous node with the same address has probably already left the MANET.

A different mechanism devised by the MANET group of Internet Engineering TASK Force (IETF) is described in [10]. When a node is connected to the network, it selects a temporary address at random in the range 1-2047 in the class B 169.254/16 network. This address is used only for the negotiation of the final IP address. Then the node selects another address at random in the range 2048-65534 in the 169.254/16 network, and asks all the nodes of the network, in order to check whether this address is in use. If a reply is received, a new address must be found. Otherwise, the address selected is automatically considered available, and the node uses it as its final IP address. The specific address negotiation messages for the Ad hoc On-Demand Distance Vector (AODV) reactive routing are the ones used by the system.

Other solutions try to extend the automatic self-configuration of IPv6 to ad-hoc networks, by using site-local addresses [4]. These are addresses that enable packets to reach the nodes belonging to their site, without specifying the global prefix of the Ipv6 address.
The simplest solution [8] provides that all the nodes of the ad-hoc network belong to the same site. A site-local address consists of the first fixed 48 bits, of 16 subnet ID bits, and of 64 interface ID bits. Since the subnet ID has no physical meaning in an ad-hoc network (meaning that it does not provide any information about the routing procedures), each node can create its complete site-local address, by selecting its interface ID and subnet ID at random. The latter ID simply becomes an extension of the former. In order to make this address effective, a Duplicated Address Detection (DAD) procedure must be enabled, so to detect whether the site-local address is already in use. This can be done by using the standard Neighbor Discovery Protocol (NDP) messages: Neighbor Solicitation (NS) and Neighbor Advertisement (NA) [13]. The solution presented in [14] proposes a mechanism for assuring the uniqueness of the IP address even if some networks merge. A hierarchical structure is created, in order to avoid packet flooding for the DAD procedure in limited areas. Some leader nodes are elected in the network. Each leader selects a subnet ID that is made known to the nodes within its scope (set of nodes whose longest distance is less or equal to \( r_s \) steps from the node examined, where \( r_s \) is a parameter of the protocol). Such nodes use this information only for creating the site-local address, and leader nodes perform the DAD procedure only within their scope, in order to avoid duplicate Interface IDs. In order to have unique subnet ID in the whole network, the leader nodes must also start the DAD procedure among them for the 16 bits of subnet ID. The merging of networks is obtained either by enabling the DAD procedure among the leader nodes, or through the statistic analysis of the entries in the Neighbor Caches of nodes, whose sudden increase may be a sign of the merging. These solutions are not very efficient, because selecting the frequency for the DAD procedure is not easy. Furthermore, the increase of neighbor number may not be due to a merging (for instance, very dynamic networks), likewise a merging of networks may not immediately cause an increase in neighbor nodes.

In the approach proposed in [7] the authors create a system for the management of the IP addresses, which is distributed in all the nodes of the network. When a node (Requester) is going to enter, it has to rely on a configured node (Initiator), which
negotiates the address for the new node. Each node belonging to the network stores all the used addresses, as well as the ones that are going to be assigned to the new elements. This allows to know the available addresses at any time. Then the Initiator selects an address among the available ones, and asks the other nodes for the permission to use it. This is a way for checking whether the same address is being assigned in another part of the network. If all the nodes send a positive reply for this request, the address is assigned, and all the nodes of the network are notified. This mechanism therefore provides for two broadcasts by the Initiator (one for the IP request and one for the confirmation of the IP assignment) and a reply message (positive or negative) by all the nodes in the network. This implies a high volume of traffic in big networks, as well as some complexity for keeping the information updated in all the nodes of a network whose channels are not reliable. If a node decides to leave the network, it has to release the address, by sending a bye message in broadcast. The nodes that have suddenly disconnected are taken up, and the corresponding IP addresses are retrieved, since they do not respond to the next assignment procedures. The system also provides for a mechanism for managing the migration of the Requester to a new Initiator, without losing the work done for assigning the address. For managing the merging of different networks, a single network ID is used, which is selected by the node with the lowest IP address. So when nodes belonging to different networks get in contact, they detect the merging and check for possible duplicated addresses. The system has to verify also if network partitioning occurs. If some nodes do not respond to the subsequent assignment procedure of the IP address, the partitioning is detected. If such nodes also include the one that originally determined the network ID, a new node must be found with the lowest IP address. This node will then determine the new network ID. Since the IP assignment operations may not take place for a long time, and thus no partitioning can be detected, the node with the lowest IP address must broadcast a message from time to time to show its presence. One cannot easily determine how often this message needs to be sent, since this depends on the extension and on the dynamics of the network.
III. THE AIPAC SOLUTION

The AIPAC protocol allows to develop ad hoc networks whose nodes are automatically configured with IP addresses. A network is generally created as soon as two distinct nodes enter each other’s range (Network Initialization phase). This enables both nodes to obtain valid IP addresses. The protocol AIPAC locates the nodes without a valid IP address, by using the Host Identifier (HID). The node with the higher HID selects the configuration parameters for both nodes. After these steps, other nodes that join the networks (Requester) obtain the IP address through one of the configured nodes (Initiator). The Initiator negotiates in the network for a valid address for the Requester and then gives to it the network parameters for a correct configuration.

Along with the initial network configuration, the AIPAC protocol must manage the nodes that connect, disconnect, or move through the network. The mobility of nodes may cause those two distinct networks overlap. If these networks contain nodes with the same IPx address, a correct routing of the packets cannot be assured through the network. Figure 1 shows this problem in an ad hoc network.

In order to solve the problem of address duplication, AIPAC uses a Network ID (called NetID), similar to the UUID described in [7]. The NetID is a 4-byte number, and must be stored by all the nodes belonging to the network. It is determined by the node with the highest HID during the phase of Network Initialization. When some nodes with a different NetID come in contact, the merging of different networks is detected. Unlike the other proposals, AIPAC does not provide for an immediate management
of network merging that can solve any address duplication. In fact, if no node is involved in data exchange, a new configuration of the networks only means a waste of resources. AIPAC therefore provides a reactive management of network merging. The main difference between proactive and reactive approaches is that with the first, nodes exchanged periodically control packets in order to maintain update information on the network. Conversely with the second one, nodes exchange information only when they need to communicate. From comparisons between proactive and reactive protocols, like in [12], it proves that, in terms of control overhead, reactive protocols perform the best at high movement speed and for a large number of nodes. This is due to the capability of such kind of protocols to adapt to changes in the network topology, avoiding storage of outdated information. So they result the most versatile solutions for wide and dynamic network. For these reasons we developed a reactive protocol for the IP configuration task, in order to make the protocol suitable in cases in which resource consumption is critical. When two nodes need to exchange some packets, the Source checks that the Destination node has no duplicated addresses IPx. If the duplication is detected, AIPAC notifies the nodes with IPx address about the presence of duplication and forces them to search for unused addresses in the network.

The choice of associating each network with a NetID involves the managing the partitioning of a network into two or more parts. Imagine that a network is partitioned into A and B. A new node is configured with the IPx address, and after areas A and B merge. If the IPx address is duplicated, both nodes cannot be distinguished, since they are using the same (IP,NetID) pair. AIPAC implements an efficient mechanism to detect a partitioning within a network and avoid this kind of problems.

When different networks come in contact, they may split again within a short period of time. Otherwise, they may merge completely, until they form a single network. In the latter case, even if AIPAC works correctly in presence of several networks, the best condition is that only one network is created, because it means that all the nodes have unique addresses and the path discovery procedure does not suffer from delay in possible duplicate IP corrections. AIPAC provides for an innovative mechanism called
Gradual Merging of networks, which changes the NetID of the nodes according to the changes in the network topology. The nodes decide to switch from a NetID to another one according to the changes they detect in the topology. This mechanism acts uniformly and continuously to avoid overloading nodes and also transmission channels with control traffic.

In the next paragraph we provide a detailed analysis of the AIPAC solutions for the issues we have mentioned.

IV. The AIPAC implementation

We have designed AIPAC in order to minimize the amount of data storage and the network traffic load. In fact, ad hoc networks are characterized by limited resources of devices and fluctuating capacity of channels. In next sections we show how our protocol works.

The problem of the self-configuration of the node address can be divided into four parts:
1) initial configuration of nodes, which enables the nodes (that create or enter a network) to obtain a single IP address;
2) management of the merging of different networks. This is caused by the dynamics of the nodes the networks consist of;
3) management of the partitioning of a network into two or more different networks;
4) mechanism of Gradual Merging of networks.

Regarding the first issue, many possible solutions have already been proposed in literature: in Section IV-A we show how AIPAC combines some of them in order to get a new solution to the problem of the initial address configuration. Regarding the other issues, new and original solutions are presented in Section V, VI and VII.

A. Initial address configuration

The procedure of IP address configuration implemented in AIPAC is based on the mechanism proposed by the MANET group in [10]. This mechanism avoid the storage of a lot of information about the network in all the nodes, it does not produce too much traffic in the communication channels, and its implementation is very easy. Since
the work [10] does not provide a way for solving the problem of two nodes using the same temporary IP address, AIPAC uses the figures of Requester and Initiator, which is defined in [7]. According to this definition, the new node (Requester) relies on the Initiator (already configured), which negotiates the address for the new node.

In our solution, when a node is going to join an ad-hoc network, it selects the HID, and periodically broadcast a SendRequest packet, until a reply is received from at least one neighboring node. In this case, two conditions may occur: either the node that replied is isolated, or it belongs to an existing ad-hoc network. In the first case (that happens when two single nodes get in contact and are not therefore configured with a significant IP address), the procedures of Network Initialization must be started. In this process, the node with the highest ID selects the NetID for the new network that is being created, as well as the IP address for itself and for the second node, at random within the range of allowed values. Then it notifies the second node with the NetID and the IP (Initialize packet). In particular, AIPAC provides that the IP addresses must be selected at random in a range of \( n = 2^{32} \) possible values.

In the second case, the IP address of the node contacted is already configured. This node works as Initiator and starts the Assigning Address procedure. The Initiator selects an address at random among the allowed addresses, and sends in broadcast a Search/IP packet. The address selected is specified in the packet. Any node receiving this packet checks whether the address is known (whether this address belongs to it or to another node in its routing tables). If the match is detected, the node sends a reply Used/IP to the Initiator. When the Initiator receives the Used/IP message, the Assigning Address procedure is restarted, and a new address is selected. Conversely, if no reply is received for a given time interval (Search/IP timer), the Initiator sends the Search/IP packet again, in order to face up possible errors in wireless channels. If neither replies arrive, it means that the address is not used yet. Then the Initiator notifies the Requester with the NetID of the network and the IP address that it has to use.

AIPAC exhibits the following benefits over the model proposed in [10]: 1) confusion is less likely to occur among the IDs of the nodes to be configured, since two nodes with
the same HID that rely on the same Initiator are less likely rather than two nodes in [10] have the same temporary IP address, even because the HID consists of 32 bits chosen at random, while the temporary IP address consists of 11 bits only; 2) the address is assigned independently from the routing protocol selected, because the messages are defined by this scheme and do not need support from specific routing protocol.

Compared to the solution [7], we have discarded the address negotiation system, because it requires to update the list of IP addresses available in all of the nodes. This way, we have reduced the signaling traffic generated in the network, as well as less information to be stored in each device.

V. MANAGEMENT OF THE NETWORK MERGING

When different networks come in contact, their nodes can be distinguished through the NetID. Even if a network merging is detected, AIPAC does not take any action to correct possible duplicate addresses, until a node needs to send data. AIPAC works in combination with reactive routing protocols. It is not useful to couple a reactive approach for the management of IP addresses with a proactive protocol, since the resources saved with the former are used with the latter. When a node needs to send data, the reactive routing protocol must start the Route Discovery procedure, in order to determine the path between the source and the destination. AIPAC uses the Route Discovery packets of the routing protocol to check (and remove, if necessary) the address duplication of the destination node. This way, the data packets can be sent through the network correctly, by using the IP address only, and limiting the use of the resources of network devices.

Whenever a node X needs to send data, the routing protocol broadcast a Route_Request packet to discover the path between source (X) and destination (Y). Then it waits to receive a Route_Reply from the destination. AIPAC inserts the NetID of the destination node, along with the typical information of the routing protocol, in the Route_Reply packet. If several replies come from nodes with a different NetID, this means that the IP address of the destination node IPy is duplicated. In this case, the source X starts the Change_IP procedure. This facilitates notification to all the nodes with the IPy address about the address duplication, and to specify the nodes that need to change
their address.

VI. MANAGEMENT OF PARTITIONING

Since every network has a unique identifier (NetID), any network partitioning must be detected, so to avoid to create two networks with the same NetID. When a general system (consisting of a set of different networks) is considered, the partitioning to be detected is only the one among the nodes of the same network. Let us refer to Figure 2. If the link 1 that connects different networks is broken, this is not a problem. Otherwise, if the link 2 that splits the same network is broken, this is dangerous. A new node may enter part A, and can receive an address present in part B (IPx address). A merging between A and B causes an address duplication that cannot be detected with the above-mentioned mechanism, because the nodes IPx of part A and B would have also the same NetID2.

The pseudocode about the detection and the management of the partitioning in AIPAC is shown in Figure 3.

To minimize the amount of data stored in the network devices, we allow nodes to know just about the neighborhood, that is the set of nodes one hop away. The list of neighbors is kept in the Neighbor_Table and is regularly updated through Hello packets that each node sends periodically. Whenever a node X receive a Hello packet from a new neighbor S, it insert S in its Neighbor_Table. If X does not receive Hello packets...
for a TIMER_NEIGHBORS period, most likely a partitioning is occurred. The link breakage is detected by both nodes involved, but only the one with the highest IP starts the process of Partitioning Check (e.g., X node). Then, after deleting the S entry in the Neighbor_Table, it uses a Route_Request packet to search for a path toward the neighbor S. This path must be through just the nodes of its network. This is done for avoiding a fragmentation of the network. Otherwise, the management of the partitioning would become more complex. For this reason, the Route_Request packet is discarded whenever it passes from a network to another. If no reply about the path toward the missing node is received, AIPAC assumes that a network partitioning has occurred. So
X selects a new NetID, and informs the nodes of its own network to change the NetID sending a Change NetID packet. It specifies the original and the new NetIDs, so that the nodes of the other networks can be excluded from the process. Then it deletes all the entry in its Neighbor Table, to avoid unnecessary checking of partitioning during the reconfiguration of neighbors.

Of course, several nodes can detect the partitioning at the same time. For instance, in Figure 4(a), A, B and C detect a possible partitioning at the same time. In this case, three different situations may occur:

1) \( IP_a < IP_b < IP_c \): both A and B detect that the IP of C is higher than its own IP. They do not start the procedure of network reidentification, which is done by C within its partition. The network 2 changes its NetID, while the network 1 keeps its original NetID;

2) \( IP_a < IP_c < IP_b \): A does not start the procedure of network reidentification, and leaves this task to C, which in turn leaves it to B. In this case, the network 1 changes its NetID, while the network 2 keeps it unchanged;

3) \( IP_c < IP_a < IP_b \): both A and B take the task of providing a new NetID to partition 1, but the nodes that receive different update messages about the same original NetID automatically select the highest NetID as the new one.

If Change NetID packets are lost, some nodes do not renew their NetIDs, but the Gradual Merging process (see next section) guarantees the transition to the new NetID in a short time.

If a node departs from the network, this can cause a partitioning in the network, like in Figure 4(b). For this reason it sends in broadcast a goodbye packet, in which it specifies the IP address of just one neighbor. So the other neighbors start up the described procedure with reference to this IP address.

VII. GRADUAL MERGING

When networks with different NetIDs come in contact, the creation of one network with a single NetID is more convenient. In fact, the more the system is fragmented, the higher is the probability that duplicated addresses may occur. In presence of a duplicate
address, nodes need to change it in order to perform a correct communication. The procedure for address correction causes a delay in packet delivering. For this reason a very fragmented system is not desirable. Actually, the networks may not remain in contact for a long time, and they are very likely to restore the original state in a short time. In this case, the merging process of the networks should not be started as soon as the contact is detected. Furthermore, the selection of the NetID to be used for the global network may be difficult. In this case, the selection of the NetID of one network would be better, in order to avoid that all the nodes get involved in the change of network parameters. However, how can we select the NetID to be used, if we assume that the nodes keep minimal amount of information about the network (for instance, number of nodes, energy of the nodes, and reliability of the channels)?

In order to solve these problems, AIPAC provides a mechanism called \textit{Gradual Merging}, which causes the nodes to switch from a network to another, according to the evolution of the whole system. \textit{Gradual Merging} allows a very heterogeneous system to become more uniform, decreasing the number of different networks, as we proved in our previous work [5]. The idea of ”Gradual” means that the procedure follows the evolution of the networks in a long period of time. If two networks tend to merge, the system focuses...
on the NetID of the network with a higher number or with a higher density of nodes. Conversely, if the networks have few contact points, the procedure to distribute the NetID is stopped.

Figure 5 show the pseudocode of *Gradual Merging* procedure. Each node keeps the information about its neighborhood in the *Neighbor_Table*. So at any time it knows how many neighbors are present and the NetID of the networks they belong to. As long as the number of links with the nodes of its network (*n_mine*) is higher than the number of links with the nodes of a different network (*n_other*), the node keeps its own NetID. Otherwise, if the number of links with the second network is higher than the number of links with the nodes of its own network, the node switches to the other network. We indicate $\Delta n = n_{other} - n_{mine}$ as the gap between the number of nodes of the two networks being examined, and $n_{tot} = n_{other} + n_{mine}$ as the total number of neighbors observed. A node switches from a network to another one whenever the under condition is true:

```
1. void AlpacAgent::GradualMerging()
2. {
3.     n_mine=NumberNeighbor(my_netid)
4.     n_other=NumberNeighbor(other_netid);
5.     if (n_other>(n_mine+(n_mine+n_other)*igm))
6.         {
7.             sendGoodbye();
8.             state=REQUESTER;
9.             my_netid=0;
10.            my_ip=-1;
11.            initiator=Neighbor(other_netid);
12.            sendRequester(my_hid, initiator);
13.            nb_table.empty();
14.        } else
15.        tmrg.resched(TIMER_GRADUAL_MERGING);
16.    }
```

Fig. 5. Pseudocode about *Gradual Merging*. 
The parameter $i_{gm}$, called Gradual Merge Index, provides the nodes with the threshold for switching from their network to another one. Therefore $i_{gm}$ denotes the level of acceptable heterogeneity in the system. The higher $i_{gm}$ is, less are the nodes that can switch from a network to another. Conversely, the changes in configuration are favored with low values of $i_{gm}$, and the system tends to create a single network. If the condition (1) is verified for the node X, it needs to reconfigure its IP address with another that is unique in the new network. So it has to 1) comes back to the Requester state, 2) reset the network parameters as NetID and IP address, 3) choose as Initiator a neighbor belonging to the new network and sends it a SendRequest packet for a new IP address, 4) delete the information about the neighborhood because as Requester it can not executes the procedures of network management. Each node verifies the threshold condition every TIMER\_GRADUAL\_MERGING seconds.

\[
\frac{\Delta n}{n_{tot}} > i_{gm}
\]  

VIII. SIMULATION RESULTS

We implemented AIPAC with version 2.26 of the ns2 simulator [2] with the CMU extension to support ad hoc networks [6].

AIPAC is independent from the routing protocol used, but has to interact with it. In the simulation experiments, we used AODV routing protocol [9]. The simulation results are unaffected by this choice. In fact, we estimated the number/type of messages exchanged by our protocol and the address allocation latency, but not the quantitative evaluation of traffic load or delays. Moreover we did not perform simulations in which nodes transfer data coming from the application level, because we focused our attention in assessing the traffic generated by AIPAC, independently from the upper layers.

The interaction between AIPAC and AODV is disclosed in the implementation of the Route_Request and Route_Reply procedures, that have calls to the AIPAC protocol. In Figures 6 and 7 we show the new implementation of AODV:recvRequest(), that is the
procedure that manages Route Discovery packets, and of AODV:sendReply(), that is the procedure that generates Route Reply packets.

```c
1. void AODV::recvRequest(Packet *p)
2. {
3.     struct hdr_ip *hn = HDR_IP(p);
4.     struct hdr_aodv_request *rn = HDR_AODV_REQUEST(p);
5.     struct hdr_aipac_2 *hr2 = HDR_2(p);
6.     
7.     /*
8.     * We are either going to forward the REQUEST or generate a
9.     * REPLY. Before we do anything, we make sure that the node
10.    * is able to process the packet.
11.    */
12.    if ((aipac_state==INITIATOR)||(aipac_state==NORMAL))
13.    {
14.       ...aipac_recvA(p);
15.    }
16.    Packet_free(p);
17. }
```

Fig. 6. recvRequest() function in the AODV protocol with AIPAC.

A node processes the AODV Route Request packet (Figure 6) only if it has a valid IP address, that means it is in the NORMAL state or it is working as Initiator (INITIATOR state). If the node does not have a valid IP address, it cannot take part in routing operations. If the Route Request packet is processed, a call to an AIPAC procedure allows the management of partitioning in the network. When a Route Reply packet is generated (Figure 7), AIPAC puts into the packet’s header the NetID information, in order to verify if duplicated addresses are in the network.

Our simulations have been done in three different stages.

In the first stage we have investigated the following parameters in detail, which are typical of an ad hoc network:

- Percentage of nodes that communicate with each other according to the total number of nodes.
- Evaluation of the round-trip time of a packet between a source node and any
Fig. 7. sendReply() function in the AODV protocol with AIPAC.

possible destination.

In particular, we want to identify a reasonable value for the Search_IP timer, specified in Sec. IV-A

In the second stage we have therefore assessed the performances of the configuration protocol we proposed, as well as the overhead introduced in the communication.

In the third stage we have studied how the Gradual Merge Index $i_{gm}$ affects the traffic due to the merging and partitioning management.
By using the tool of the CMU random waypoint mobility model [6], we have created several scenarios with 10, 20, 30, 40, 50, and 60 nodes respectively into an area of 1000x1000 meters. In the first stage, the simulations were performed using static nodes; all of the nodes were switched on at random from t0=0 to t1=50 seconds, while only one node (N+1) was switched on at time t2=75s. This way, we have obtained an estimation of the nodes involved in broadcast communication, as well as the response times of the nodes after a specific request coming from node N+1. By working out the average of the highest values, we have obtained an estimate for the highest round-trip time. The

<table>
<thead>
<tr>
<th>Number of nodes</th>
<th>% nodes connected</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>50</td>
</tr>
<tr>
<td>20</td>
<td>60.5</td>
</tr>
<tr>
<td>30</td>
<td>83.33</td>
</tr>
<tr>
<td>40</td>
<td>87.5</td>
</tr>
<tr>
<td>50</td>
<td>92.0</td>
</tr>
<tr>
<td>60</td>
<td>93.333</td>
</tr>
</tbody>
</table>

(a) Percentage of connected nodes.

(b) Maximum round-trip time of a packet between two nodes.

Fig. 8. Communication among nodes in a two-dimensional space of 1000x1000 meters.

results are shown in Figure 8(a) and Figure 8(b). In the first column of Figure 8(a) we show the total number of nodes in the network, while in the second column we show the average percentage of the nodes that communicate with each other. We notice that the density of wireless coverage and the number of connected nodes increase, as long as nodes increase. The graph in Figure 8(b) shows the progress in the maximum round-trip time for all the sets of nodes. The values of round-trip times can be found in the ordinate, while in the abscissa we can find the number of nodes receiving the
SendRequest packet sent by node N+1. The number of nodes in the abscissa is binded to the percentage shown in Table 8(a). For instance, if we consider the set of 30 nodes, we realize that 50% of them (with the value of 15 nodes in abscissa) receive the packet in 0.05 sec. Conversely, no value can be found beyond 83% (value 26 in abscissa). On average, the longest time taken by a packet in its round-trip path is 0.11 seconds in any set of nodes, as we can find with the maximum value expressed in the graph for the 40 node set. Due to our remarks, we used the value of 0.250 sec for the Search_IP timer. This timer is the time interval within which nodes in the network must reply to an IP address assignment request (SendRequest packet) and it is correlated to the round-trip time of the network. We have set the Search_IP timer to much more of double of the maximum measured round-trip time, in order to consider also greater round-trip time in wider networks or processing capacity, congestion, loss of packets and retransmission delays. So the value of 0.25 sec for the Search_IP timer is great enough to avoid errors in response time for delay and it can be generalized to other types of ad hoc networks.

In the second stage of our analysis, we have created 6 more scenarios, one for each set of nodes. The simulation time has been 500 seconds. 50% of the nodes were started from t0=0 to t1=75sec, while the remaining 50% were started after t2=200sec.

The histograms in Figures 9(a) and 9(b) show configuration times for nodes. The number of configured nodes is shown in the ordinate (their value is normalized) and the configuration times are shown in the abscissa. These times are expressed in 4 different intervals. In fact, from simulations results, we noticed peaks in the distribution of configuration times. This is caused from the Search_IP timer that considerably affects the configuration time for nodes entering the network. During the Initial Configuration procedure, when Initiator selects an IP address, it performs two Search_IP broadcasts, in order to face up possible errors in wireless channels, as explained in Section IV-A. This means that, if there are not replies to the Search_IP request, configuration time for nodes that make use of Initiator is the sum of several elements: time of synchronization between Requester and Initiator, two Search_IP timer expirations (0.5 sec) and time
(a) Scenario with 10, 20, and 30 nodes.  (b) Scenario with 40, 50, and 60 nodes.

Fig. 9. Histograms referring to the configuration times.

to notify Requester of available address (Initialize packet). If a node is configured in less than 0.4 sec, it is obvious that it does not rely on an Initiator. In such case or a node X self-configures its own IP address (it is the case of a couple of nodes (X, Y) in which X starts the \textit{Network Initialization} procedure) and configuration times are very short (like 0.05 sec), or X depends on another node Y that does not perform the \textit{Search IP} procedure (it is the case of a couple of nodes (X,Y) in which Y starts the \textit{Network Initialization} procedure and X waits for configuration parameters). The second case involves a longer configuration time due to synchronization between nodes. A node is configured in more than 0.7 sec if, during the \textit{Search IP} procedure, a conflict is pointed out and the procedure is restarted one or more (n) times (\textit{configuration time} = \textit{n} \times \textit{round_trip time} + 2 \times \textit{Search IP timer}).

Summarizing in Figure 9(a) and 9(b):

- the first interval includes nodes that configure address by themselves because they are forming a new network and start the \textit{Network Initialization} procedure.
- the second interval represents the number of nodes receiving configuration parameters from a neighbor that starts the procedures of \textit{Network Initialization}.
- the third interval includes the nodes that need an Initiator for configuring a valid address. About 0.5 sec will be necessary to find an available address with the
selected *Search IP* timer.

- The fourth interval shows how the probability of collision on selected IP affects configuration time. It includes nodes for which the procedure of *Search IP* runs more of two times because the selected IP is already used in the network.

Configuration times are strongly dependent on the *Search IP* timer. A lot of nodes, especially if node density is high, receive a valid IP address after about 0.5 sec, that is the delay necessary to check duplications on the new address. To reduce configuration times, we should select a shorter *Search IP* timer, but this could cause errors in the configuration procedure due to propagation and processing delay. For this reason, we think that with the *Search IP* timer we have set, we get the right tradeoff between configuration time and correctness in configuration. The Figure 9(a) and 9(b) show data for several set of nodes. The histogram for 10 nodes shows that the most of nodes are configured in the first two intervals, that means without Initiators. This happens because the nodes are far away and more subnets with several NetID are configured. On the contrary, the histogram for 60 nodes shows that the most of the nodes are configured in the interval 0.4-0.6 sec, because the nodes are close to each other. Thus, it is more likely that a new node finds an Initiator. We need to outline that the configuration times remain unchanged even with the variation in the number of nodes involved, because the network is configured in independent subnets with different NetIDs and IPs. As we have already said in the previous sections, the *Gradual Merging* process will gradually create a single network.

Other simulations have been performed for analyzing the time taken by the protocol for the solution of collisions.

Finally, we have analyzed messages exchanged during the procedure of configuration. Currently the algorithm AIPAC relies on AODV. The packets of AODV on ns2 used for implementing the features of AIPAC are *Hello* (to update the *Neighbor Tables*) and *Route Request* (to check for partitioning). The most important packets generated by AIPAC are *Initialize* (to give configuration parameters to Requesters) and *SendRequest* (to ask for configuration parameters). The overhead introduced by these packets can
be assessed in Figures 10 and 11. The graphs generated here are characterized by the number of packets exchanged among the nodes (in the ordinate), and by the time interval (0-500 sec) in the abscissa. The total number of messages exchanged between all the nodes in the interval $t=0$ and at the time $t_x$ can be found in the ordinate. The graph in Figure 10(a) refers to the $Hello$ broadcasts. The traffic is generated similarly to the protocol AODV. In this case, the number of packets sent tends to increase, because the $Hello$ messages are continually sent. Furthermore, we notice that the inclination of the curve increases after 200s, due to an increase in the number of nodes that take part
in the communication. The graph in Figure 11(b) refers to the SendRequest packets of AIPAC. We need to outline that the number of packets sent becomes stable with 40 and 60 nodes, while the number grows with 20 nodes. This happens because some node likely remains isolated in the 20 node scenario. They can not be configured and then they continue sending configuration requests. We can conclude that, after checking the number of exchanged packets, the overhead resulting from the configuration stage is less if referred to the number of Hello messages exchanged (Hello messages present in the AODV). The number of Hello messages exchanged is higher than the other packets exchanged by two orders of magnitude. The values of the ordinates of the graphs point out this assumption. Finally, according to our simulations, we can outline that the algorithm examined shows these benefits:

- It does not change the configuration times, if the number of nodes present in the network is increased.
- It solves the IP conflicts, without changing the configuration times significantly.
- The overhead introduced is not high, if compared with the traffic generated by the Hello messages of the routing protocol.

![Graph](image)

**Fig. 12.** Number of packets exchanged varying \(i_{gm}\) with 20, 40, and 60 nodes.

The results can therefore be considered consistent with the remarks we have done during the design.
In the third stage of our study we have tried to optimize the merging operations according
to the traffic generated. The simulations have been done with the same parameters and
scenarios used in the previous stage. All the nodes were switched on at random from
t0=0 to t1=50 seconds. The tests have been done according to the $i_{gm}$ parameter. As we
mentioned before, the nodes know the number of neighbors belonging to its network
($n_{\text{mine}}$), as well as the number of neighbors belonging to a network different than its
own ($n_{\text{other}}$). They periodically evaluate if the condition (1) is verified and if they
have to switch to the new network. $i_{gm}$ can have values in the range [-1,1]. However,
our study is focused on the positive values of $i_{gm}$, because the condition for switching
from a network to another makes sense when $\Delta n > 0$ (that is, when a node see several
neighbors belonging to another network). Furthermore, this definition of $i_{gm}$ allows to
optimize the protocol, notwithstanding the density of nodes in the network since the
value of $\Delta n$ that allows the merging is related with the total number of observed nodes.
For instance, if $i_{gm} = 0.3$, a node with $n_{\text{tot}} = 5$ switches from a network to another if
$\Delta n$ equals at least 2 (e.g. $n_{\text{other}} = 4$, $n_{\text{mine}} = 1$). Conversely, if the same node has
$n_{\text{tot}} = 20$, the merging happens only when $\Delta n$ is higher than 6 (e.g. $n_{\text{other}} = 14$
$n_{\text{mine}} = 6$).

Our purpose is to select a value for $i_{gm}$, so to minimize the total traffic. The graph
of Figure 12 shows the total traffic generated with the variation of $i_{gm}$. This traffic is
characterized by the procedures of Gradual Merging and of partitioning management.
Partitioning traffic is bound to the degree of homogeneity in the whole system. In fact,
as explained in Section VI, if two nodes with the same NetID detect a link breakage,
partitioning is occurred if there is not a path between such nodes through just the nodes
of their network. If the system has only one NetID and it is not splitted into isolated
partitions, there is a path between nodes through just the nodes of their network. If
the system is fragmented (it means a lot of subnetworks with different NetIDs), even
if it is not splitted into isolated partitions, a partition can be detected, because the
path between nodes includes nodes with different NetID. So the more the system is
fragmented, the higher is the partitioning traffic. Low values of $i_{gm}$ means that the nodes are more likely to merge, and that the whole system tends to a uniform NetID. This leads to a low partitioning traffic. However, the trend to switch from a network to another is high, so the traffic for merging is high. High values of $i_{gm}$ means that the nodes are less likely to merge, and that the system is therefore more heterogeneous. This leads to a low merging traffic, but the partitioning traffic is high. The minimum in the graph shows the value of $i_{gm}$ where the merging and the partitioning are balanced, and generate as less traffic as possible. According to the measurements done for 60 nodes, we find that the optimal value of $i_{gm}$ is 0.3. When $\Delta n$ tends to 0, and $i_{gm}$ therefore tends to 0, the nodes tend to switch from a network to another frequently. The reason is that the merging happens even when just one node arrives or leaves. The value of $i_{gm}=0.3$ is quite far from 0, so this flickering is avoided and then the merging traffic is limited. At the same time, a limited traffic for the partitioning management is obtained. Consequently, frequent NetID configurations are avoided.

We can conclude that these simulations allowed us to select optimal values for configuration parameters, as well as to assess the correctness of the algorithm and the overhead introduced by the control packets of the protocol.

IX. Conclusions

In this paper we have presented AIPAC, an innovative protocol that provides for a self-configuration mechanism for IP address assignment in ad-hoc network, as well as the way of maintaining its uniqueness after merging into a single network due to the mobility of nodes. We have focused our attention mainly on how AIPAC supports the characteristics of ad-hoc networks (that is, the limited resources of the devices and the unreliability of wireless channels). Each network is identified with its NetID. When two networks merge, and the merging is persistent, the different NetIDs should not be kept. However, determining which network should change its NetID is not easy, if the information is unknown about the whole new network. A Gradual Merging mechanism has therefore been designed. This allows a node to switch from a NetID to another, according to what is observed by the node about the changes in the network.
REFERENCES