An Efficient and Robust Addressing Protocol for Node Autoconfiguration in Ad Hoc Networks

Natalia Castro Fernandes, Marcelo Duffles Donato Moreira, and Otto Carlos Muniz Bandeira Duarte

***Abstract*—Address assignment is a key challenge in ad hoc networks due to the lack of infrastructure. Autonomous addressing protocols require a distributed and self-managed mechanism to avoid address collisions in a dynamic network with fading channels, frequent partitions, and joining/leaving nodes. We propose and analyze a lightweight protocol that configures mobile ad hoc nodes based on a distributed address database stored in filters that reduces the control load and makes the proposal robust to packet losses and network partitions. We evaluate the performance of our protocol, considering joining nodes, partition merging events, and network initialization. Simulation results show that our protocol resolves all the address collisions and also reduces the control traffic when compared to previously proposed protocols.**

***Index Terms*—Ad hoc networks, computer network management.**

I. Introduction

MOBILE ad hoc networks do not require any previous infrastructure and rely on dynamic multihop topologies for traffic forwarding. The lack of a centralized administration makes these networks attractive for several distributed applications, such as sensing, Internet access to deprived communities, and disaster recovering. A crucial and usually unaddressed issue of ad hoc networks is the frequent network partitions. Network partitions, caused by node mobility, fading channels [1], and nodes joining and leaving the network, can disrupt the distributed network control. Network initialization is another challenging issue because of the lack of servers in the network [2].

As other wireless networks, ad hoc nodes also need a unique network address to enable multihop routing and full connectivity. Address assignment in ad hoc networks, however, is even more challenging due to the self-organized nature of these environments. Centralized mechanisms, such as the Dynamic Host Configuration Protocol (DHCP) or the Network Address Translation (NAT), conflict with the distributed nature of ad hoc networks and do not address network partitioning and merging.

In this paper, we propose and analyze an efficient approach called Filter-based Addressing Protocol (FAP) [3]. The proposed protocol maintains a distributed database stored in filters containing the currently allocated addresses in a compact fashion. We consider both the Bloom filter and a proposed filter, called Sequence filter, to design a filter-based protocol that assures both the univocal address configuration of the nodes joining the network and the detection of address collisions after merging partitions. Our filter-based approach simplifies the univocal address allocation and the detection of address collisions because every node can easily check whether an address is already assigned or not. We also propose to use the hash of this filter as a partition identifier, providing an important feature for an easy detection of network partitions. Hence, we introduce the filters to store the allocated addresses without incurring in high storage overhead. The filters are distributed maintained by exchanging the hash of the filters among neighbors. This allows nodes to detect with a small control overhead neighbors using different filters, which could cause address collisions. Hence, our proposal is a robust addressing scheme because it guarantees that all nodes share the same allocated list.

We compare FAP performance with the main address autoconfiguration proposals for ad hoc networks [4]–[6]. Analysis and simulation experiments show that FAP achieves low communication overhead and low latency, resolving all address collisions even in network partition merging events. These results are mainly correlated to the use of filters because they reduce the number of tries to allocate an address to a joining node, as well as they reduce the number of false positives in the partition merging events, when compared to other proposals, which reduces message overhead.

The remainder of this paper is structured as follows. We overview the related work in Section II. The proposed protocol is then detailed in Section III and the analytical evaluation in Section IV. We describe the simulation results in Section V. Finally, Section VI concludes the paper.

II. Related Work

The lack of servers hinders the use of centralized addressing schemes in ad hoc networks. In simple distributed addressing schemes, however, it is hard to avoid duplicated addresses because a random choice of an address by each node would result in a high collision probability, as demonstrated by the birthday paradox [7]. The IETF Zeroconf working group proposes a hardware-based addressing scheme [8], which assigns an IPv6 network address to a node based on the device MAC address. Nevertheless, if the number of bits in the address suffix is smaller than number of bits in the MAC address, which is always true for IPv4 addresses, this solution must be adapted by hashing the MAC address to fit in the address suffix. Hashing the MAC address, however, is similar to a random address choice and does not guarantee a collision-free address allocation.

Manuscript received October 22, 2010; revised July 06, 2011 and July 06, 2011; accepted July 26, 2012; approved by IEEE/ACM Transactions on Networking Editor Y. Bejerano. Date of publication January 11, 2013; date of current version June 12, 2013. This work was supported by CNPq, CAPES, FINEP, FUNTTEL, FAPERJ, and FUJB. A preliminary version of this paper was published in the proceedings of the IEEE International Conference on Computer Communications (INFOCOM), Rio de Janeiro, Brazil, April 19–25, 2009.

The authors are with the GTA/COPPE, Universidade Federal do Rio de Janeiro, Rio de Janeiro 21941-972, Brazil (e-mail: natalia@gta.ufrj.br; marcelo@gta.ufrj.br; otto@gta.ufrj.br).

Digital Object Identifier 10.1109/TNET.2012.2227977

Address autoconfiguration proposals that do not store the list of allocated addresses are typically based on a distributed protocol called Duplicate Address Detection (DAD) [4]. In this protocol, every joining node randomly chooses an address and floods the network with an Address Request message (AREQ) for a number of times to guarantee that all nodes receive the new allocated address. If the randomly chosen address is already allocated to another node, this node advertises the duplication to the joining node sending an Address Reply message (AREP). When the joining node receives an AREP, it randomly chooses another address and repeats the flooding process. Otherwise, it allocates the chosen address. This proposal, however, does not take into account network partitions and is not suitable for ad hoc networks.

A few extensions to the Duplicate Address Detection (DAD) protocol use Hello messages and partition identifiers to handle network partitions [5], [9]. These identifiers are random numbers that identify each network partition. A group of nodes changes its partition identifier whenever it identifies a partition or when partitions merge. Fan and Subramani propose a protocol based on DAD to solve address collisions in the presence of network merging events. This protocol considers that two partitions are merging when a node receives a Hello message with a partition identifier different from its own identifier or when the neighbor set of any node changes [5].

Other proposals use routing information to work around the addressing problem. Weak DAD [10], for instance, routes packets correctly even if there is an address collision. In this protocol, every node is identified by its address and a key. DAD is executed on the 1-hop neighborhood, and collisions with the other nodes are identified by information from the routing protocol. If some nodes choose the same address and key, however, the collision is not detected. Moreover, Weak DAD depends on modifying the routing protocols.

Other more complex protocols were proposed to improve the performance of network merging detection and address reallocation [6], [11]. In these protocols, nodes store additional data structures to run the addressing protocol. MANETconf [6] is a stateful protocol based on the concepts of mutual exclusion of the Ricart–Agrawala algorithm. In this protocol, nodes store two address lists: the Allocated list and the Allocated Pending list. A joining node asks for an address to a neighbor, which becomes a leader in the address allocation procedure. The leader chooses an available address, stores it on the Allocated Pending list, and floods the network. If all MANETconf nodes accept the allocation request and positively answer to the leader, then the leader informs the allocated address to the joining node, moves the allocated address to the Allocated list, and floods the network again to confirm the address allocation. After receiving this message, each node moves the address from the Allocated Pending list to Allocated list. MANETconf handles address reallocation, but partition detection depends on periodic flooding. Therefore, this protocol incurs in a high control overhead.

Another stateful protocol is the Dynamic Address assignment Protocol (DAP) in mobile ad hoc networks [11], which is based on available-address sets, Hello messages, and partition identifiers. In DAP, a node subdivides its available address set with a joining node whenever it is argued for an address by the joining node. When a node has an empty address set, it asks for an address set reallocation. This reallocation and the detection that a given address is not being used anymore can cause a high control load in the network, depending on how the addresses are distributed among nodes. DAP requires the use of DAD in merging events not only for the allocated addresses, but also for the available address list stored in each node, increasing the control load.

Prophet [12] allocates addresses based on a pseudo-random function with high entropy. The first node in the network, called prophet, chooses a seed for a random sequence and assigns addresses to any joining node that contacts it. The joining nodes start to assign addresses to other nodes from different points of the random sequence, constructing an address assignment tree. Prophet does not flood the network and, as a consequence, generates a low control load. The protocol, however, requires an address range much larger than the previous protocols to support the same number of nodes in the network. Moreover, it depends on the quality of the pseudo-random generator to avoid duplicated addresses. Therefore, it needs a mechanism, like DAD, to detect duplicated addresses, which increases the protocol complexity and eliminates the advantage of a low control message overhead.

Our proposal aims to reduce the control load and to improve partition merging detections without requiring high storage capacity. These objectives are achieved through small filters and an accurate distributed mechanism to update the states in nodes. Furthermore, we propose the use of the filter signature (i.e., a hash of the filter) as a partition identifier instead of random numbers. The filter signature represents the set of all the nodes within the partition. Therefore, if the set of assigned addresses changes, the filter signature also changes. Actually, when using random numbers to identify the partition instead of hash of the filter, the identifier does not change with the set of assigned addresses. Therefore, filter signatures improves the ability to correctly detect and merge partitions.

III. FAP

The proposed protocol aims to dynamically autoconfigure network addresses, resolving collisions with a low control load, even in joining or merging events. To achieve all these objectives, FAP uses a distributed compact filter to represent the current set of allocated addresses. This filter is present at every node to simplify frequent node joining events and reduce the control overhead required to solve address collisions inherent in random assignments. Moreover, we propose the filter signature, which is the hash of the address filter, as a partition identifier. The filter signature is an important feature for easily detecting network merging events, in which address conflicts may occur.

We propose the use of two different filters, depending on the scenario: the Bloom filter, which is based on hash functions, and the Sequence filter, proposed in this paper, which compresses data based on the address sequence.

A. Bloom Filters

The Bloom filter is a compact data structure used on distributed applications [13], [14]. The Bloom filter is composed of an m-bit vector that represents a set $A = \{a_1, a_2, a_3, \ldots, a_n\}$

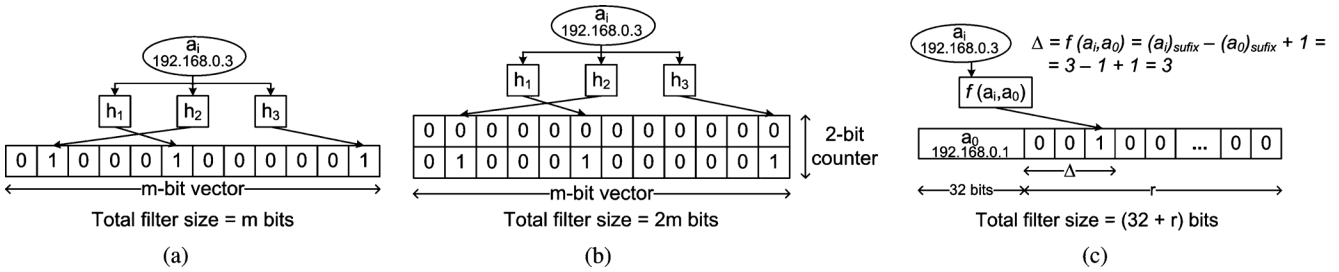


Fig. 1. Insertion procedure of the address element $a_i = 192.168.0.3$ in the filters used with FAP. For the sequence filter, the address range, whose size is r, goes from $a_0 = 192.168.0.1$ to $a_{r-1} = 192.168.0.254$ with a $192.168.0.0/24$ subnetwork. (a) Bloom filter with $k = 3$ hash functions and $m = 12$ bits of filter size. (b) 2-bit Counter Bloom filter with $k = 3$ hash functions and $2m$ bits of filter size. (c) Sequence filter, assuming an address range of $r = 254$ addresses.

composed of n elements. The elements are inserted into the filter through a set of independent hash functions, $h_1, h_2, \ldots, h_k$, whose outputs are uniformly distributed over the m bits. First, all the bits of the vector are set to zero. After that, each element $a_i \in A$ is hashed by each of the k hash functions, whose output represents a position to be set as 1 on the m-bit vector, as shown in Fig. 1(a). To verify if an element a_j belongs to A, we check whether the bits of the vector corresponding to the positions $h_1(a_j), h_2(a_j), \ldots, h_k(a_j)$ are all set to 1. If at least one bit is set to 0, then a_j is not on the filter. Otherwise, it is assumed that the element belongs to A. There is, however, a false-positive probability that an element $a_j \notin A$ be recognized as being in A. This may happen when the bits at the positions $h_1(a_j), h_2(a_j), \ldots, h_k(a_j)$ are all set by previously inserted elements.

Broder and Mitzenmacher show that the probability that a specific bit in a k hash function Bloom filter is still 0 after n element insertions is $P0 = (1 - 1/m)^{(k.n)}$ [15]. The false positive probability is the probability that all the k selected bits of a noninserted element are set. Hence, the false-positive probability (P_{fp}) is given by the probability of these k bits not being 0, which means that

$$P_{\text{fp}} = (1 - P_0)^k. \tag{1}$$

Then, (1) shows that the false-positive probability decreases when the number of elements n of set A is decreased or the size of the filter, m, is increased. Moreover, by equating to zero the derivative of (1), we obtain the value of k that minimizes the false-positive probability, which is given by $k = \left\lceil \frac{m \cdot \ln 2}{n} \right\rceil$.

When the removal of elements from the filter is required, each bit of the filter is replaced by a counter, as shown in Fig. 1(b). Each counter c_i indicates the number of times each position was set. To avoid false negatives, it is necessary to guarantee that the counters do not overflow. Fan *et al.* shows that, using 4-bit counters and k that minimizes the false-positive probability, the overflow probability does not depend on the number of counter cells in the filter, m, or the number of elements n, and is considered negligible [14].

B. Sequence Filters

The other filter structure that we propose is called Sequence filter, and it stores and compacts addresses based on the sequence of addresses. This filter is created by the concatenation of the first address of the address sequence, which we call initial element (a_0), with an r-bit vector, where r is the address range size. In this filter, each address suffix is represented by one bit, indexed by Δ, which gives the distance between the initial element suffix $(a_{0_{sufix}})$ and the current element suffix $(a_{i_{sufix}})$. If a bit is in 1, then the address with the given suffix is considered as inserted into the filter; otherwise, the bit in 0 indicates that the address does not belong to the filter. Therefore, there are neither false positives nor false negatives in the Sequence filter because each available address is deterministically represented by its respective bit. The Sequence filter and the procedure to insert an element into the filter are illustrated in Fig. 1(c).

C. Filter Selection

The best filter for FAP depends on network characteristics such as the estimated number of nodes in the network and the number of available addresses. It also depends on the false-positive and false-negative rates of the filter. Bloom filters do not present false negatives, which means that a membership test of an element that was inserted into the filter is always positive.[1] These filters, however, present a false-positive probability. Hence, a membership test of an element that was not inserted into the Bloom filter may be positive. If we choose a false-positive probability of ≈ 0.06, assuming that $m \gg k$ and a 4-bit counter to avoid buffer overflow, then the ratio between the number of cells in the filter (m) and the maximum number of inserted elements (n) is $m/n = 6$, and the ideal number of hash functions is 4, according to (1). Hence, the size s of the Bloom filter with 4-bit counters $(b_c = 4)$ is $s = (m/n) \cdot n \cdot b_c = 6 \cdot n \cdot 4$. Therefore, the size of the Bloom filter is not determined by the address range, but by the maximum number of elements to be inserted into the filter, which is the upper-bound estimate of the number of active nodes in the network. On the other hand, the Sequence filter is deterministic and, as a consequence, neither false positives nor false negatives are created. The size of the Sequence filter depends only on the size of the address range, r, and on the address size A_s. The address range size is defined by the number of bits in address suffix, b, so that $r = 2^b$. Hence, the number of bits in the Sequence filter is given by $s = A_s + r$. Fig. 2 shows the size of both filters, assuming 32-bit addresses, a false-positive probability of 0.06 for the Bloom filter, and the use of an ideal k, which means that $m/n = 6$ and $k = 4$. Fig. 2(a) and (b) shows that the Bloom filter size is constant to the address range size and grows with the number of elements,

[1]We assume a choice of parameters that guarantees that there is no buffer overflow in the Counter Bloom Filter.

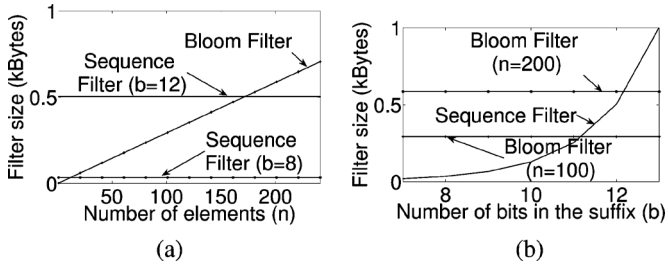


Fig. 2. Required filter size according to available addresses and the estimate number of nodes in the network. (a) According to the number of nodes. (b) According to the address suffix.

whereas the Sequence filter follows the opposite: The filter size is constant to the number of elements and increases with the address range size. As a result, the Bloom filter is more suitable for an extensive address range, whereas the Sequence filter is more adequate to a large number of elements.

D. Procedures of FAP

1) Network Initialization: The network initialization procedure deals with the autoconfiguration of the initial set of nodes. Two different scenarios can happen at the initialization: the joining nodes arrive one after the other with a long enough interval between them, called gradual initialization, or all the nodes arrive at the same time, called abrupt initialization. Most protocols assume the gradual scenario with a large time interval between the arrival of the first and the second joining nodes. For example, the protocol proposed by Fan and Subramani [5] assumes that the first node is alone to choose a partition identifier. Then, the following joining nodes are handled by the first node through the joining node procedure. If all nodes join the network approximately at the same time, each node will choose a different partition identifier. This triggers many partition merging procedures simultaneously, which creates a high control load and can cause inconsistencies in the address allocation procedure, generating address collisions. We argue that address allocation protocols must operate without any restriction to the way the nodes join the network. Our filter-based proposal fits well for both gradual and abrupt initialization scenarios, using Hello and AREQ messages, shown in Fig. 3(a) and (b). The Hello message is used by a node to advertise its current association status and partition identifier. The AREQ message is used to advertise that a previously available address is now allocated. Each AREQ has an identifier number, which is used to differentiate AREQ messages generated by different nodes, but with the same address.

In FAP, a node trying to join the network listens to the medium for a period T_L. If the node does not receive a Hello message within this period, then it starts the network, acting as the initiator node. An initiator node may start the network alone, or with other initiator nodes. Otherwise, if the node receives a Hello message, then the network already exists and the node acts as a joining node.

An initiator node randomly chooses an address, considering the address range defined by the bits of the network prefix, creates an empty address filter, and starts the network initialization phase. In this phase, the node floods the network N_F times with AREQ messages to increase the probability that all initiator nodes receive the AREQ message. If there are other initiator nodes, they also send their AREQ N_F times, advertising their randomly chosen addresses. After waiting a period T_W without listening to AREQs from other initiator nodes, in case they exist, the node leaves the initialization phase and inserts on the address filter all the addresses received with AREQs. After that, the node starts to send Hello messages with the address filter signature, which is a hash of the filter. This signature identifies the network and is used to detect partitions, in case they occur. If the initiator node receives any AREQ with the same address that it has chosen, but with a different identifier number, which means that there is an address collision, the node waits for a period T_C and then chooses another available address and sends another AREQ. During the period T_C, the node receives more AREQs with other already allocated addresses. Therefore, after T_C, the node knows a more complete list of allocated address, which decreases the probability of choosing a used address. Hence, the period T_C decreases the probability of collisions and, consequently, reduces network control load.

After the initialization phase of FAP, all initiator nodes have chosen a unique address due to the random address choice and the validation using AREQ messages with identifier numbers. Additionally, every node knows all currently allocated addresses with a high probability due to the N_F times flooding the network. Consequently, each node also creates an address filter containing all the allocated addresses.

2) Node Ingress and Network Merging Events: After the initialization, each node starts broadcasting periodic Hello messages containing its address filter signature. Upon the reception of a Hello, neighbors evaluate whether the signature in the message is the same as its own signature to detect merging events. Only the nodes that have already joined the network are able to send Hello messages, receive a request of a node to join the network, and detect merging events.

The node ingress procedure is described in Fig. 4(a). When a node turns on, it listens to the medium for a period T_L. If the node listens to a Hello, there is at least one node with an address filter, and the network already exists. Hence, the node knows that it is a joining node instead of an initiator node. The joining node then asks for the source of the first listened Hello message (the host node) to send the address filter of the network using an Address Filter (AF) message, shown in Fig. 3(c). When the host node receives the AF, it checks bit I, which indicates whether the message is being used for a node-joining procedure or a partition-merging procedure. If $I = 1$, the message came from a joining node. Then, the host node answers the request with another AF with bit R set to 1, indicating that the AF is an answer to a previous filter request. When the joining node receives the AF reply message, it stores the address filter, chooses a random available address, and floods the network with an AREQ to allocate the new address. When the other nodes receive the AREQ, they insert the new address in their filters and update their filter signatures with the hash of the updated filter.

Merging events are also detected based on Hello and AF messages, as described in Fig. 4(b). Nodes in different partitions choose their address based only on the set of addresses of their partition. Hence, nodes in different partitions can select the same address, which may cause collisions after the partitions merged. In FAP, when a node receives a Hello, it checks

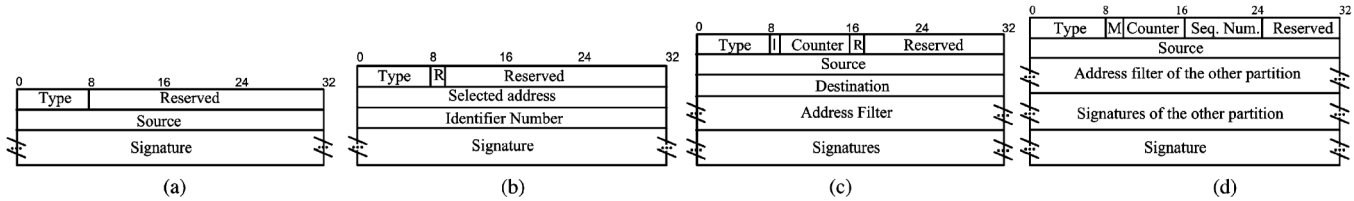


Fig. 3. Messages of FAP for initialization, joining node, and partition merging procedures. (a) Hello. (b) AREQ. (c) AF. (d) Partition.

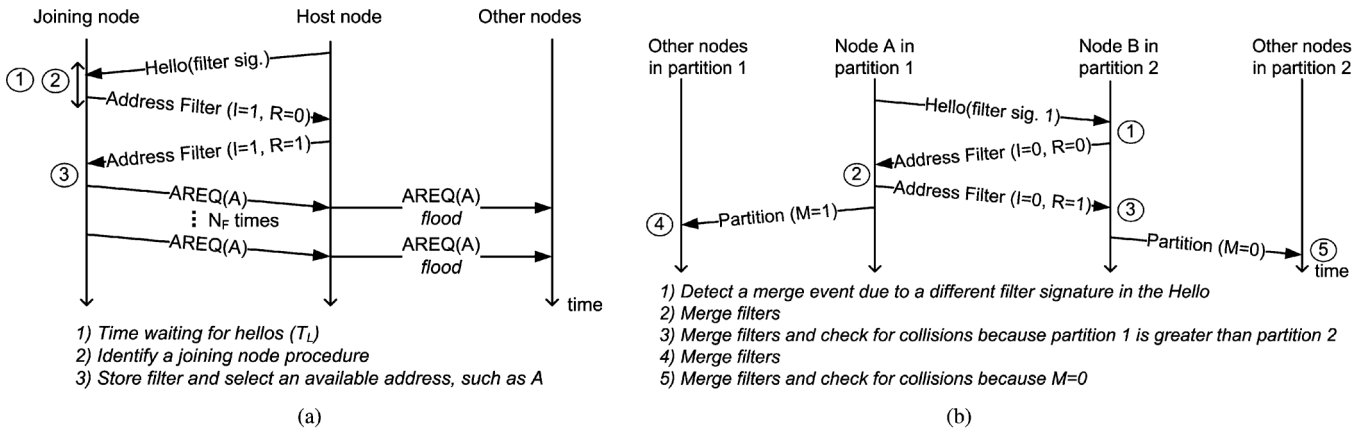


Fig. 4. FAP schemes for node ingress and network merge. (a) Node ingress. (b) Partition detection and merge.

whether the filter signature on the message is different than its current signature. If so, the node knows that they have different sets of allocated addresses. If there is more than T_P s since the last merging event, a new merging procedure is started. In this procedure, both nodes exchange AF messages to disseminate the filters of the two partitions. First, each node checks whether its address is equal or greater than the address of the other node. The node with the greatest address, or both nodes, in case addresses are equal, starts the process. The node starting the process sends an AF message with its current address filter to the other node, which stores the received filter and sends back an AF message with the filter of its partition. Then, both nodes flood their partitions with a Partition message, shown in Fig. 3(d), so that all nodes update their filters with the other partition data. Upon reception of the Partition message, each node must check the bit M on the Partition message to verify if it is on the lowest priority partition. The lowest priority partition $(M = 0)$ is selected as the smallest partition or, if both partitions are of the same size, it is selected as the partition of the node that started the process. Each node on the lowest-priority partition must check whether its address is on the other partition filter to detect collisions. If there is a collision, the node randomly chooses an available address in both filters and floods the network with an AREQ to allocate the new address. If the node receives an AREQ with the same address that it has chosen, but with a different sequence number, it chooses another address because another node has also chosen the same address. Finally, all the nodes merge the other partition filter with its own filter, insert the addresses received in the AREQs into the new filter, and update the filter signature.

FAP requires a mechanism to synchronize the filter signature update after a node ingress or a merging event to avoid false positives in network merging detections. For instance, let us assume a scenario with $N_F = 3$ in which a node, called node A, lost the first and the second AREQ transmissions due to collisions, but a neighbor of node A was able to receive the first transmission of the message. If the neighbor that received the message immediately updates its filter signature, node A may receive a Hello message with a different filter signature. Then, node A would incorrectly identify a network merging event, which characterizes a false positive, generating a high message overhead. The proposed mechanism to update filter signatures relies on the storage of extra signatures for a short period. In the proposed mechanism, instead of immediately updating the filter signature, the neighbor generates the new signature and stores it for a period T_S. The period T_S must be long enough to guarantee that all retransmissions of AREQ messages were already flooded. Hence, during T_S, node A receives the AREQ, generates the new signature, and stores it. Then, no merging event will be noticed by node A. After T_S, the neighbor stores the old signature for more T_S seconds and starts using the new signature. To avoid false positives, all the stored signatures are accepted as valid by a node. Hence, when the neighbor updates its signature, node A will notice that the new signature used by the neighbor is stored and is valid, as well as the neighbor will accept the old signature in the Hellos of node A because this signature is also stored. Hence, the new signature is used after T_S and old signature is discarded only after $2 \cdot T_S$ to guarantee that no node updates its filter signature until all the nodes have been notified about the new event, as well as no node discards the old filter signature until all the nodes have updated their filter signatures. With this procedure, each node can correctly validate its filter during the processes that change the address filter content. If the signature in the Hello is different of its current filter signature, but equal to some stored signature, the node does not consider that the neighbor is in another partition.

The AF message has a special field called *Signatures*, which contains all the address filter signatures acceptable by the node sending a message. After receiving the AF message, a node stores the received signatures for a period $T_{S'}$ to maintain compatibility with the nodes of the other partition.

3) Node Departure: When a node leaves the network, its address should become available for the other nodes. If the departing node is correctly shut down, it floods the network with a notification to remove its address from the address filter. If the departing node does not notify the network, the address remains allocated in the filters, which can make the available addresses scarce with time. This can be identified in the address filter by the fraction of bits set to 1 in the Bloom and in the Sequence filter and by the fraction of counters greater than one in the Counter Bloom Filter. Therefore, every node verifies this fraction in their address filters every time the filter is updated. If this fraction reaches a threshold that indicates that the filter is full or almost full, all the nodes reset their address filters and returns to the network initialization. Instead of choosing a new address, the node uses its current address, which is not collided, to reduce message overhead and to avoid breaking active data connections. The AREQ messages used during filter renews present the bit $R = 1$ to indicate that all nodes must restart their filters. Although the initialization phase seems to overcharge the network, it is equivalent in terms of control load to a partition merging in DAD-based autoconfiguration protocols, such as the protocol proposed by Fan and Subramani [5]. To avoid frequent address filter renews in networks with filters fulfilled due to a high occupation, there is a minimum period between filter renews, defined as T_M.

A node can leave the network during any procedure. FAP uses timers to avoid that a leaving node causes inconsistencies in the address filter. Hence, if a host node leaves the network during a joining node procedure, the joining node will notice this event after T_F s without an answer from the host and will search for a new host node. After N_T times trying to find a host node, the joining node considers it is alone and starts the initialization procedure. A similar mechanism is also used to avoid inconsistencies during the partition merging procedure. In the initialization procedure, all nodes have equal roles, and then, if an initiator node fails, it does not stop or disrupt the initialization procedure. All the other initiator nodes follow with the address allocation without noticing the node failure.

IV. Analytical Results

A. Probability of Collisions in FAP

We analyzed FAP to evaluate the probability that our scheme causes an address collision. A collision occurs when two different joining nodes generate AREQs with the same address and the same identifier number or if two disjoint partitions own exactly the same filters. In the first case, the joining nodes do not notice that their addresses are the same because the message from the other node seems to the first node like a retransmission of its own message. In the second case, the partition merging procedure is not started because the signatures of the Hellos are the same for both the partitions, and, consequently, the network would have a collision for each of its addresses.

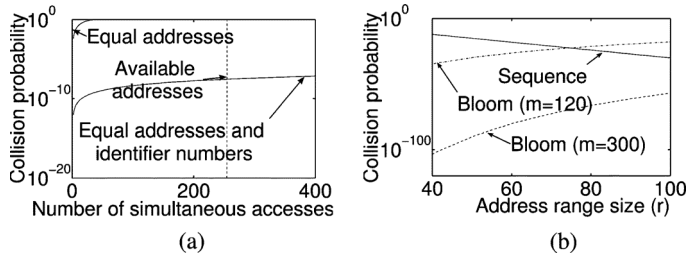


Fig. 5. Analysis of collision probability in FAP. (a) Probability of AREQ collisions for n simultaneous accesses ($r = 256$). (b) Probability of address filter collision during partition merging events.

Aside from these two cases, nodes joining the network never cause address collisions. A joining node always sends an AREQ and, when any node that has already advertised the address receives the AREQ, it must check for a collision, regardless of its current state. Assuming there is no malicious behavior in the network, this situation occurs only in the initialization or when two nodes join the network at approximately the same time because both nodes could choose the same available address in the filter. Therefore, if two joining nodes receive the address filter roughly at the same time and choose the same address, both nodes will change their addresses because each node will receive an AREQ with the selected address from the other node. In case the AREQ of one of the joining nodes is lost due to link losses and collisions, only one node will change the address, solving the collisions. The case in which both AREQs are lost could cause a collision, but the probability of this event is negligible, especially due to the use of N_F repetitions of a flooding message. Other message loss events also do not cause collisions in FAP because they are compensated by false merging procedures. If a node does not receive any of the transmissions of an AREQ, its filter will be different from the filter of the other nodes. This causes false merging procedures until all the nodes present the same information in their filters.

The probability that two nodes choose the same address and the same identifier number for an AREQ, causing a collision, P_A, can be derived by considering the birthday paradox[2] [7], with e_s being the space size of the concatenation of the address with the identifier number, and n the number of nodes that are trying to access the network at approximately the same time, which means a set of n initiator nodes or a set of n joining nodes that search for an address approximately at the same time. Hence

$$P_A = 1 - \left[\frac{e_s - 1}{e_s} \cdot \frac{e_s - 2}{e_s} \cdots \frac{e_s - n + 1}{e_s}\right]. \qquad (2)$$

Fig. 5(a) shows the collision probability, P_A, considering an address range of 256 entries. We observe that there is a high probability of address collision, but the AREQ collision probability is negligible, even for a number of nodes greater than the number of available addresses, due to the use of the identifiers.

The probability that two different partitions have the same address filter is derived for both the Bloom filter (P_{BF}) and the Sequence filter (P_{SF}). Let A_1 be the set of addresses of the first

[2]The birthday paradox calculates the probability that there are at least two people with the same birthday in a set of randomly chosen people. In this problem, the event space size e_s is 365, representing the days in a year, and the number of events n is the number of people. The probability is given by $1 - n!\binom{e_s}{n}/{e_s}^n$.

partition, A_2 be the set of addresses of the second partition, and R be set of available addresses. Hence, $A_1 \subset R$, $A_2 \subset R$, and $|A_1| + |A_2| \leq |R|$ to guarantee that there are enough addresses for all nodes in the network. Also, we assume that all the pairs $(|A_1|, |A_2|)$ are equally probable, since the size of each partition depends only on the network topology. For the Sequence filter, P_{SF} is obtained dividing the number of cases in which two filters have the same inserted addresses by all the possible configurations of two filters. The number of cases of two filters with the same inserted addresses is obtained by counting all possible combinations of sets of up to $\lfloor \frac{r}{2} \rfloor$ elements, where $|R| = r$ is the number of addresses in the available address range. The total number of pairs of sets is computed by all the possible combinations of elements in the first set multiplied by all the possible combinations of elements in the second set, with the constraint that the sum of the sizes of the two sets is equal or smaller than r. Therefore, P_{SF} is given by

$$P_{\mathrm{SF}} = \frac{\sum_{i=1}^{\lfloor \frac{r}{2} \rfloor} \binom{r}{i}}{\sum_{j=1}^{r-1} \left(\binom{r}{j} \cdot \sum_{g=1}^{r-j} \binom{r}{g} \right)}. \tag{3}$$

For the Bloom filter, the probability of filter collision also depends on the false positives because the representation of a set is equal to the representation of a different set that contains the same elements of the first set plus its false positives. The filter collision probability is greater for a Bloom Filter than for a Counter Bloom Filter. Some filter collisions caused by false positives in the Bloom Filter will not occur in the Counter Bloom Filter because the counters can differentiate the filters. For instance, if a Bloom Filter contains elements "a," "b," and "c," and also checks positive for element "d," then another Bloom Filter containing "a", "b," "c," and "d" will have the same representation as the first filter. In the case of Counter Bloom Filters, the insertion of element "d" changes the counters in the second filter, thus the first and the second filters have different representations, even though the false positive still exists in the first filter. Hence, the probability of two different partitions having the same Bloom filter (P_{BF}) is an upper bound for the case of Bloom filters with counters.

We calculated P_{BF} dividing all the possible cases of filter collision, weighted by the probability of that filter configuration happens, by all the possible partition configurations, which means all the possible partition size pairs $(|A_1|, |A_2|)$. Hence, by the probability that two Bloom filters have the same bits in 1 and in 0, which is given by

$$P_{\mathrm{BF_{col}}} = \sum_{i=0}^{m} \binom{m}{i} \left[P_0^i P_1^{(m-i)} \right] \left[P_0'^i P_1'^{(m-i)} \right] \tag{4}$$

assuming that the size of the first partition is $|A_1| = n$, the size of the second partition is $|A_2| = n'$, and that

$$P_{\mathrm{total}} = \left[\sum_{j=0}^{m} \binom{m}{j} P_0^j P_1^{m-j} \right] \left[\sum_{g=0}^{m} \binom{m}{g} P'^g_0 P'^{m-g}_1 \right] = 1 \tag{5}$$

we obtain the probability of filter collision for any partition size, which is given by

$$P_{\mathrm{BF}} = \frac{\sum_{n=1}^{r-1} \sum_{n'=1}^{r-n} P_{\mathrm{BF_{col}}}}{\sum_{n=1}^{r-1} \sum_{n'=1}^{r-n} P_{\mathrm{total}}} \tag{6}$$

where P_0 and P_0' are respectively the probability of a bit being equal to 0 after n and n' insertions into the filter, $P_1 = 1 - P_0$, $P_1' = 1 - P_0'$, r is the address range size, m is the filter size, and k is the number of hash functions.

Fig. 5(b) plots the expressions of P_{SF} (3) and P_{BF} (6) and shows that these probabilities are negligible in an address range with size $40 \leq r \leq 100$. The P_{SF} decreases when the address range size increases because the greater the address range, the greater the size of the filter. The P_{BF} increased with r because we assumed a fixed-size Bloom filter and, consequently, as we insert more elements, the filter gets overloaded with a higher false-positive rate. Assuming $r = 100$, which means that the number of elements in both filters varies from 2 to 100, and a Bloom Filter with size 120, we obtain a collision probability smaller than 10^{-10}. Indeed, the greater m, the lower the probability that there is a filter collision when using Bloom Filters or Counter Bloom Filters because the greater the m, the lower the false-positive probability. Therefore, FAP is not prone to errors due to AREQs collisions or to address filter collisions in initialization, joining node, and partition merging procedures.

B. Control Overhead Estimate

The main procedures in addressing protocols are network initialization, node joining/leaving, and merging. Usually, these procedures, as well as the ordinary protocol operation, generate control overhead, reducing the available bandwidth. We estimate the number of control messages sent by all these procedures for FAP, the extension of DAD [4] proposed by Fan and Subramani [5], hereafter called DAD with partition detection (DAD-PD), and MANETconf [6] (Mconf). DAD-PD uses partition identifiers, which are numbers shared by the nodes in the same partition to make it possible to distinguish the current partition from the others. Every time a node joins the network or a node observes that it lost a neighbor, the partition identifier of the whole network is changed. We also compare our protocol with MANETconf, which is based on the knowledge of the Allocated list, which describes the allocated addresses, and the Allocated Pending list, which describes the list of the addresses under evaluation to be allocated to joining nodes.

Table I shows an estimate of the number of messages sent by each protocol in the optimal case for node joining, partition merging, and initialization procedures. As optimal case, we mean the case in which there are no message losses and only one procedure is being executed at a time. Table II describes the variables used in this analysis.

As shown on Table I, the overhead of DAD-PD in a node joining procedure depends on the number of address collisions, C. Indeed, this protocol does not store the list of used addresses, which means that the joining node randomly chooses a new address that may already be allocated in the network.

TABLE I
NUMBER OF CONTROL MESSAGES ASSUMING NO MESSAGE LOSSES AND THAT ONLY ONE PROCEDURE IS RUNNING AT A TIME

Protocol	Joining Node			Partition Merging			Initialization		
	Flood	Unicast	Broadcast	Flood (F_i)	Unicast (U_i)	Broadcast	Flood	Unicast	Broadcast
FAP	N_F	1	1	$N_F(2+C)$	2	0	$N_F(N+2C)$	0	0
DAD-PD	$N_F(1+C)$	$N_F \cdot C$	0	$N_F(N_P+2C)$	$2C$	0	$\sum_{i=1}^{(N-1)} F_i$	$\sum_{i=1}^{(N-1)} U_i$	$N_T N$
Mconf	2	$N-1$	$3+N_N$	$2+4C$	$4+C(N_P-1)$	$3C$	$\sum_{i=1}^{(N-1)} F_i$	$\sum_{i=1}^{(N-1)} U_i$	$N_T N$

TABLE II
NOTATION USED FOR OVERHEAD ESTIMATE

Variable	Description
N_F	Number of flooding message transmissions by a source node in each flood event
C	Number of address collisions
N	Number of nodes in the network
N_P	Number of nodes in both network partitions
F_i	Number of flooding messages in the partition merging event i
U_i	Number of unicasted messages in the partition merging event i
N_T	Number of tries before concluding that the node is alone
N_N	Number of neighbors of the joining node

FAP performs better than DAD-PD when one or more address collisions, $C \geq 1$, occur because this increases the number of floods only in DAD-PD and floods are the most costly operation. The value of C depends on the ratio between the number of nodes in the network, N, and the number of available addresses, r. When $N/r \rightarrow 1$, the probability of address collisions increases and, consequently, DAD-PD performance decreases. MANETconf generally has a larger overhead than FAP because MANETconf is based on the assumption that all nodes must agree before allocating an address, which demands many control messages. Thus, the joining node exchanges three messages through broadcasts with the node that will allocate the address, which is called initiator in MANETconf. The initiator, then, floods the network asking whether all nodes agree that the chosen address is available. If all the other nodes agree, then the initiator floods the network again to allocate the chosen address. Besides the overhead caused by all the flood events, depending on the routing protocol, each unicasted message flow can imply in a flood to search for a route between the source and the destination node. Hence, an intensive use of unicasts to different destinations, such as in MANETconf, can generate a high control overhead.

Table I also shows that FAP has a low-cost partition merging procedure because the filters in FAP identify the collided addresses during a partition merging. In FAP, the two nodes that detected the merging exchange their filters and then flood the other partition filter. Afterwards, only one flood is requested per collision, to advertise the new allocated addresses. In MANETconf, the nodes that detected the partition also exchange and flood lists like in FAP, but the collisions are treated in a similar way to the joining node mechanism, which implies a high message overhead. In DAD-PD, all nodes flood the network after a partition merging detection, and each collision will involve the unicast of two AREPs and the flood of two AREQs, generated by the nodes involved in the collision.

In the initialization, FAP presents a high probability of having the smallest overhead because it is similar to the procedure proposed by DAD. DAD-PD and MANETconf have a larger control load because they specify only a gradual initialization procedure. If several nodes join the network at approximately the same time, each node believes it is alone, and randomly chooses an address and a partition identifier. After this, the nodes notice the presence of neighbors with different partition identifiers and start partition merging procedures. Since all nodes must merge their partitions to create only one partition representing the whole network, they do at least $N-1$ partition merging procedures. Because the number of flood and unicast messages in a partition merging event of DAD-PD and MANETconf depends on the number of nodes in both partitions (N_P), each partition merging event i during network initialization will present a different number of flooding (F_i) and unicast (U_i) messages per partition merging event.

After the initialization, each node starts the common protocol operation. Each node in FAP, DAD-PD, and MANETconf periodically broadcasts Hello messages. In addition, MANETconf is also based on periodic floods to maintain the partition identifiers. Hence, MANETconf is the most consuming protocol during ordinary operation.

The node leaving procedure is only needed by FAP and MANETconf because they maintain state about the allocated addresses. In the node joining procedure of MANETconf, the node that chooses the address for the joining node, called initiator, contacts all the other network nodes to guarantee that each node agrees that the new address can be allocated. After N_{MR} times trying to contact a specific node without answer, the initiator concludes that the node is absent and floods the network with this information. In FAP, the addresses allocated for nodes that have already left the network are released only when the address filter is almost full. In this situation, the filter is cleared and an initialization procedure is started. Therefore, the node leaving procedure of FAP is costly, but it is rarely evoked, which means that this procedure has a low impact over the whole FAP overhead.

V. SIMULATION RESULTS

We implemented FAP in the Network Simulator-2 (NS-2) and evaluated it considering the Shadowing model for radio propagation and the NS-2 IEEE 802.11 model for the Medium Access Control. These models account for creating a scenario similar to a real community network, using parameters of commercial equipments. Therefore, the parameters used for our simulations are: an average transmission range of 18.5 m, a maximum carrier sense range of 108 m, and a density of 0.0121 nodes/m^2 [16]. We measured the control traffic, the

TABLE III
PARAMETERS OF FAP (F), DAD-PD (PD), DAD (D), AND MCONF (M)

Variable	Description	Value	Protocol
T_L	Max. time listening to the medium	1.0 s	F, PD
T_P	Partition merging min. interval	3.0 s	F, PD
T_W	AREQ/AREP waiting time	1.2 s	F, PD, D
T_R	Message replication interval	0.3 s	F, PD, D
T_H	Hello Timer	1.0 s	F, PD, M
T_C	Waiting time before changing address	0.3 s	F
T_S	Generated-filter-signature storage time	0.5 s	F
$T_{S'}$	Received-filter-signature storage time	3.0 s	F
T_M	Min. interval between filter renews	5.0 s	F
T_F	Max. time waiting for a filter	0.5 s	F
T_{MA}	Address Allocation Timer	0.7 s	M
T_{MR}	Request Reply Timer	0.3 s	M
T_{MN}	Neighbor Reply Timer	0.35 s	M
T_{MAP}	Allocation Pending Timer	1.0 s	M
T_{MP}	Partition Timer	2.0 s	M
N_F	Transmissions of a flooding message	2	F, PD, D
N_T	Neighbor Reply Threshold	3	F, M
N_{MR}	Request Reply Retry	2	M
N_{MI}	Initiator Request Retry	2	M
N_{MP}	Partition Identifier Threshold	3	M

delays, and the number of address collisions in FAP, considering a confidence level of 95% in the results. We also implemented in NS-2 the addressing protocols proposed by Perkins *et al.* [4], called DAD, by Fan and Subramani [5], which we call DAD-PD, and by Nesargi and Prakash [6], called MANETconf and indicated in the results by Mconf.[3] Although DAD does not work in partition-prone environments, we evaluated this protocol because it is a simple proposal with low overhead. Our main objective is to show that our proposal also presents a low overhead and works in any scenario.

Comparing FAP to DAD-PD, we observe the performance impact of the use of the hash of the address filters instead of arbitrated partition identifiers to detect partition merging events. In the original DAD-PD, however, the new partition identifier after a partition merging is given by the sum of partition identifiers, which causes instability in the protocol. Therefore, we improved the protocol performance by choosing the greatest partition identifier in network merging events instead of summing them, which reduces the number of false partition merging detections. In addition, we compared FAP to MANETconf because both proposals use an allocated address list.

The protocol parameters are shown in Table III. These parameters were chosen based on experiments to increase all the four protocols performance and also on recommendations from the authors of the other proposals. Hence, we selected these values focusing on reducing the delays and the overhead while still avoiding instabilities in the simulated scenario.

Specifically, T_L specifies the time listening to the medium before a node decide if it is alone or not. Hence, this period must be, at least, equal to the Hello Timer (T_H). For a better performance, the Hello message required by FAP should be appended to the Hello message of routing protocols. For this reason, we use in FAP the same interval that is usually recommended for Hello messages in routing protocols.

The minimal interval between partition merging events, T_P, avoids high overheads in FAP in environments prone to high forwarding delays and/or many packet losses. This value is even more important for the DAD-PD protocol, in which partition merging mechanisms are frequently called. We evaluate this parameter choice throw simulations. Moreover, both the parameter T_R, which specifies the interval among retransmissions of flooding messages, and T_C, which reduces address collisions during the initialization phase, impacts on FAP performance and are also evaluated in the following simulations.

The parameter T_W is used during FAP initialization to specify when a node is allowed to use its chosen address. Hence, this interval specifies the period that a node should wait for more AREQs before concluding that the initialization phase is ended. T_W does not interfere in the protocol stabilization delay. The values T_S and $T_{S'}$ avoid wrong detections of partition merging events during new node and partition merging events. The use of high values temporarily increases the storage overhead. The minimal interval between filter renews, T_M, impacts FAP overhead only if the network is full. This interval defines the frequency in which the filter is checked to discover if any node has left the network to provide addresses to joining nodes. A small T_M implies on a frequent verification, which increases the overhead, while a high T_M implies in low overhead, but long wait for new nodes to join an almost full network. The value T_F controls the time a joining node waits until the selected neighbor sends the current address filter. This timer only provides resilience to malfunctioning nodes or to neighbors that leave the transmission range of the joining node, and then it usually does not influence on protocol performance. The number of transmissions of a flooding message, N_F, also impacts FAP performance and is evaluated through simulations. Finally, the last FAP parameter, N_T, has a resilience function similar to T_F and presents a small impact over FAP performance.

Both FAP and DAD-PD use equal-sized Hello messages because we assume that both partition identifier and filter signature are composed of 4 B. We assume an address range of 150 addresses and a network with a maximum of 100 nodes to guarantee that the address range is not a constraint that can cause instabilities for any protocol. According to these parameters, we use a Sequence Filter of 23 B.

We first analyze the impact of one node joining the network. We consider a rectangular space with nodes distributed in grid. We measure the control load after the last node joins the network and the required delay to obtain an address, as shown in Fig. 6. In the results, we observe that our proposal, FAP, presents an overhead larger than DAD because our protocol uses Hello messages to detect partitions. DAD-PD presents the greatest control overhead for more than 36 nodes because unnecessary partition merging procedures are started due to mistakes in partition merging event detection after a node joins the network. Indeed, DAD-PD has a partition detection mechanism that is based on partition identifiers. When a new node joins the network, the network partition identifier must be changed to represent the new set of allocated addresses. The update of this value, however, can cause false partition merging detections which increase the control load overhead. For more than

[3]We compared the equations in Table I and the simulation results for small-sized networks, which present low error rate. The two overhead estimate methods present compatible results.

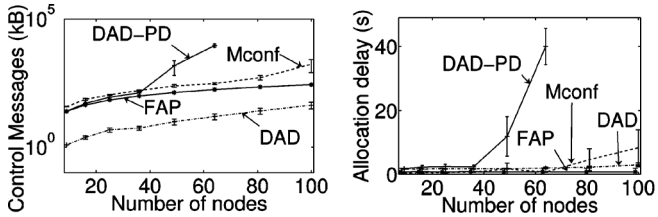


Fig. 6. Impact of a joining node procedure according to the number of nodes, assuming $N_F = 2$. (a) Control load after the joining. (b) Delay for address allocation.

64 nodes, DAD-PD becomes unstable and does not stop until the end of simulation. MANETconf, as expected, also presents a larger control load than FAP, as shown in Section IV-B.

FAP, MANETconf, and DAD present small delays to allocate a new address. Indeed, the number of nodes has almost no influence to the delay of FAP. In this protocol, the joining node obtains the current filter, chooses an available address, and floods the network. Both DAD-PD and MANETconf, however, are influenced by the number of nodes. In DAD-PD, the joining node chooses a new address randomly. When there are more nodes using the address range, the probability of address collision in a random choice of an address increases, and collisions force the joining node to repeat the address allocation mechanism. The delay required by MANETconf also increases with the number of nodes because an address allocation depends on the answer of all the nodes. Concluding, FAP has a low control overhead and a small and constant delay, which shows FAP is more efficient than DAD-PD and MANETconf during the node joining procedure. This is especially important for a growing number of applications in which the contact time is on the order of seconds [17].

In Fig. 7, we evaluate the impact of the network size, the network density, and the number of transmissions of flooding messages in abrupt network initialization, assuming that nodes are distributed in grid. Fig. 7(a) shows that the growth of the number of nodes impacts all the protocols because of the use of floods. DAD-PD, however, suffers a greater influence on the control load than FAP, DAD, and MANETconf because it floods the network more times due to the false positives in the partition merging detection. FAP has a control load smaller than MANETconf and DAD-PD. By Fig. 7(b), we conclude that DAD-PD has the greatest convergence delay and becomes unstable in networks with more than 49 nodes. Also, we observe that the delay of FAP is smaller than the delay of DAD and MANETconf for a high number of nodes. FAP reduces in up to 10% the delay of MANETconf and up to 36% the delay of DAD in a network with 100 nodes. Fig. 7(c) shows that MANETconf is prone to address collisions as we increase the number of nodes. The abrupt initialization in this protocol occurs through simultaneous partition merging procedures, which are not robust and cause address collisions.

We also varied the node density from ≈ 0.064 nodes/m^2, representing high-populated urban scenarios, to ≈ 0.0035 nodes/m^2, representing very low populated rural regions [16]. We used a scenario with 36 nodes to guarantee that all protocols are stable. Fig. 7(d) shows that the density has a small impact over FAP, DAD, and DAD-PD control load. The control load of MANETconf increases with network density because MANETconf to try to compensate the more frequent packet collisions. MANETconf also could not solve all the address collisions, as shown in Fig. 7(e), due to these packet losses.

In Fig. 7(f)–(h), we evaluate the impact of using one, two, or three transmissions of each flooding message (N_F) in the network initialization. We selected a scenario with 100 nodes to simulate a high load scenario. Since MANETconf does not use retransmissions of flooding messages and DAD-PD gets unstable with more than 49 nodes, we evaluate only DAD and FAP. Although DAD sends fewer messages than FAP with one transmission of flooding messages, it is not robust to message losses, as we show in Fig. 7(h). Hence, it is not safe to use DAD with only a single transmission of each flooding message. FAP is robust to message losses with one, two, or three transmissions because if a node does not receive a given address information, its address filter will be different from the filter of the other nodes. Hence, false merging procedures are started until all nodes have the same address filter. With only a single transmission of each message, FAP is able to compensate losses with the false merging procedures, but DAD cannot. That is the reason for the longer average delay of FAP when using one transmission than when using two transmissions, as we see in Fig. 7(g). Hence, we used $N_F = 2$ to reduce the address collision probability.

The next analysis evaluates partition merging events. We vary the number of partition merging events in a static scenario composed of 50 nodes by activating/deactivating some nodes responsible for connecting disjoint partitions. In all the protocols, we started the partition merging events only after the initialization is completed. The clouds that formed each partition have a density of approximately 0.03 nodes/m^2 to guarantee that the density of the whole network is approximately 0.01 nodes/m^2, simulating a typical community network. The nodes that interconnect the partitions are activated in fixed times. Fig. 8(a) shows the control load since the first merging event. DAD has almost no control load because it does not detect partitions. The control load of DAD is only composed of the AREQs sent by the nodes connecting the partitions. DAD-PD has a control load up to 25 times larger than FAP because FAP reduces the number of AREQs during the merging procedure. In FAP, only half of the nodes with collided address choose a new address and send an AREQ, while in DAD-PD, all the nodes should send an AREQ, and each collision implies, at least, one more flood in the network. The number of partitions does not impact the control load of FAP. The number of Address Filter messages increases with the number of partitions, but these messages are sent in unicast between two neighbors and do not impact the control load. Nevertheless, the number of Partition and AREQ messages, which are flooding messages to advertise the filter and to solve collisions, respectively, decreases because the number of nodes in each partition is shorter, reducing the impact of floods. This also explains the decrease in the control load of MANETconf, which is up to 4.1 times larger than the load of FAP.

Fig. 8(b) shows the number of address collisions after the merging events. DAD, as expected, does not resolve any collisions because it was not designed to control partitions merging events. MANETconf presents less address collisions than DAD

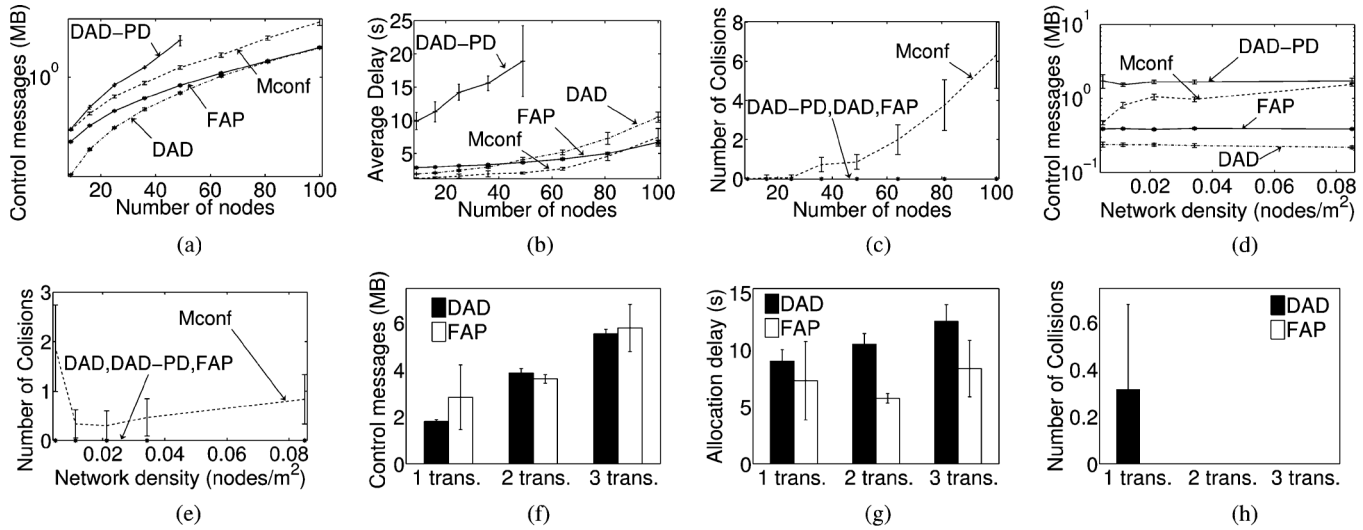


Fig. 7. Impact over the network of the number of nodes, the density, and the number of transmissions of flooding messages (N_F) in a network abruptly initialized. (a) Control overhead according to the number of nodes. (b) Average delay according to the number of nodes. (c) Number of collisions according to the number of nodes. (d) Control overhead according to network density, $N = 36$ nodes. (e) Number of collisions according to network density, $N = 36$ nodes. (f) Control overhead according to N_F, $N = 100$ nodes. (g) Average delay according to N_F, $N = 100$ nodes. (h) Number of collisions according to N_F, $N = 100$ nodes.

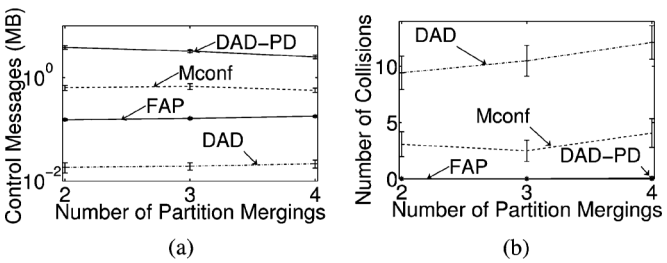


Fig. 8. Impact of merging events in a network with 50 nodes. (a) Control load. (b) Number of address collisions.

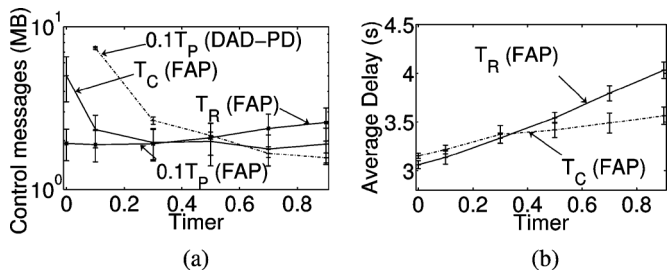


Fig. 9. Analyzing performance according to different values of FAP parameters. (a) Control overhead. (b) Average delay to obtain an address.

because this protocol handles merging events, but it could not avoid all the address collisions. FAP presented no address collision, but DAD-PD presented a small probability of collisions caused by message losses. FAP avoids this kind of collision because of the mechanism that detects message losses and starts false merging procedures until all the nodes have received the same information. As a result, FAP resolves all the collisions and also presented a small control load.

In Fig. 7, we analyzed the impact of the retransmissions of flooding messages in network initialization. Now, we analyze the impact of the delay between flooding message retransmission (T_R), the time waiting before choosing a new address in the initialization (T_C) and the minimal interval between partition merging events (T_P).

First, we analyze the impact of T_C in a network with 100 nodes. We observe in Fig. 9(a) that a greater T_C reduces the overhead because the nodes learn more allocated addresses before choosing a new address. Nevertheless, this also increases the delays in the network, as shown in Fig. 9(b). An intermediate value in this scenario is $T_C = 0.3$ s, which was used in the other simulations. In the same scenario, we analyze T_R and observed that a small T_R reduces both the overhead and the delay. Finally, we observed the impact of T_P. We repeated the partition merging scenario with four partition merging events. In this scenario, $T_P \in [0, 9]$. We observe in Fig. 9(a) that a small T_P is better for FAP, but is very prejudicial for DAD-PD. Indeed, FAP presents a better partition merging event detection than DAD-PD. DAD-PD performs many unnecessary partition merging mechanisms and a greater T_P reduces these false detection events. Nevertheless, a greater T_P also increases the stabilization time of DAD-PD during network initialization. Then, we selected an intermediate value for T_P to balance the requisites of both protocols.

VI. Conclusion

We proposed a distributed and self-managed addressing protocol, called Filter-based Addressing protocol, which fits well for dynamic ad hoc networks with fading channels, frequent partitions, and joining/leaving nodes. Our key idea is to use address filters to avoid address collisions, reduce the control load, and decrease the address allocation delay. We also proposed to use the hash of the filter as the partition identifier, providing an easy and accurate feature for partition detection with a small number of control messages. Moreover, our filter-based protocol increases the protocol robustness to message losses, which is an important issue for ad hoc networks with fading channels and high bit error rates.

The use of the hash of the filter instead of a random number as the partition identifier creates a better representation of the set of nodes. Hence, a change in the set of nodes is automatically reflected in the partition identifier. This identifier is periodically advertised, allowing neighbors to recognize if they belong to different sets of nodes. In the other proposals, a mechanism to change the arbitrated partition identifier is requested, which increases the complexity and the packet overhead of the protocol.

The proposed protocol efficiently resolves all address collisions even during merging events, as showed by simulations. This is achieved because FAP is able to detect all merging events and also because FAP is robust to message losses. FAP initialization procedure is simple and efficient, requiring a control load similar to the control load of DAD, which is a protocol with a small overhead but that does not handle network partitions. Moreover, FAP presents smaller delays in the joining node procedure and on network partition merging events than the other proposals, indicating that the proposed protocol is more suitable for very dynamic environments with frequent partition merging and node joining events.

References

[1] D. O. Cunha, O. C. M. B. Duarte, and G. Pujolle, "A cooperation-aware routing scheme for fast varying fading wireless channels," *IEEE Commun. Lett.*, vol. 12, no. 10, pp. 794–796, Oct. 2008.

[2] N. C. Fernandes, M. D. Moreira, and O. C. M. B. Duarte, "A self-organized mechanism for thwarting malicious access in ad hoc networks," in *Proc. 29th IEEE INFOCOM Miniconf.*, San Diego, CA, Apr. 2010, pp. 1–5.

[3] N. C. Fernandes, M. D. Moreira, and O. C. M. B. Duarte, "An efficient filter-based addressing protocol for autoconfiguration of mobile ad hoc networks," in *Proc. 28th IEEE INFOCOM*, Rio de Janeiro, Brazil, Apr. 2009, pp. 2464–2472.

[4] C. E. Perkins, E. M. Royers, and S. R. Das, "IP address autoconfiguration for ad hoc networks," Internet draft, 2000.

[5] Z. Fan and S. Subramani, "An address autoconfiguration protocol for IPv6 hosts in a mobile ad hoc network," *Comput. Commun.*, vol. 28, no. 4, pp. 339–350, Mar. 2005.

[6] S. Nesargi and R. Prakash, "MANETconf: Configuration of hosts in a mobile ad hoc network," in *Proc. 21st Annu. IEEE INFOCOM*, Jun. 2002, vol. 2, pp. 1059–1068.

[7] B. Parno, A. Perrig, and V. Gligor, "Distributed detection of node replication attacks in sensor networks," in *Proc. IEEE Symp. Security Privacy*, May 2005, pp. 49–63.

[8] S. Thomson and T. Narten, "IPv6 stateless address autoconfiguration," RFC 2462, 1998.

[9] M. Fazio, M. Villari, and A. Puliafito, "IP address autoconfiguration in ad hoc networks: Design, implementation and measurements," *Comput. Netw.*, vol. 50, no. 7, pp. 898–920, 2006.

[10] N. H. Vaidya, "Weak duplicate address detection in mobile ad hoc networks," in *Proc. 3rd ACM MobiHoc*, 2002, pp. 206–216.

[11] H. Kim, S. C. Kim, M. Yu, J. K. Song, and P. Mah, "DAP: Dynamic address assignment protocol in mobile ad-hoc networks," in *Proc. IEEE ISCE*, Jun. 2007, pp. 1–6.

[12] H. Zhou, L. Ni, and M. Mutka, "Prophet address allocation for large scale MANETs," in *Proc. 22nd Annu. IEEE INFOCOM*, Mar. 2003, vol. 2, pp. 1304–1311.

[13] M. D. D. Moreira, R. P. Laufer, P. B. Velloso, and O. C. M. B. Duarte, "Capacity and robustness tradeoffs in Bloom filters for distributed applications," *IEEE Trans. Parallel Distrib. Syst.*, vol. 23, no. 12, pp. 2219–2230, Dec. 2012.

[14] L. Fan, P. Cao, J. Almeida, and A. Z. Broder, "Summary cache: A scalable wide-area web cache sharing protocol," *IEEE/ACM Trans. Netw.*, vol. 8, no. 3, pp. 281–293, Jun. 2000.

[15] A. Broder and M. Mitzenmacher, "Network applications of Bloom filters: A survey," *Internet Math.*, vol. 1, pp. 485–509, 2002.

[16] M. E. M. Campista, I. M. Moraes, P. Esposito, A. Amodei Jr., L. H. M. K. Costa, and O. C. M. B. Duarte, "The ad hoc return channel: A low-cost solution for Brazilian interactive digital TV," *IEEE Commun. Mag.*, vol. 45, no. 1, pp. 136–143, Jan. 2007.

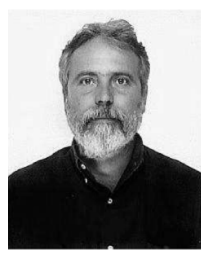
[17] M. G. Rubinstein, F. B. Abdesslem, M. D. De Amorim, S. R. Cavalcanti, R. D. S. Alves, L. H. M. K. Costa, O. C. M. B. Duarte, and M. E. M. Campista, "Measuring the capacity of in-car to in-car vehicular networks," *Commun. Mag.*, vol. 47, no. 11, pp. 128–136, Nov. 2009.

Natalia Castro Fernandes received the electronics and computer engineering degree, M.Sc. degree, and D.Sc. degree in electrical engineering from Universidade Federal do Rio de Janeiro, Rio de Janeiro, Brazil, in 2006, 2008, and 2011, respectively.

After her D.Sc. in 2011, she has received a postdoctoral scholarship from Universidade Federal do Rio de Janeiro. Her major research interests are in ad hoc networks, security, privacy, and future Internet.

Marcelo Duffles Donato Moreira received the electronics and computer engineering degree and M.Sc. degree in electrical engineering from Universidade Federal do Rio de Janeiro (UFRJ), Rio de Janeiro, Brazil, in 2009.

His major research interests are in network security, wireless networks, and future Internet architectures.



Otto Carlos Muniz Bandeira Duarte received the electronic engineer degree and M.Sc. degree in electrical engineering from Universidade Federal do Rio de Janeiro (UFRJ), Rio de Janeiro, Brazil, in 1976 and 1981, respectively, and the Dr.Ing. degree in computer engineering from ENST, Paris, France, in 1985.

Since 1978, he has been a Professor with UFRJ. His major research interests are in security and mobile communications.