An Efficient Filter-based Addressing Protocol for Autoconfiguration of Mobile Ad Hoc Networks

Natalia Castro Fernandes, Marcelo Duffles Donato Moreira, and Otto Carlos Muniz Bandeira Duarte
GTA/COPPE - Universidade Federal do Rio de Janeiro
Rio de Janeiro, Brasil
Email: {natalia,marcelo,otto}@gta.ufrj.br

Abstract—**Address autoconfiguration is an important issue for ad hoc networks in order to provide autonomous networking and self-management. The IP address assignment for ad hoc nodes requires a distributed procedure that resolves all the address collisions in a dynamic network with fading channels, frequent partitions, and joining/leaving nodes. We propose a filter-based addressing protocol for autoconfiguration of mobile ad hoc networks that is lightweight and robust to packet losses. We present a probabilistic analysis of address collisions and discuss filters functionalities in the address autoconfiguration. We evaluate the performance of our protocol for static and mobile scenarios, considering joining nodes and partition mergings. Simulation results show that our protocol resolves all the address collisions and reduces up to 22 times the control traffic when compared to the other addressing protocols.**

I. Introduction

Mobile ad hoc networks are envisioned to have dynamic multi-hop topologies which are likely composed of bandwidth-constrained wireless links. These networks are independent of any fixed infrastructure or centralized administration, which makes them attractive to many applications, such as sensoring, Internet access to deprived communities and disaster recovering. On the other hand, ad hoc network protocols have to deal with limited resources, such as bandwidth and energy, the shared medium, and high bit error rates due to the properties of the wireless channel [1]. Moreover, partitions are also usual in these networks, because network topology frequently changes.

The self-organization nature of ad hoc networks brings new challenges to IP address assignment. If a node lacks an IP address, it becomes non-operational, because the connectivity in current ad hoc networks depends upon using the IP address as the node identifier. In order to support spontaneous networking, the address assignment should be autonomous. Mechanisms like Dynamic Host Configuration Protocol (DHCP) are conflicted with concepts of ad hoc networks, because these networks aim to configure the whole network automatically, without any kind of server and dealing with partition mergings. Therefore, ad hoc networks require a distributed self-configuration mechanism to allocate addresses.

In this paper, we propose a novel and efficient approach called Filter-based Addressing Protocol (FAP). The proposed protocol autoconfigures addresses according to the current set of addresses assigned to network nodes. This set of addresses is represented in a compacted fashion using filters, which assure the univocal configuration of the joining nodes in the network and the detection of address collisions after the merging of several partitions. With the filters, any node can check if an address is available before trying to allocate it. Also, a node can check if the hash of its filter is the same of the hash of the filter of its neighbors, to identify partitions. Therefore, the filters reduce the control overhead and allow an accurate detection of partition mergings. This is important because the higher the control overhead, the less the lifetime of nodes and the less the available bandwidth to send data. The procedures of FAP assure the correct update of states in each node in a distributive way with no address collisions. The analysis and the simulation experiments show that FAP achieves low complexity, low storage, low communication overhead and low latency, resolving address collisions in a timely manner.

The paper is structured as follows. Related research efforts concerning addressing in ad hoc networks are introduced in Section II. The proposed protocol is detailed in Section III and the performance of FAP is analytically evaluated in Section IV. Besides, we performed a simulation described in Section V. Finally, Section VI concludes the paper.

II. Related Work

The absence of servers hinders the use of centralized addressing in ad hoc networks. In distributed addressing schemes, however, it is hard is to avoid duplicated addresses, because a random choice of addresses by each node would cause a high collision probability [2].

The IETF Zeroconf working group proposes a hardware-based addressing [3], which assigns an IP address to a node based on the device MAC address. Nevertheless, this addressing scheme has privacy problems. The use of MAC addresses as IP suffix implies a fixed interface identifier, which makes possible the identification and the correlation of different activities of a given user by a sniffer strategically placed on a link [4]. Besides, if the number of devices is greater than the address space, this proposal cannot be used. We also cannot guarantee that all nodes and interfaces contain unique IEEE MAC identifiers and these identifiers can be easily changed.

Stateless proposals to autoconfigure IP addresses in ad hoc networks are typically based on a distributed protocol called Duplicate Address Detection (DAD) [5]. 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. If another node has already chosen the same address, it sends an Address Reply message (AREP) to the joining node. If 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 handle network partitions and, as a consequence, does not fit well for ad hoc networks. Extensions to DAD were proposed taking into account network partitions, using Hello messages and network identifiers. These identifiers are random numbers that identify each network partition. A group of nodes changes its network identifier whenever it identifies a partition or a merging of the network. The network identifiers are required because nodes cannot change the network prefix when network partitions occur, because nodes are still connected to the same domain, even though they are temporarily disconnected from each other. Therefore, the network identifiers work as partition identifiers. Fan and Subramani propose a protocol based on DAD to detect network mergings whenever a node receives a Hello with a network identifier different from its own identifier or when the neighbor set of any node changes [6]. Fazio *et al.* also propose a protocol based on network identifiers, but their proposal works in a reactive fashion [7]. Instead of sending AREQs and AREPs every time a node joins the network or when a network merging is identified, the protocol only identifies collisions when data exchange is required. Although it reduces the number of control messages, the reactivity of the protocol causes a delay on data transmission. Besides, this protocol only applies for reactive routing protocols, because, in proactive routing protocols, the routes are calculated in advance and the existence of address collisions would cause wrong choice of routes.

Other proposals use routing information to detect collisions [8], [9]. Weak DAD, for instance, proposes to route packets correctly even if there is some address collision. In this protocol, every node is identified by the IP and a key. DAD is realized among neighbors and collisions among the other nodes are identified by information of the routing protocol. If some nodes choose the same IP and key, however, the collision is not detected. Besides, Weak DAD is a specific solution for some routing protocols.

Stateful protocols were proposed to reduce the control load and also to improve the performance on network merging detection or on the address reallocation. In these protocols, nodes store a data structure to accomplish the addressing. In MANETconf [10], nodes store two lists of the addresses: allocated IP list and allocated-pending list. In this protocol, all nodes must approve the address for a joining node before the allocation. Hence, when a joining node asks for an address to another node, this node chooses an available address, stores it on the allocated-pending list, and floods the network. If all network nodes positively answer the request for address allocation, the node notify the address to the joining node, moves the address to the allocated address list, and floods the network again to confirm the address allocation. Because of the query to all network nodes, this protocol handles the address reallocation, but the partition detection depends on periodic flooding. Therefore, this protocol can correctly detect partitions and can reallocate addresses, but it incurs in a large number of control messages and nodes are required to have a high storage capacity.

Another stateful protocol is the Dynamic Address assignment Protocol in mobile ad-hoc networks (DAP) [11], which is based on the allocation of available address sets for each node, on Hellos, and on network 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. The main problem of this proposal is the partition merging procedure, which is based on the network identifiers. These identifiers give no information about the current set of nodes in each partition and, hence, are not suitable partition identifiers. DAP requires the use of DAD in partition mergings not only for the allocated addresses, but also for the available address list stored in each node, increasing the control load. Another problem is that these lists are not always sequential, requiring nodes with high storage capacity.

Prophet [12] proposes an address allocation 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. The protocol generates a low control load, because it does not need any flooding message. Prophet, however, requires an address space much larger than the previous protocols to support the same number of nodes in the network. Besides, it depends on the quality of the pseudo-random generator to avoid duplicated address. Therefore, it needs an algorithm, like DAD, to detect duplicated addresses, which increases the protocol complexity and eliminates the advantage of a low message overhead.

Our proposal aims to reduce the control load and to improve partition merging detections without requiring nodes with a high storage capacity. These objectives are achieved through small filters and an accurate distributed mechanism to update the states in nodes. We use a filter signature, which is a hash of the filter, to identify partitions instead of using network identifiers. Whereas network identifiers are only identifiers randomly chosen by the nodes, the filter signature represents all the nodes in the partition, which improves the capacity to correctly detect partitions in our protocol and, also, increases the robustness to message losses.

III. Filter-based Addressing Protocol (FAP)

The proposed protocol aims to dynamically autoconfigure addresses, identifying and solving addresses collisions with a low control load, even when there are nodes joining the network or partition mergings. To attain all these objectives, the Filter-based Addressing Protocol (FAP) uses filters as data structure to compactly represent the current set of allocated addresses. With filters, the partition detection becomes easier and the number of control messages is reduced. We also use the hash of the filter, denoted filter signature, as a partition identifier. The filter signature represents a set of nodes, and thus fits well for easily detecting partitions on the network. The filters and their signatures also increase protocol robustness.

We propose the use of two different filters, depending on the scenario. The first structure is a Bloom filter, which is based on hash functions. The second one is a Sequence filter, which compresses data based on the address sequence.

A. Bloom Filters

Bloom filters are a data structure with high compression capacity, used on many applications, like IP traceback [13] and web cache [14]. A Bloom filter is a vector of m bits representing a set $A = \{a_1, a_2, a_3, \ldots, a_n\}$ composed of n elements. The elements are inserted in the filter through a set of independents hash functions (h_1, h_2, $\ldots$, h_k) whose outputs are uniformly distributed over m bits. Firstly, the bit vector is set to zero. After that, each element $a_i \in A$ is hashed by each of the k hash functions, which result represents a position to be set as 1 on the m bit vector, as shown on Fig. 1. To verify if an element a_j belongs to A, we check whether the bits of the array 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. On the contrary, it is assumed that the element belongs to A. There is, however, a probability of false-positives. By the probability of a bit to be 0 after the insertion of n elements, P_0, given by

$$P_0 = \left(1 - \frac{1}{m}\right)^{kn}, \tag{1}$$

we obtain the false-positive probability, P_{fp}, given by

$$P_{fp} = (1 - P_0)^k = \left(1 - \left(1 - \frac{1}{m}\right)^{kn}\right)^k. \tag{2}$$

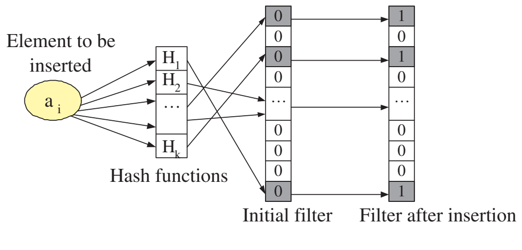


Fig. 1. Insertion of elements in the filter.

Eq. 2 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. Besides, by the derivative of Eq. 2, we obtain the value of k that minimizes the false-positive probability, which is given by

$$k = \lceil \frac{m \cdot ln(2)}{n} \rceil. \tag{3}$$

To remove elements from the filter, a counter for each bit of the filter is introduced, as shown in Fig. 2. Each counter, c_i, shows the number of times each bit was set, and $\sum_0^{m-1} c_i/k$ gives the number of elements on the filter. In [14], the authors show that the probability of a counter be greater or equal to i, $P(c \geq i)$, is given by

$$P(c \geq i) \leq m\left(\frac{e \cdot n \cdot k}{i \cdot m}\right)^i. \tag{4}$$

To avoid false-negatives, it is necessary to guarantee that the counters do not overflow. Supposing 4-bit counters ($i = 16$), a false-positive ratio of 5% and the corresponding ideal k, the overflow probability is lower than $m \cdot 10^{-12}$, which is considered negligible.

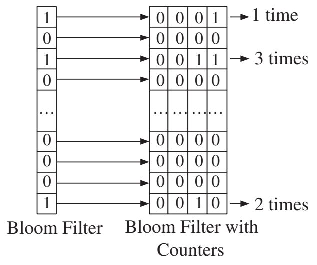


Fig. 2. Bloom filter with counters.

B. Sequence Filters

We propose another structure to store and compact addresses based on the sequence of the addresses called Sequence filter. In this filter, each address suffix is represented by one bit. The position of the bit in the filter determines the address suffix, as shown on Fig. 3. Therefore, there is no false-positive or false-negative in the Sequence filter. The procedure to insert an element on the filter is depicted on Fig. 4.

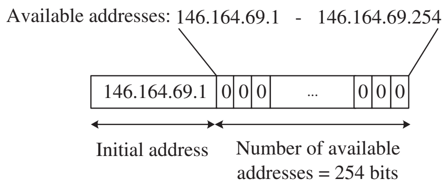

Available addresses: 146.164.69.1 - 146.164.69.254
146.164.69.1
Initial address
Number of available addresses = 254 bits

Fig. 3. The Sequence filter.

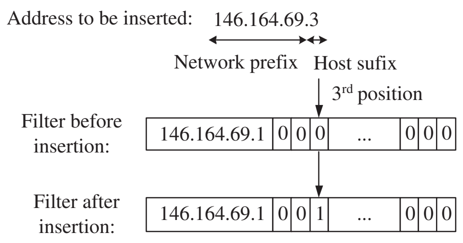


Fig. 4. Insertion of elements in the Sequence filter.

C. Bloom Filter $\times$ Sequence Filter

We use filters in the addressing to compact the list of allocated addresses. When a node joins the network, it requests to a neighbor the address filter, randomly chooses an address and check in filter whether this address is allocated. If the address is on the filter, the node chooses another address. Otherwise, the node knows that this address is available. Then, the node allocates to itself the address and floods the network with this information. While DAD [5] based protocols flood the network every time they receive an address reply indicating an address collision, the proposed protocol floods the network only one time when there is a joining node. The filters also simplify the network partition detection when compared to the use of network identifiers. With FAP, a network partition is identified when neighbors have different filters, because different filters indicate different sets of nodes. Besides, the nodes can estimate the number of nodes in each partition with the filters and only the nodes on the smaller partition with collided addresses must change their addresses. In DAD based protocols, all nodes must check their addresses for collisions. Consequently, our proposal improves the efficiency in the address allocation, reduces the control load and simplifies the network partition identification.

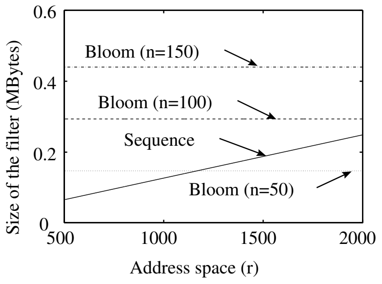


Fig. 5. Size of the filters in function of the number of available addresses, supposing 5% of false-positives for the Bloom filter.

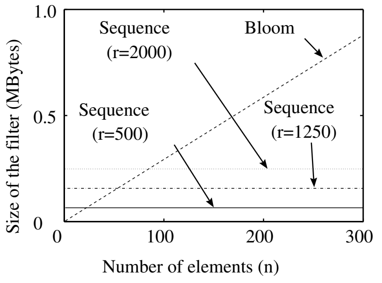


Fig. 6. Size of the filters in function of the number of elements, supposing 5% of false-positives for the Bloom filter.

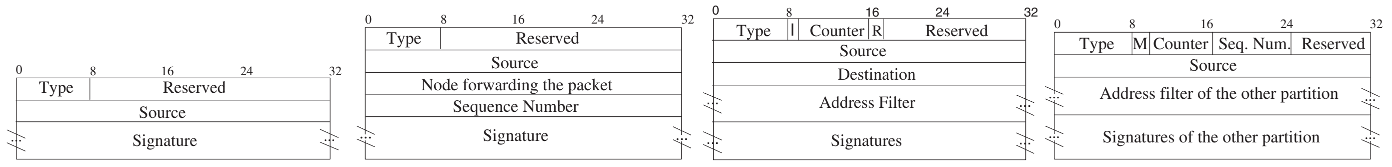
Both Bloom and Sequence filters avoid false-negatives, which means that an address check of an element that was inserted on the filter will always be positive. On the Bloom filter, however, there is a false-positive probability. Hence, an address check of an element that was not inserted on the Bloom filter may be positive. Therefore, with Bloom filters, we must choose an acceptable false-positive threshold. If we choose a false-positive probability of ≈ 0.05, the relation among the number of bits in the filter (m) and the number of inserted elements (n) is $m/n = 6$ and the ideal number of hash functions is 4, according to equations 2 and 3. Hence, the size of the Bloom filter with counters is $m = 6 \cdot n \cdot 4$. The higher the number of allocated addresses represented in the Bloom filter, n, the higher the false-positive probability. Therefore, the size of the Bloom filter is not determined by the address space, but by the number of elements to be inserted in the filter, which is the average number of active nodes in the network. On the other hand, the Sequence filter is deterministic and do not create false-positives or false-negatives. In the Sequence filter, it is easier to check if some address was already allocated. The size of this filter only depends on the size of the address space, r, and on the address size, A_s. Hence, the number of bits in the Sequence filter is given by $m = A_s + r$. Figures 5 and 6 show the size of both filters according to the size of the address space and the average number of allocated addresses, supposing IPv4 address, a false-positive probability of 5% for the Bloom filter, and the use of an ideal k, which means that $m/n = 6$ and $k = 4$. By the figures, we observe that the filters apply for different scenarios. While Bloom filter is suitable for an extensive address space and a small number of allocated addresses, the Sequence filter is adequate to a narrow address space.

D. Procedures of FAP

We now detail the main procedures of FAP and how the address filters are used in the protocol.

1) Network Initialization: The network initialization procedure deals with the autoconfiguration of the initial set of nodes. Network initialization can be gradual, with a long interval between the ingresses of nodes, or all nodes may join the network approximately at the same time. Address allocation protocols must work independently of the network initialization style, but many protocols suppose that there is a great interval between the first node ingress and the second node ingress. An example is the protocol proposed by Fan and Subramani [6], in which the first node must be alone to choose a network identifier. The next joining nodes will be handled as joining nodes by the first node. FAP works with gradual and non-gradual ingress, using Hello and *Address Requests* (AREQs) messages, shown in Figures 7(a) and 7(b).

In FAP, a joining node listens to the medium for a period T_O. If the node does not listen to any Hello, it randomly chooses an address with respect to the bits of the network prefix and starts the network initialization phase. In this phase, the node floods the network N_R times with the AREQ. After a period T_I without listening to AREQs, the node leaves the initialization phase and inserts on the address filter all the address received with AREQs. After that, the node starts to send Hellos with the address filter signature, which is a hash of the filter. If the node receives any AREQs with the same address it chose, but with a different sequence number, the



(a) Hello. (b) Address Request (AREQ). (c) Address Filter (AF). (d) Partition.

Fig. 7. Messages of FAP.

node waits for a period T_T and, then, sends another AREQ. During the period T_T, the node receives more AREQs with other allocated addresses. Therefore, after T_T, the node has a small probability of choosing again an address already chosen by another node, which avoids collisions and, consequently, reduces network control load.

2) Collision Detection in Node Ingresses and in Partition Mergings: After initialization, nodes start to send Hellos announcing their address filter signature. Any node which already has an address filter can receive a request of a node to join the network and can detect a partition merging based on the Hellos.

When a node turns on, it listens to the medium for a period T_O. If the node listens to a Hello, the network already exists and, then, the node knows that it is a joining node. Hence, the joining node will ask for the first listened node to send the address filter of the network. To obtain the filter, the joining node sends an Address Filter message (AF), shown in Fig. 7(c), to the first listened node. When this node receives the AF, it checks if bit *I* is set to 1 to confirm if the message came from a joining node. If $I = 1$, the node answers the request with another AF with bit *R* set to 1. When the joining node receives the AF with $R = 1$, it stores the address filter, randomly chooses an available address, and floods the network with an Address Request (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.

The network merging detection is also based on Hellos and AFs. 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 causes collisions after a network merging. In FAP, when a node receives a Hello, it checks if the filter signature on the message is the same of its current filter. If it is false, the node knows that the other node has a different address set. If there is more than T_P seconds since the last partition merging, a network merging procedure is started. In this procedure, both nodes exchange AFs to disseminate the filters of the two partitions. Firstly, the node checks if its IP is greater than the other node IP, because only the node with the greatest IP can start the process to avoid that both node sends AF simultaneously with bit $R = 0$. If the check is positive, the node sends the AF to the other node with its current address filter. When the other node receives the message, it stores the received filter and sends an AF with the data about its partition to the other node. Then, both nodes flood their partitions with a Partition message, shown in Fig. 7(d), to allow that all nodes update their filters with the other partition data. All nodes must check if they are on the low priority partition. The low priority partition is chosen as the shortest partition, which is known based on the number of bits set to 1 in the filters. If both partitions are of the same size, the partition of the node that started the process is chosen as low priority partition. The bit M on the partition message indicates the partition classification to the other nodes. The nodes on the low priority partition must check if their addresses are on the other partition filter, to detect the 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 it has chosen, but with a different sequence number, it chooses another address, because another node also chosen the same address. Finally, all nodes merge the received filter with its own filter, insert the received AREQs in the new filter, and update the filter signature.

The update of the filter signature after a node ingress or a partition merging follows rules and requires the field *signatures* in the AF. The AF reports the current address filter and all the current address filter signatures acceptable by the node sending the message. The nodes store these extra signatures for a short period to avoid false detections of network mergings. These false detections would occur if some node had updated its signature and sent a Hello before its neighbors have received the new information. Therefore, this time is needed to guarantee that all nodes have already received the new information on a node ingress or on a network merging. After a node ingress or a network merging, a node can change the signature of its filter in the Hellos only after T_S, when all the other nodes have obtained the new filter. Besides, the old signature can be discarded only after $2 \cdot T_S$ when all the nodes have updated their signatures to the new filter. With this procedure, all the nodes can correctly validate their filters during the mechanisms that change the address filter content. If the signature on the Hello is different of their current filter signatures, but equal to some stored signature, the nodes will not consider that the neighbor is on another partition.

3) Nodes Departure: When a node leaves the network, its address should become available for the joining nodes. In case the departing node is correctly shut down, it floods the network with a notification, which would cause the removal of its address of the address filter of each network node. If the departing node does not notify the network, the address remains allocated on the filters, which can make the available addresses scarce after many departures. This can be identified in the address filter by the fraction of bits set to 1 in the filter. Therefore, every node verify the fraction of bits in 1 in their address filters every time the filter is updated. If this fraction reaches a threshold, all nodes reset their address filters and returns to the network initialization. Although the initialization phase seems to overcharge the network, it is equivalent in terms of control load to a partition merging in stateless autoconfiguration protocols, such as the protocol proposed by Fan and Subramani [6]. To avoid frequent address filter renews in networks with high occupation of the address space, there is a minimum period among filter renews, T_M.

IV. Analytical Results

We analyzed FAP, whose state machine is on Fig. 8, to evaluate possible errors during the address allocation which could cause an address collision. Errors can occur if two different nodes generate Address Requests (AREQs) with the same address and the same sequence number, or if two isolated partitions have exactly the same filters and try to merge. In the first case, the joining nodes do not identify that their addresses are the same, because the message of 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 partitions, and, consequently, the merged network would have a collision for each of its addresses. Apart from these two cases, nodes joining the network never cause errors. A joining node always sends an AREQ, and, when any node already with an address receives an AREQ, it must check for collision independent of its current state. Therefore, if two joining nodes receive at the same time the address filter and choose the same address, both nodes will change their addresses because each node will receive the AREQ of the other. Messages losses also do not cause errors in FAP because they are compensated by false partition merging procedures. If some node does not receive an AREQ, its filter will be different from the filter of the other nodes. This will cause a false partition merging procedures until all nodes have the same information in their filters.

The probability of two nodes choosing the same address and the same sequence number for an AREQ, P_A, can be calculated based on the birthday paradox with r being the address space plus the sequence number space and n, the number of simultaneous accesses to the network. Hence,

$$P_A = 1 - \left[\frac{r-1}{r} \cdot \frac{r-2}{r} \cdots \frac{r-n+1}{r}\right]. \tag{5}$$

Fig. 9 shows P_A as function of the number of nodes simultaneously trying to access the network, considering host suffixes of 8 bits. We observe that even for a number of nodes greater than the number of available addresses, this probability is very small and negligible.

The probability of two different partitions having the same address filter was calculated for both Bloom filter and Sequence filter. For the Sequence filter, this probability, P_{SF}, is calculated based on the probability of two filters having the same number of elements and on the probability that two filters with the same number of elements have the same set of inserted addresses. Let A_1 be the set of address of the first partition, A_2 be the set of address of the second partition, $N(x)$ be the number of elements in set x, and r be set of available addresses. Hence, $A_1 \subset r$, $A_2 \subset r$, and $N(A_1) + N(A_2) \leq N(r)$. Therefore, P_{SF} for the Sequence filter and for an address space r, is given by

$$P_{SF} = \sum_{i=1}^{\lfloor \frac{r}{2} \rfloor} \frac{C_i^r}{\sum_{j=1}^{r-1} \left(C_j^r \cdot \sum_{k=1}^{r-j} C_k^r\right)}. \tag{6}$$

For the Bloom filter with counters, this probability also depends on the false-positives. The false-positive probability of a Bloom filter is greater than the false-positive probability of the same filter with counters, and then, the probability of two different partitions have the same Bloom filter (P_{BF}) is an upper bound for the case of Bloom filters with counters. Hence, we calculated the probability of two Bloom filters have

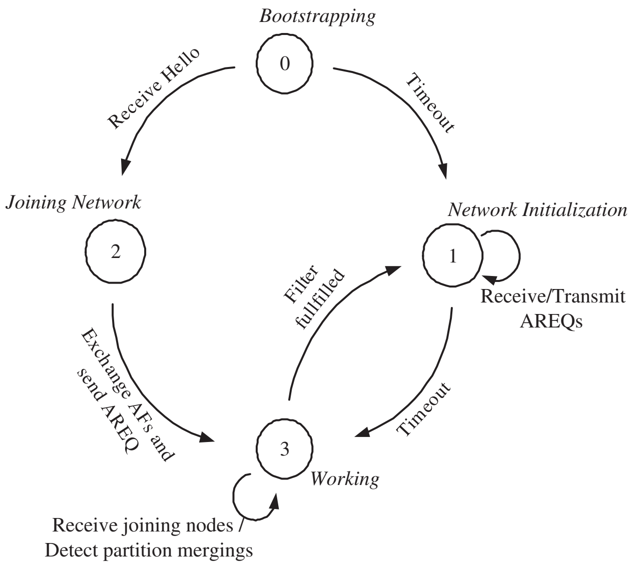


Fig. 8. State Machine of FAP.

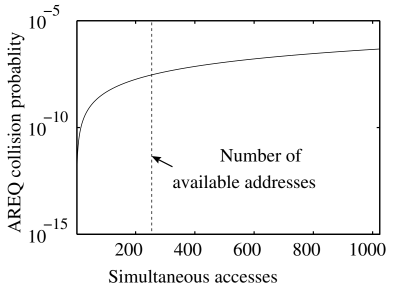


Fig. 9. Probability of collisions of addresses and sequence numbers during simultaneous accesses to the network.

the same bits in 1 and in 0 for all possibilities of number of elements in each filter, which is given by

$$P_{BF} = \sum_{n=1}^{r-1} \sum_{n'=1}^{r-n} \sum_{i=0}^{m} \frac{C_i^m \left[p_0^i \cdot p_1^{(m-i)}\right] \left[{p'}_0^i \cdot {p'}_1^{(m-i)}\right]}{\sum_{n=1}^{r-1} \sum_{n'=1}^{r-n} 1}, \quad (7)$$

where p_0 and p'_0 are the probability of a bit be equal to 0 after n and n' insertions on the filter, which is given by Eq. 1, $p_1 = 1 - p_0$, $p'_1 = 1 - p'_0$, r is the address space, m is the size of the filter, and k is the number of hash functions.

Both Equations 6 and 7 are presented on Fig. 10, which shows that these probabilities are also negligible. Therefore, FAP is prone to errors due to AREQs collisions or to address filter collisions in initialization, joining nodes, and partition merging procedures.

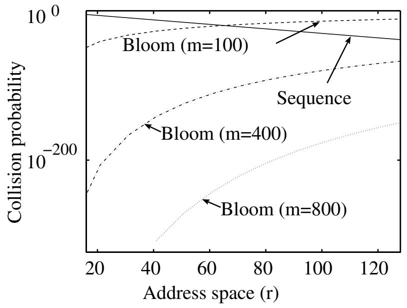


Fig. 10. Probability of address filter collision during partition mergings.

V. Simulation Results

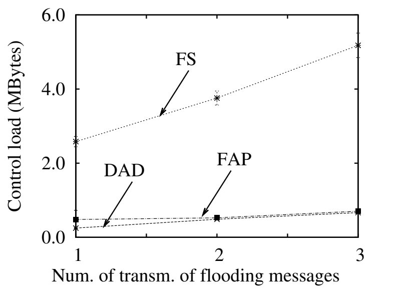
We implemented FAP in the Network Simulator-2. The simulations model radio propagation using the Shadowing model and the Medium Access Control using ns-2 IEEE 802.11 model. These models account for creating a scenario similar to a real community network, using parameters of commercial equipments. Therefore, the parameters used for our simulation are an average transmission range of 18.5 m, a maximum carrier sense range of $108m$, and a density of $0.0121\ nodes/m^2$ [15]. We evaluated the control traffic, the delays, and the number of address collisions on FAP and on the other analyzed protocols. We compare FAP with the current version of Duplicate Address Detection (DAD), because DAD is the base for many protocols. We also compared FAP with an improvement of the proposal of Fan and Subramani, which we call FS in the graphs. The original FS is a stateless protocol, which uses network identifiers to detect network partitions and DAD to resolve collisions. In the original proposal, nodes sum network identifiers in all network mergings, but this strategy causes instability in the protocol. Therefore, we improved the protocol performance by choosing the greatest network identifier in network mergings instead of summing them, which reduces the number of false partition merging detections. By comparing FAP with FS, we can examine the performance impact of the use of the address filters to reduce the control load. We consider a confidence level of 95% in the results.

TABLE I
Parameters of FAP.

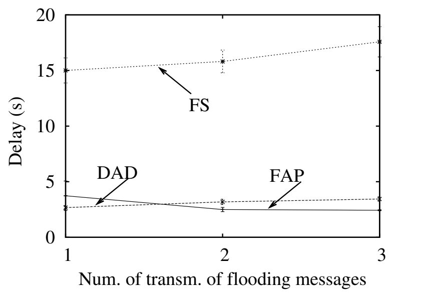
Variable	Description	Value
T_O	Bootstrap time	1 s
T_P	Min. interval between partition merging procedures	3 s
T_T	Time waiting before changing address	0.3 s
T_I	Time waiting for AREQs in the initialization	1.2 s
T_R	Interval between replications of flooding messages	0.3 s
T_H	Interval between retransmissions of Hellos	1s
T_S	Time storing filter signatures of its own partition	0.5 s
$T_{S'}$	Time storing filter signatures of other partitions	3 s
T_M	Min. interval between filter renews	5 s

The protocol parameters are on Table I. These parameters were chosen based on experiments to increase all the three protocols performance. Both FAP and FS use Hellos with the same size, because we assume that both network identifier and filter signature are composed of 4 bytes. We assume an address space of 150 addresses and an address space average occupation of $1/3$, which means about 50 nodes in the network. Therefore, we use the Sequence Filter of 23 bytes.

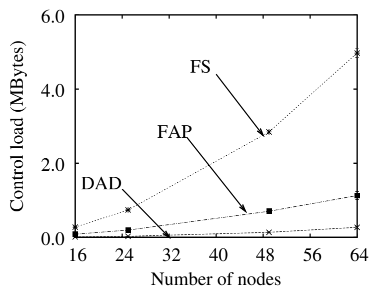
Network initialization is the first analyzed procedure. We consider a rectangular space with 49 nodes. All nodes join the network at the same time and randomly choose their addresses based on pseudo-random number generators with a different seed for each node. The average number of collisions on the initial address set was 8 collisions and, at the end of the simulation, all protocols resolved all the collisions. The total load in bytes of the address autoconfiguration protocols are on Fig. 11(a). DAD has the shortest load with one transmission of flooding messages, because DAD does not handle partitions, and, therefore, does not use Hellos. With two or three transmissions of flooding messages, FAP and DAD have approximately the same control load, because FAP sends less AREQs and does not need any AREP, which balances the use of Hellos. Another reason for the load of FAP being greater than the load of DAD considering one transmission is the capacity of FAP to detect message losses. In DAD or FS, if some message collides and does not reach a node, this node will not receive the information unless there is a retransmission of the message that does not collide. This can cause address collision if the lost message has the same address of the node which does not receive the message. FAP is robust to message losses, 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 partition union procedures, caused by losses, will be started until all nodes have the same address filter. With only one transmission of each message, there are information losses that are compensated by FAP with the false partition procedures, but not by DAD or FS, which explains the difference in load of DAD and FAP. FS has the worst result because it is stateless, uses Hellos, and does not handle network initialization with simultaneous ingresses. When many nodes join the network at the same time, each



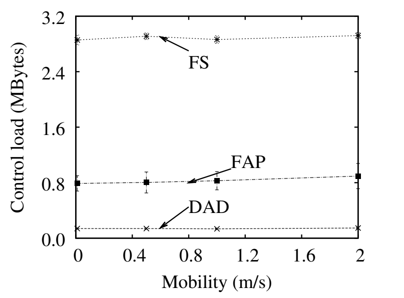
(a) Initialization control load with 49 nodes and simultaneous access.



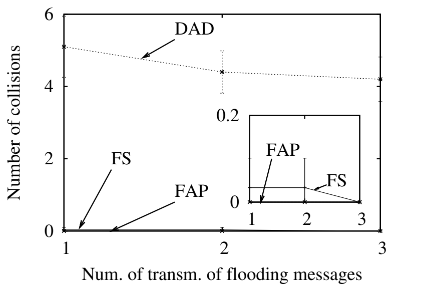
(b) Delay to allocate an address on the initialization with 49 nodes and simultaneous access.



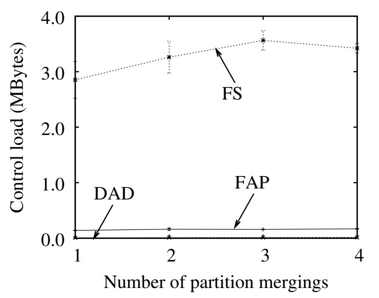
(c) Effects of the number of nodes with two transmissions of flooding messages and gradual ingress.



(d) Effects of mobility with 49 nodes, two transmissions of flooding messages, and gradual ingress.



(e) Number of address collisions with 50 nodes on a partition merging.



(f) Control load with 50 nodes on partition mergings and two transmissions of flooding messages.

Fig. 11. Simulation results for control load, delays and collisions.

node using FS chooses its own network identifier and handles the neighbors as partitions. Hence, FS has a load up to seven times greater than the load of FAP.

The delay for allocating address on the network initialization for the three protocols is on Fig. 11(b). The delay is the time between the simulation start and the reception of the last address autoconfiguration message for each node. Nodes cannot establish connections until the end of the address autoconfiguration, because, during this process, any node can change its address, which would end the connection. FAP has the best result for two or three transmissions of flooding messages, with a reduction of up to 1 s when compared to DAD and of 15 s when compared to FS. With only one transmission, the delay of FAP is greater than the delay of DAD due to the false partition merging procedures to compensate lost messages.

In the initialization, all the protocols can resolve all the collisions, but FAP and DAD show the best result in terms of control load and delays. DAD, however, is a simple protocol that does not handle network partitions, and both DAD and FS do not handle message losses.

The second analysis is related to the impact of the number of nodes in the network. We used again a rectangular space, simulating a community network, composed of static nodes. Each node randomly chooses the time to ingress in the network among 1 and 20 s and the time to leave the network among 20 and 40 s. Again, all the protocols resolved all the collisions and the results for control load with two transmissions of each flooding message are on Fig. 11(c). The increase of the number of nodes impacts all the protocols, because of the floodings to disseminate messages through the network. FS, however, suffers a greater influence on the control load than FAP and DAD, because it floods the network more times due to the false-positives in the partition merging procedure without the use of an address filter. In this scenario, FS has a load up to 4.4 greater than the load of FAP.

Mobility effect is analyzed in a scenario of 50x50m and 49 nodes, to evaluate the impact of mobility in the protocols without the influence of the partition mergings. Each node in our simulation moves according to the random-way point model with a pause time of 1 s. We suppose two transmissions of each flooding message, to reduce the effects of message losses. The nodes randomly choose the time to ingress in the network among 1 and 20 s, to simulate a gradual ingress in the network. None of the protocols suffered with mobility. The difference among FAP and DAD control load is due to the use of Hellos and to the joining node ingress procedure in FAP. FS has a load up to 3.7 times greater than the load of FAP, even though the ingress of the nodes was gradual, which benefits FS. Therefore, the use of filters has a great impact on reducing the control load.

The last analysis is the performance on partition mergings of the three protocols. We vary the number of partitions mergings in a static scenario composed of 50 nodes by activating/deactivating some nodes responsible for connecting isolated partitions. The nodes on each isolated partition are activated simultaneously and the nodes that interconnect the partitions are activate in fixed times. Fig. 11(e) shows the number of address collisions after the end of the simulation for one partition merging. DAD, as expected, does not resolve any collisions, because this protocol was not designed to control partitions mergings, and, consequently, cannot resolve collisions created by the partition merging. FAP has no address collision after all runs, but FS presents collisions when using only one retransmission of the flooding messages. FAP avoids this kind of collision because of the mechanism that detects message losses and start false partition merging procedures until all the nodes have received the same information. As FS does not detect message losses, the protocol is efficient only when more than one transmission of the flooding messages is used.

Fig. 11(f) depicts the control load of the protocols during partition merging procedures with each flooding message being transmitted two times. The control load of the network initialization is not considered in this analysis. DAD has almost no control load because it does not detect partition. The control load of DAD is only composed of the AREQs of the nodes connecting the partitions. Therefore, this protocol is inefficient to ad hoc networks, because it cannot avoid collisions caused by partition mergings. FS has a control load up to 22 times greater than FAP, because FAP reduces the number of AREQs during the partition merging. In FAP, only half or less of the nodes check whether their addresses are collided and only the nodes with collided address choose a new address and send an AREQ. In FS, all the nodes of the network should send an AREQ, and each collision will imply, at least, one more flooding of the network. By Fig. 11(f), we can also observe that the number of partitions does not impact the control load of FAP. The number of AFs increases with the number of partitions, but these messages are sent in unicast and do not impact the control load. The number of AREQs, however, decreases, because the number of nodes in each partition is shorter, reducing the impact of floodings.

VI. Conclusion

Address assignment in ad hoc networks should be automatic, fast, and without collisions. We proposed a Filter-based Addressing protocol (FAP), which uses address filters to reduce the control load and the delay to allocate addresses. Besides, filters allow an accurate partition merging detection and increase the protocol robustness.

Simulation shows that FAP resolves all the address collisions even during partition mergings. In the initialization of the network, FAP presents a control load similar to the control load of DAD, which is a simple protocol that does not handle partition. Therefore, FAP is an efficient proposal to autoconfigure addresses in the network. Besides, simulation results show that FAP, when compared to protocols like the proposal of Fan and Subramani (FS), is less impacted by the increase of the network size. In the partition merging procedure, FAP has a control load up to 22 times smaller than the control load of FS. Moreover, FAP with two transmissions of the flooding messages reduces the delay in the autoconfiguration even when compared to DAD.

FAP is robust to messages losses, which is an important issue for ad hoc networks, which has fading channels and high bit error rates. In addition, FAP handles nodes joining/leaving the network and partition mergings in a distributive way, and is efficient in terms of transmitted messages.

Acknowledgment

The authors would like to thank CNPq, CAPES, FINEP, FUNTTEL, FAPERJ, RNP, and FUJB.

References

[1] D. O. Cunha, O. C. M. B. Duarte, and G. Pujolle, "A cooperation-aware routing scheme for fast varying fading wireless channels," *To appear in IEEE Communications Letters*.

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

[3] S. Thomson and T. Narten, *IPv6 stateless address autoconfiguration*, RFC 2462, Dec. 1998.

[4] T. Narten and R. Draves, *Privacy Extensions for Stateless Address Autoconfiguration in IPv6*, RFC 3041, Jan. 2001.

[5] C. E. Perkins, E. M. Royers, and S. R. Das, *IP address autoconfiguration for ad hoc networks*, Internet Draft, Jul. 2000.

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

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

[8] J. Eriksson, M. Faloutsos, and S. V. Krishnamurthy, "DART: dynamic address routing for scalable ad hoc and mesh networks," *IEEE/ACM Transactions on Networking*, vol. 15, no. 1, pp. 119–132, 2007.

[9] S.-C. Kim and J.-M. Chung, "Message complexity analysis of mobile ad hoc network address autoconfiguration protocols," *IEEE Communications Magazine*, vol. 45, no. 1, pp. 136–143, Jan. 2008.

[10] S. Nesargi and R. Prakash, "MANETconf: configuration of hosts in a mobile ad hoc network," in *Twenty-First Annual Joint Conference of the IEEE Computer and Communications Societies (IEEE INFOCOM 2002)*, vol. 2. IEEE, jun 2002, pp. 1059–1068.

[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 *IEEE International Symposium on Consumer Electronics (ISCE 2007)*. IEEE, jun 2007, pp. 1–6.

[12] H. Zhou, L. Ni, and M. Mutka, "Prophet address allocation for large scale MANETs," in *Twenty-Second Annual Joint Conference of the IEEE Computer and Communications Societies (IEEE INFOCOM 2003)*, vol. 2. IEEE, mar 2003, pp. 1304–1311.

[13] R. P. Laufer, P. B. Velloso, D. O. Cunha, I. M. Moraes, M. D. D. Bicudo, M. D. D. Moreira, and O. C. M. B. Duarte, "Towards stateless single-packet IP traceback," in *32nd IEEE Conference on Local Computer Networks (LCN'2007)*, Oct. 2007, pp. 548–555.

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

[15] 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 Communications Magazine*, vol. 45, no. 1, pp. 136–143, Jan. 2007.