ISSN:0975-9646

A Filter Base Addressing Protocol for Node Auto-Configuration in Wireless Ad Hoc Network Using Cuckoo Filter.

Mr. Bhushan A. Badgujar.
Computer Science &Engineering (M.E.2nd Year)
G.H.R.I.E.M.,Jalgaon.
Jalgaon.

Mrs. Swati patil.
Assist. Professor.
G.H.R.I.E.M.,Jalgaon.
Jalgaon.

Mr. Mayur Agrawal.
Assist.Professor.
G.H.R.I.E.M.,Jalgaon
Jalgaon.

***Abstract*— Mobile ad hoc networks do not have fixed infrastructure. Nodes are keep on joining and leaving the network dynamically. Assigning addresses for mobile nodes is a challenging task. The difficulty is even raised due partitions in network and dynamic joining and leaving of the nodes in MANET. Address collisions are quite a common problem in mobile ad hoc networks. Here, a light weight protocol called Filter-based Addressing Protocol (FAP) is used to solve this problem that configures mobile ad hoc nodes based on a distributed address database stored in filters. This paper describe the use of two filters cuckoo filter and sequence filter to design a filter based protocol.**

Cuckoo filters is used to adding and removing items dynamically while achieving higher lookup performance and also use less space than conventional Bloom filters for applications that require low false positive rates. Cuckoo filters also have lower space overhead than space-optimized Bloom filters.

***Keywords*—Ad-hoc networks, Addressing mechanisms, IP Address configuration, Filters.**

I. INTRODUCTION

Mobile ad hoc networks connect mobile devices in an infrastructure without wires and configure themselves continuously to connect with network.Mobile nodes are free to move in the network as per their wish and have frequently changing positions and links. But every node in network acts as a router. creating a mobile ad hoc network means providing each node with the necessary information for routing the traffic [4]. These networks may operate by themselves or may get connected to larger internet. This makes it a dynamic, autonomous topology[10].

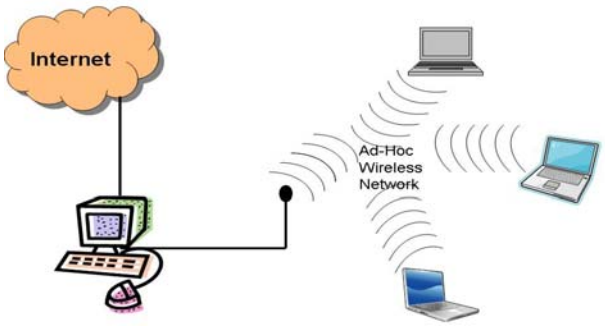


Fig.1 MANET

Now-a-days, many distributed applications prefer mobile ad hoc networks as there is no centralized administration. Mobile ad hoc networks are based on dynamic multihop topologies for communication and do not have any previous infrastructure. In addition, network partitioning is a significant problem which unfortunately is not properly taken care of Mobility of nodes, channel fading and many other issues disturb the control over the distributed network. Initialization of network is also not an easy task due to the lack of servers [11]. Achieving multi-hop routing and full connectivity makes a unique addressing network is more essential. Self-management makes this network more complex. Further,DHCP (Dynamic Host configuration protocol) and NAT(Network Allocation Table) do not serve the distributed nature like dynamic partitioning and merging of the network. This paper describes the FAP model which uses filters to solving the above mentioned issues.Here, distributed database containing the currently allocated addresses is maintained in filters in a compact fashion [4]. To design a filter-based protocol, here use two filters called a Cuckoo and a Sequence filter. This scheme ensures not only the univocal address configuration of the nodes joining the mobile network and address collision detection after partition merging. this is a simple way because here every node has the knowledge of the already assigned addresses. Also, to easily detect network partitions.in this technique a hash of the filter is provided as the partition identifier [4]. But here used Cuckoo filter instead of conventional Boom filters. Because Cuckoo filters allow adding and removing items dynamically while achieving higher lookup performance [13], and also use less space than conventional and non-deletion-supporting Bloom filters [1]. Here we presents the cuckoo filter, It is a data structure that can replace both counting and traditional Bloom filters with three major advantages:

1. itsupports adding and removing items dynamically;
2. itachieves higher lookup performance; and
3. it requiresless space than a space-optimized Bloom filter when the target false positive rate e is less than 3%.

II. RELATED WORK

2.1. Sequence Filters:

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, Therefore, there is no false-positive or false-negative in the

Sequence filter. It stores and compacts addresses based on the sequence of addresses. This filter is created bythe concatenation of the first address of the address sequence, which they call initial element (a_0), with a r-bit vector, where r is the address range size [4].

2.2. Bloom Filters:

Bloom filters are a data structure with high compression capacity, used on many applications, like IP traceback [4][5] and web cache [9]. 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 in Fig. 1. To verify if an element *aj* belongs to *A*, They check whether the bits of the array corresponding to the positions $h_1(aj), h_2(aj), \ldots, h_k(aj)$ are all set to 1. If at least one bitis set to 0, then *aj* 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 [3]. By the probability of a bit to be 0 after the insertion of *n* elements, P_0, given by,

$$P_0 = (1 - \frac{1}{m})^{kn} \ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots (1)$$

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

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

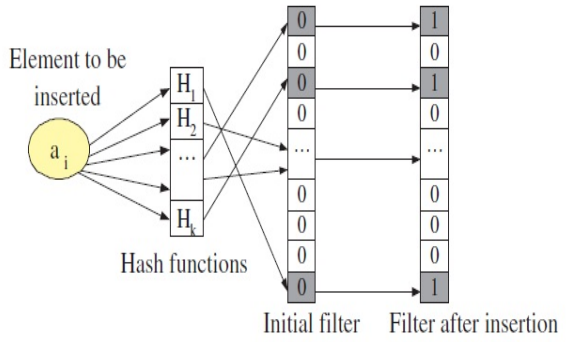


Fig.2. Insertion of elements in the filter.

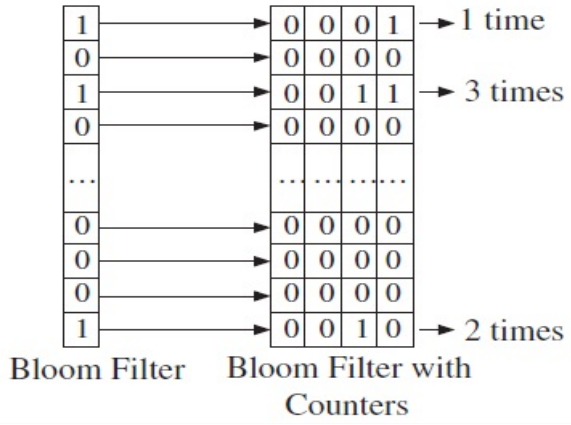
Equation-2. shows that the false-positive probability decreases when the number of elements, n, of set A is decreased or the size of the filter, m, is increased. Besides, by the derivative of Eq. 2, they obtain the value of k that minimizes the false positive probability, which is given by,

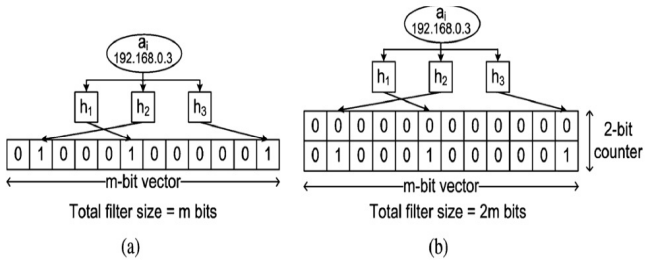
$$K = [\frac{m.\ln(2)}{n}] \ldots\ldots\ldots\ldots\ldots\ldots\ldots (3)$$

To remove elements from the filter, a counter for each bit of the filter is introduced, as shown in Fig.3. Each counter c_i, shows the number of times each bit was set, and $\sum_0^{m-10} \frac{ci}{k}$ gives the number of elements on the filter. the authors [7] show that the probability of a counter be greater or equal to i, $P(c \geq i)$, is given by,

$$P(c \geq i) \leq m (\frac{e.n.k}{i.m})^i \ldots\ldots\ldots\ldots\ldots....(4)$$

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 · 10−12, which is considered negligible [3][7].





(a) Bloom filter with k=3 hash functions (b) 2 bit counter bloom filter

Fig.3. Bloom filter with counters.

2.3 Bloom Filters and Variants:

Here we compare standard Bloom filters and the variants that include support for deletion or better lookup performance.Cuckoo filters achieve higher space efficiency and performance than these data structures [1].

2.3.1. Standard Bloom filters [9]:

This filter provides a compact representation of a set of items that supports two operations: ***Insert*** and ***Lookup.*** A Bloom filter allows a tunable false positive rate Є so that a query returns either "definitely not" (with no error),or "probably yes" (with probability Є of being wrong). Thelower Є is, the more space the filter requires.A Bloom filter consists of k hash functions and a bit arraywith all bits initially set to "0". To insert an item, it hashesthis item to k positions in the bit array by k hash functions,and then sets all k bits to "1". Lookup is processed similarly,except it reads k corresponding bits in the array: if all thebits are set, the query returns true; otherwise it returns false.Bloom filters do not support deletion. Bloom filters can be very space-efficient, but are not optimal [3]. For a false positive

rate Є, a space-optimized Bloom filter uses k = $\log_2$ (1/Є) hash functions. Such a Bloom filter can store each item using 1.44 $\log_2$ (1/Є) bits, which depends only on (1/Є) rather than the item size or the total number of items.The information-theoretic minimum requires $\log_2$ (1/Є) bits per item, so a space-optimized Bloom filter imposes a 44%space overhead over the information-theoretic lower bound.The information theoretic optimum is essentially achievable for a static set by using fingerprints and a perfect hash table [6]. To efficiently handle deletions, we replace a perfect hash function with a well-designed cuckoo hash table [13].

2.3.2 Counting Bloom filters [7]:

This extend Bloom filters to allow deletions. A counting Bloom filter uses an array of countersin place of an array of bits. An insert increment the value of k counters instead of simply setting k bits, and a lookupchecks if each of the required counters is non-zero. The delete operation decrements the values of these k counters. To prevent arithmetic overflow (i.e., incrementing a counter that hasthe maximum possible value), each counter in the array must be sufficiently large in order to retain the Bloom filter's properties.In practice, the counter consists of four or more bits,and a counting Bloom filter therefore requires 4X more spacethan a standard Bloom filter. (One can construct countingBloom filters to use less space by introducing a secondary hash table structure to manage overflowing counters, at the expense of additional complexity.)

2.3.3 Blocked Bloom filters [10]: do not support deletion, but providebetter spatial locality on lookups. A blocked Bloom filter consists of an array of small Bloom filters, each fitting in one CPU cache line. Each item is stored in only one of these small Bloom filters determined by hash partitioning. As a result,every query causes at most one cache miss to load that Bloomfilter, which significantly improves performance. A drawbackis that the false positive rate becomes higher because of theimbalanced load across the array of small Bloom filters.

2.3.4 d-left Counting Bloom filters [5]: are similar to the approach we use here. Hash tables using d-left hashing [9] store fingerprints for stored items. These filters delete items by removing their fingerprint. Compared to counting Bloom filters, they reduce the space cost by 50%, usually requiring 1.5-2X thespace compared to a space-optimized non-deletable Bloom filter. Cuckoo filters achieve better space efficiency than d-leftcounting Bloom filters as we show, and have other advantages,including simplicity.

2.3.5 Quotient filters [9]: are also compact hash tables that storefingerprints to support deletion. Quotient filters uses a technique similar to linear probing to locate a fingerprint, and thus provide better spatial locality. However, they require additional meta-data to encode each entry, which requires 10 ~ 25% more space than a comparable standard Bloomfilter. Moreover, all of its operations must decode a sequence of table entries before reaching the target item, and the morethe hash table is filled, the longer these sequences become.As a result, its performance drops significantly when the occupancyof the hash table exceeds 75%.

2.3.6 Limitations of conventional Bloom filters [1]: One major limitation of Bloom filters is that the existing items cannotbe removed without rebuilding the entire filter. Severalproposals have extended classic Bloom filters to supportdeletion, but with significant space overhead: *counting Bloom filters* [3] are 4Xlarger and the recent *d-leftcounting Bloom filters (dl-CBFs)* [8], which adopt a hashtable-based approach, are still about 2X larger than a space-optimized Bloom filter.

III. PROPOSED WORK.

FILTER-BASED ADDRESSING PROTOCOL (FAP) :

This protocol aims to dynamically auto-configure addresses, identifying and solving addresses collisions with a low control load, even when there are nodes joining the network or partition merging [3][5]. To obtained all these objectives, Here we used the Filter-based Addressing Protocol (FAP) that 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. They also use the hash of the filter, that represent filter signature, as a partition identifier. The filter signature represents a set of nodes, and fits well for easily detecting partitions on the network. It uses the filter signature (a hash of the filter) as a partition identifier instead of random numbers. The filter signature (i.e.Hash of filter) represents the set of all the nodes within the partition. Therefore, if the set of assigned address of the node is changes, the filter signature also changes.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 [10].

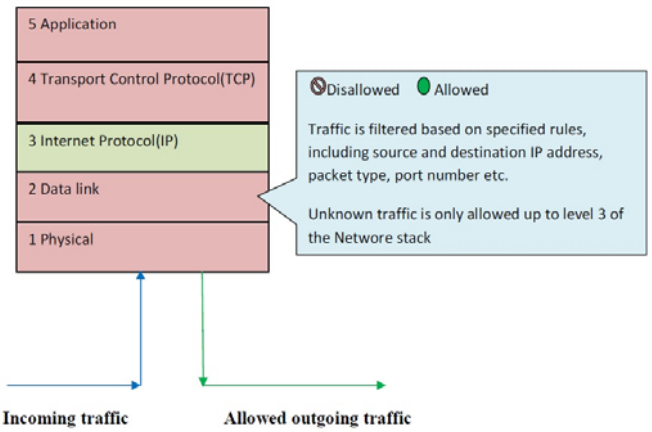


Fig. 4 Filtering of Traffic at different layers.

Centralized addressing schemes are not suitable for mobile adhoc networks because there are no servers. Further, simple distributed schemes are vulnerable to duplicated addresses which may result in high probability of address collisions. In view of the problems involved in the above mentioned schemes, another solution is needed. One such way is the Filter-based Addressing Protocol. Here, present a lightweight protocol called Filter-based Addressing Protocol (FAP) that configures mobile ad hoc nodes based

on a distributed address database stored in filters.This protocol maintains a distributed database stored in filters containing the currently allocated addresses in a compact fashion [4]. FAP consider both the Cuckoo filter and Sequence filter. But here used cuckoo filters that can replace traditional Bloom filters for many approximate set-membership test applications [1]. Cuckoo filters allow adding and removing items dynamically while achieving higher lookup performance and also use less space than conventional, and non-deletion-supporting Bloom filters. Cuckoo filters also have lower space overhead than space-optimized Bloom filters.

A limitation of standard Bloom filters is that one cannot remove existing items without rebuilding the entire filter. To solve this problem here we used the cuckoo filter,It is a practical data structure that provides four major advantages.

1. It supports adding and removing items dynamically;
2. It provides higher lookup performance than traditional Bloom filters.
3. It is easier to implement than alternatives such as the quotient filter; and
4. It uses less space than Bloom filters in many practical applications, if the target false positive rate Є is less than 3%.

CUCKOO FILTER :

The cuckoo filter is a compact data structure for approximate set-membership queries where items can be added and removed dynamically in O(1) time. Essentially, it is a highly compact cuckoo hash table that stores fingerprints (i.e., short hash values) for each item. This technique was first introduced in previous work [6], but there the context was improving the lookup and insert performance of regular cuckoo hash tables where full keys were stored. In contrast, this paper focuses on optimizing and analyzing the space efficiency when using partial-key cuckoo hashing with only fingerprints, to make cuckoo filters competitive with or even more compact than Bloom filters[1].

Basic Cuckoo Hash Table:

Cuckoo hashing [13] is an open addressing hashing scheme to construct space-efficient hash tables [6]. A basic cuckoo hash table consists of an array of buckets where each item has two candidate buckets determined by hash functions $h_1(x)$ and $h_2(x)$ (see Figure 5). Looking up an item checks both buckets to see whether either contains this item. If either of its two buckets is empty, we can insert a new item into that free bucket; if neither bucket has space, it selects one of the candidate buckets (e.g., bucket 6), kicks out the existing item ("a"), and re-inserts this victim item to its own alternate location (bucket 4). Displacing the victim may also require kicking out another existing item ("c"), so this procedure may repeat until a vacant bucket is found, or until a maximum number of displacements is reached (e.g., 500 times in our implementation). If no vacant bucket is found, the hash table is considered too full to insert and an expansion process is scheduled. Though cuckoo hashing may execute a sequence of displacements, its amortized insertion time is still O(1). Cuckoo hashing ensures high space occupancy because it can refine earlier item-placement decisions when inserting new items.

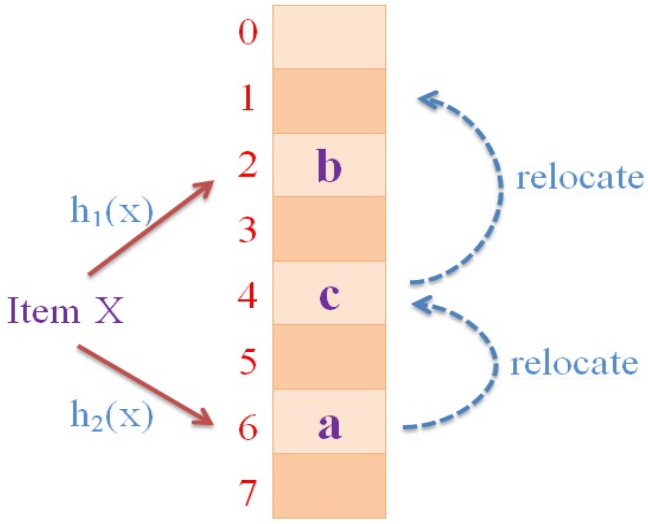


Fig. 5: A cuckoo hash table with eight buckets

Dynamic Insert

When inserting new items, cuckoo hashing may relocate existing items to their alternate locations in order to make room for the new ones. Cuckoo filters store only the item's fingerprints in the hash table and therefore have no way to read back and rehash the original items to find their alternate locations (as in traditional cuckoo hashing). Thereforehere partial-key cuckoo hashing is usedto derive an item's alternate location using only its fingerprint. For an item x, hashing scheme calculates the indexes of the two candidate buckets $i_1$and $i_2$as follows:

$$i_1 = \text{HASH}(x),$$
$$i_2 = i_1 \oplus \text{HASH}(x\text{'s fingerprint}). \quad \text{Eq. (1)}$$

The exclusive-OR (X-OR) operationin Eq. (1) ensures an important property that is $i_1$can be computed using the same formula from $i_2$and the fingerprint; so, to displace a key originally in bucket i (no matter whether i is $i_1$or i_2), we can directly calculate its alternate bucket j from the current bucket index i and the fingerprint stored in this bucket by,

$$j = i \oplus \text{HASH}(\text{fingerprint}). \quad \text{Eq. (2)}$$

Hence, insertion can complete using only information in the table, and never has to retrieve the original item x.

Note that here hash the fingerprint before it is XOR with the index of its current bucket, in order to help distribute the items uniformly in the table. If alternate location is calculated by ***"i ⊕Fingerprint"*** without hashing the fingerprint, the items kicked out from nearby buckets will land close to each other in the table, assuming the size of the fingerprint is small compared to the table size. Hashing ensures that items kicked out can land in an entirely different part of the hash table.

Partial-Key Cuckoo Hashing Ensure High Occupancy:

The values of i_1 and i_2 calculated by Eq. (1) are uniformly distributed, individually. They are not necessarily indepen-dent of each other (as required by standard cuckoo hashing). Given the value of i_1, the number of possible values of i_2 is at most 2^f where each fingerprint is f bits; when $f \leq \log_2 r$ where r is the total number of buckets, the choice of i_2 is

only a subset of all the r buckets of the entire hash table. For example, using one-byte fingerprints, given i_1 there are only up to $2^f = 256$ different possible values of i_2 across the entire table; thus i_1 and i_2 are dependent when the hash table contains more than 256 buckets. This situation is relatively common, for example, when the cuckoo filter targets a large number of items but a moderately low false positive rate.

Dynamic Delete

With partial-key cuckoo hashing, deletion is simple. Given an item to delete, here check both its candidate buckets; if there is a fingerprint match in either bucket, we just remove the fingerprint from that bucket. This deletion is safe even if two items stored in the same bucket happen to have the same fingerprint. For example, if item x and y have the same fingerprint, and both items can reside in bucket i_1, partial-key cuckoo hashing ensures that bucket $\mathbf{i_2 = i_1 \oplus HASH(fingerprint)}$ must be the other candidate bucket for both x and y. As a result, if we delete x, it does not matter if we remove the fingerprint added when inserting x or y; the membership of y will still return positive because there is one fingerprint left that must be reachable from either bucket i_1 and i_2.

Optimizing Space Efficiency

Increasing bucket capacity (i.e., each bucket may contain multiple fingerprints) can significantly improve the occupancy of a cuckoo hash table [4]; mean while, comparing more fingerprints on looking up each bucket also requires longer fingerprints to retain the same false positive rate (leading to larger tables). Here explored different configuration settings and found that having four fingerprints per bucket achieves a sweet point in terms of the space overhead per item. Here main focus on the cuckoo filters that use two hash functions and four fingerprints per bucket [13].

IV. CONCLUSION

Address assignment in ad hoc networks should be automatic, fast, and without collisions. Here 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. Here describe the use of cuckoo filter and sequence filter to design a filter based protocol. Cuckoo filters are a new data structure for approximate set membership queries that can be used for many networking problems formerly solved using Bloom filters. Cuckoo filters improve upon Bloom filters in three ways: 1. support for deleting items dynamically; 2. better lookup performance; and 3. better space efficiency for applications requiring low false positive rates. A cuckoo filter stores the fingerprints of a set of items based on cuckoo hashing, thus achieving high space occupancy.

REFERENCES.

[1] Bin fan, david G. Andersen, Michael kaminsky, michael D. Mitzenmacher "*Cuckoo Filter: Practically Better Than Bloom*" CoNEXT'14, Dec 02-05 2014, Sydney, Australia ACM 978-1-4503-3279-8/14/12.

[2] Hyesook Lim, Senior Member, IEEE, Kyuhee Lim, Nara Lee, and Kyong-Hye Park, Student Member, IEEE "*On Adding Bloom Filters to Longest Prefix Matching Algorithms*" IEEE TRANSACTIONS ON COMPUTERS, VOL. 63, NO. 2, FEBRUARY 2014.

[3] JánosTapolcai, Member, IEEE, Josef Biro, Member, IEEE, PéterBabarczi, Member, IEEE, AndrásGulyás, ZalánHeszberger, Member, IEEE, and Dirk Trossen "*Optimal False-Positive-Free Bloom Filter Design for Scalable Multicast Forwarding*" IEEE/ACM TRANSACTIONS ON NETWORKING 1063-6692 © 2014 IEEE.

[4] Natalia Castro Fernandes, Marcelo DufflesDonato Moreira, and Otto Carlos Muniz Bandeira Duarte "*An Efficient and Robust Addressing Protocol for Node Autoconfiguration in Ad Hoc Networks*" IEEE/ACM TRANSACTIONS ON NETWORKING, VOL. 21, NO. 3, JUNE 2013

[5] XiaohuaTian, Shanghai Jiao Tong University Yu Cheng, Illinois Institute of Technology "*Bloom Filter-Based Scalable Multicast: Methodology, Design and Application* "IEEE Network • November/December 2013.

[6] B. Fan, D. G. Andersen, and M. Kaminsky. MemC3: "*Compact and concurrent memcache with dumber caching and smarter hashing*." In Proc. 10th USENIX NSDI, Lombard, IL, Apr. 2013.

[7] Deke Guo, Yunhao Liu, Xiangyang Li, Panlong Yang, "*False Negative Problem of Counting Bloom Filter*", IEEE Transactions on Knowledge and Data Engineering, IEEE, vol.22, no.5, pp.651-664, 2010.

[8] Gianni Antichi, DomenicoFicara, Stefano Giordano, Gregorio Procissi, and Fabio Vitucci,University of Pisa" *Counting Bloom Filters for Pattern Matching and Anti-Evasion at the Wire Speed*"IEEE Network January/February 2009.

[9] F. Putze, P. Sanders, and S. Johannes, "*Cache-, Hash- and Space-Efficient Bloom Filters*" Experimental Algorithms (Springer Berlin / Heidelberg, 2007), pp. 108-121.

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

[11] KilianWeniger " PACMAN: Passive Auto configuration for Mobile Ad Hoc Networks" IEEE JOURNAL ON SELECTED AREAS IN COMMUNICATIONS, VOL. 23, NO. 3, MARCH 2005.

[12] S. Nesargi and R. Prakash, "*MANETconf: configuration of hosts in a mobile ad hoc network,*" in Twenty-First Annual Joint Conference ofthe IEEE Computer and Communications Societies (IEEE INFOCOM2002), vol. 2. IEEE, jun 2002.

[13] R. Pagh and F. Rodler, "*Cuckoo Hashing,*" Journal of Algorithms, vol. 51, no. 2 (May 2004), pp.122-144.