Full Function Firewalls and Fast Routing Query on MANETs

Ting Zhang
Information Systems Security
Concordia University College of Alberta

Dale Lindskog
Information Systems Security
Concordia University College of Alberta

Abstract

This paper proposes a routing storage and query mechanism for the ROFL firewall scheme, a mechanism that combines a hierarchical routing storage structure with a fast routing query method. ROFL, which stands for Routing as the Firewall Layer, is a firewall scheme designed mainly to prevent insider attacks on Mobile Ad Hoc Networks (MANETs). The proposed mechanism also extends the filtering capabilities of ROFL, allowing packet filtering based on most any traditional packet filtering criteria, e.g., source IP address, source and destination port numbers, TCP flags, etc.

1. Introduction

Mobile Ad Hoc Networks (MANET) are self-organized wireless networks which consist of an arbitrary set of independent nodes. It is an infrastructureless network in which each node plays the role of both router and host; each node may receive packets destined to them, but may also forward packets. No specialized network devices are required; furthermore, each node is independent and free to move around in the MANET. In certain circumstances, MANETs have significant advantages over traditional layered network structures, including intrinsic flexibility, ease of maintenance, infrastructure independence, auto-configuration and self-organization. On the other hand, MANETs also have intrinsic vulnerabilities or limitations, such as limited bandwidth and (normally) battery life, need for computing efficiency, and certain intrinsic weaknesses in security and reliability. Of particular interest is the fact that many traditional security control methods are inappropriate for MANETs, due to their lack of infrastructure and their dynamically changing network topology. Firewalls are a case in point.

Conventional firewalls rely for their effectiveness on fixed network topology and controlled entry points. The assumption is usually that everyone on one side of the entry point, the firewall, is to be trusted, and that anyone on the other side is, at least potentially, an enemy [1]. But in MANETs network topology is completely dynamic: nodes in MANETs are free to move and the topology changes accordingly. For this reason, traditional firewalls are problematic.

Many have proposed solutions to enhance firewalls for MANETs. One very important proposal is the distributed firewalls architecture [1]. In this architecture, all nodes implement a firewall. A central management system distributes policies to these hosts, and packets are accepted or rejected by each host according to those policies. ROFL (Routing as the Firewall Layer) is based on distributed firewalls, but is designed to address the bandwidth consumption concern with distributed firewalls. More specifically, with distributed firewalls filtering occurs at the destination host. While this is not a problem from a host penetration perspective, it is problematic in bandwidth-constrained environments such as MANETs, or in the face of denial-of-service attacks [2]. ROFL combines conventional firewalls and distributed firewalls. It provides a tradeoff between cost and risk. The main idea of ROFL is that packet filtering is a form of routing: in ROFL, port numbers are treated as part of the IP address. Port-specific routing announcements in effect identify which ports are accessible on specific nodes. When packets are destined for inaccessible port numbers, the packet will be dropped, simply because there is no route to that destination IP address and port. As unwanted traffic is blocked closer to source host, bandwidth is conserved.

ROFL has also been extended to include source prefix filtering (SPF). SPF constraints in ROFL control route propagation and packet forwarding. A ROFL announcement is passed to a node only if that node is authorized to access the advertised service. During the packet forwarding phase, a packet is dropped immediately if it is coming from a source address not specified in the SPF constraints of a matching route [2]. One effect of ROFL is that routing tables will grow in size; another related effect is that there will be a considerable amount of redundancy in those routing tables. For example, when a node receives a routing announcement which includes a set of source prefixes, then, if there are e.g.

10 elements in this set (i.e. if 10 source addresses are specified), the node needs accordingly to add 10 entries in its routing table. As the size of the routing table grows, more CPU time is needed for routing queries. This is problematic, and especially so in MANETs, where no specialized infrastructure is used for route storage and query, and where computing resources are often highly constrained. We propose an efficient routing entry storage structure combined with bloom filters to achieve high efficiency and low redundancy. This mechanism m scales well, and is consistent with full packet filtering functionality based on any filtering criteria used in traditional firewalls. We propose a hierarchical structure for routing tables: a main table contains destination IP addresses and other basic information, similar to a regular routing table. When firewalling is required for a specific IP address, instead of adding entries to the main table, a sub-table will be created and chained to this entry. The sub-table contains a set of entries, the nature of which depend upon the filtering requirements. It can be a table of port numbers, source IP addresses, flags, etc. If further filtering is again required, another table is in turn chained to an entry of the sub-table. This process continues until all filtering requirements are specified. Furthermore, when a MANET node acting as a router queries its routing tables, a bloom filter is used to efficiently determine whether an element is included in a table. When it is not included, packets can be dropped immediately without further checking through all entries in that table.

In Section II of this paper, we introduce the reader to related research, focusing on distributed firewalls, ROFL, its extension to include source address prefixes, and finally bloom filters. Section III introduces our hierarchical routing table structure and query method. Section IV estimates performance of the proposed mechanism, where battery life and CPU time are used as scalars. Finally, in Section V, we conclude the paper.

2. Review of related research

Our work is based on distributed firewalls, ROFL and its extension to include source prefixes. Bloom filters are also an important element for our proposed firewall mechanism. In this section we describe these technologies.

2.1. Distributed firewalls

Distributed firewalls were first proposed in 1999, by S. M. Bellovin. Traditional boundary firewalls are not suitable for MANETs because of the rapidly changing network topology. Distributed firewalls were proposed in order to divorce firewalls from their traditional dependency on network topology. For this reason, they are particularly suitable to MANETs.

Distributed firewalls use IPSEC, a policy language, and system management tools. Each node is protected by its own firewall; the system management software creates policies and distributes policies to all hosts implementing a firewall. In the context of MANETs, distributed firewalls have many advantages compared with traditional firewalls. Some of the advantages of distributed firewalls are:

- Traffic between local hosts can be subject to centrally controlled packet filtering rules.
- Policies can be specific according to different needs of different hosts.
- There is no single choke point. Filtering takes place at each node.
- It can provide local host identification by certificate. Spoofing attacks will be prevented.
- It can protect hosts that are not within a topological boundary.

2.2. Routing as Firewall layer (ROFL)

Although distributed firewalls provide great protection for MANETs, they waste bandwidth and computing resources. Unwanted traffic flows all the way to the destination before being discarded. To solve this problem, a scheme called ROFL was designed. ROFL dynamically controls which paths traffic should take to a destination, while a firewall dynamically controls what is blocked [2]. Like the distributed firewall architecture, policies can be enforced at end hosts; alternatively, they can be enforced at intermediate nodes acting as protective gateways [2]. To achieve this function, ROFL treats port numbers as a part of the IP address. A conventional routing advertisement is of the form

$$R = \{p/m, M\}$$

where p is an address prefix, m is a prefix length, and M is a routing metric. ROFL extends the prefix field to include a service s:

$$\{p: s / m, M\}$$

For example, to permit access to an SMTP server and a web server at 192.0.2.42, a node would originate the routes

{192.0.2.42: 25/48, M}, {192.0.2.42: 80/48, M}

Port-specific routing advertisements are handled just like any other routing advertisements: the source node and any intermediate routers do a longest-prefix match on the advertisement. If there is no matching route, the packet is dropped [2]. In effect, this is firewalling as a form of routing, and unwanted traffic is blocked closer to the source host.

2.3. Extended ROFL

ROFL was extended to support policy routing. Source prefix filtering is achieved by adding source IP address prefixes to the routing announcement. The announcement format of extended ROFL is:

$$R = \{d: s/ m, S, M\}$$

where S is defined as a set of source prefixes: $S = \{ , ,\ldots, \}$, where each $(0 \leq i \leq n)$ is a source address prefix. No source prefix constraint is specified if $S = \phi$ and, if $S = \{0/0\}$, all addresses are accepted.

Source prefix filtering (SPF) constraints in ROFL control route propagation and packet forwarding. A ROFL announcement is passed to a node only if that node is authorized to access the advertised service. During the packet forwarding phase, a packet is dropped immediately if it is coming from a source address not specified in the SPF constraints of a matching route. Moreover, this new ROFL scheme is conceivably capable of extension to a complete set of filtering functionalities characteristic of traditional firewalls [3].

2.4. Bloom filters

A bloom filter is mainly used for membership queries. It is a time and space-efficient query method that tests whether an element is a member of a set. A bloom filter can be used for various applications including IP address lookups and longest prefix matches.

An empty Bloom filter is a bit array of m bits, all set to 0. There must also be k different hash functions defined, each of which maps or hashes some set element to one of the m array positions with a uniform random distribution. To add an element, it is fed to each of the k hash functions to get k array positions. The bits at all these positions are then set to 1.

To query for an element, it is fed to each of the k hash functions, and the value is checked at the k array positions given. If at least one of these values is 0, then the element is not present in the set, and a negative response is reported. If all of the values are 1, then a positive response is generated. Note that a false positive is possible, since these k positions may have been set by during the insertion of some other elements.

Removing an element from a Bloom Filter is impossible, since resetting any of the k positions might lead to a false negative on a subsequent query for another element, which is prohibited by the no false-negatives property of the bloom filter [4].

The improvement of efficiency for queries using bloom filter has frequently been discussed, and many improvements over standard Bloom Filters have been proposed. In [5-7], a series of d-left hashing counting Bloom filers (d-LCBF) are proposed. d -left hashing is different from the standard Bloom filter essentially. Fingerprint and multiple hash functions are used to store an item. It has been shown that a d-LCBF can use less than half of the space of, yet with the same false positive rate as the counting Bloom filter. Though d -LCBF can be used in many applications, such as IP lookup [5], [8] argues that the variable length counting Bloom filter is about 1.7 times faster than standard Bloom filter.

Bloom filtering can be applied to improve IP address and prefix lookups, to determine whether a particular port number is a member of a set, and in general any query that benefits from the ability to determine whether a particular value is a member of a set; these are the uses we make of it in our mechanism.

3. Routing Storage and Query Mechanism

3.1. Chained tables structure

In our proposal, multiple tables are used to implement full function firewalling. We call it CTS (chained tables structure). Regularly, on a router or host, all routing information is stored in one table called a routing table. When a network gets bigger or more complex, there can be a corresponding increase in the number of entries in the table. When extra information is required for implement firewalling as a form of routing, as with ROFL, the routing table size increases dramatically. Obviously, to query a route from a large table takes more time and computing resources. In our mechanism, we use multiple tables to store routing information. A main table supports destination based routing queries, while sub-tables are chained to that main table, or to other sub-tables, in order to support filtering based on criteria other than destination IP addresses.

With ROFL, routing announcements are only forwarded to nodes which are allowed to access a certain server. Unwanted traffic is blocked simply by not providing a route. However, in our mechanism, routing announcements can be sent to all nodes, and therefore all nodes understand the filtering rules so that unwanted traffic can be dropped at any point on the route from source to destination.

We use an example to demonstrate this structure. Consider the network shown in Figure 1, where node E is announcing services α and β. Node A is not allowed to access either service. Node B is allowed to access α and node C is allowed to access β. Node D is allowed to access both. For access to service α, clients must use a port number that is a member of a source prefix set; otherwise packets will be dropped. Here the set of source port numbers S has n members in it.

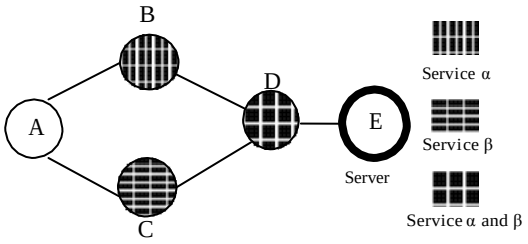

Figure 1. An example of MANETs

Main Table

Destination Address	*Destination Port NO.*	*Netmask*	*Next Hop/Chain*	*Metric*
E	α	/48	Table1	N/A
E	β	/48	Table2	N/A
B	N/A	/32	B	3
C	N/A	/32	C	5
A	N/A	/32	B	4
A	N/A	/32	C	4

Table 1

Source IP	*Source Port/Chain*	*Next Hop*	*Metric*
B	Table 3	N/A	1
D	Table 3	N/A	1

Table 2

Source IP	*Source Port/Chain*	*Next Hop*	*Metric*
C	N/A	E	1
D	N/A	E	1

Table 3

Source Port	*Next Hop*	*Metric*
S1	E	1
…	…	…
Sn	E	1

Figure 2. CTS for node D

The red lines in Figure 2 show how table 1 and table 2 are chained to the main table, and how table 3 is in turn chained to table 1. The main table contains destination IP addresses and port numbers, netmasks, and metrics, similar to a regular routing table. The difference is the next hop area in the main table, which does not contain only the next hop but may also chain to other tables. When node D receives a packet; it consults its main table for a matching destination IP prefix. If a matching entry is found containing a chain in the Next Hop/Chain column, the route lookup function on the node jumps to the chained table.

3.2. Faster routing queries

For each table, whether it is the main table or a chained sub-table, we propose that instead of iterating through table entries, bloom filters are used as a first step for routing lookups. Stored entries are accessed only when bloom filter result shows that the examined information is a member of the set of the table. Figure 3 shows the process of using bloom filter to query routing information.

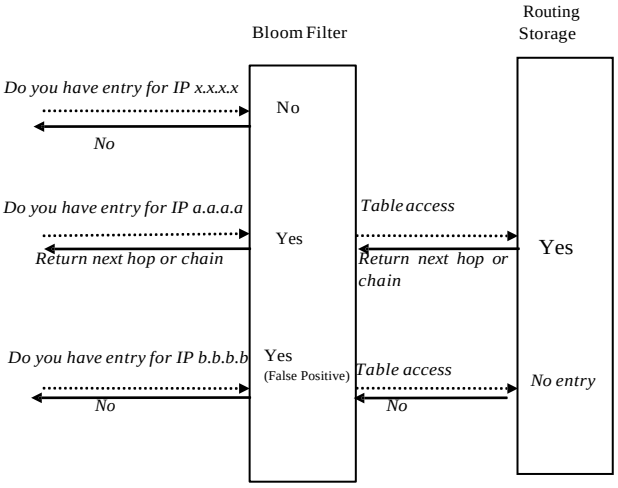

Figure .3. Bloom filter query process

We can illustrate the role of bloom filters in our mechanism by considering the network shown in Figure 1. Suppose node D receives a packet from B to access service α running on node E. The routing query process will be as follows.

1) Receive the packet and examines the IP header field.

2) Query main table for E's IP address (Destination IP) and α's port number (Destination port) using a bloom filter.

3) Bloom filter returns "true".

4) Find the entry from the main table; find a chain pointing to table 1.

5) Query table 1 for B's IP address (Source IP) using a bloom filter.

6) Bloom filter returns "true".

7) Find the entry from table 1; find a chain pointing to table3.

8) Query table 3 for B's port number (Source Port) using a bloom filter.

9) Bloom filter returns "true".

10)Find the entry from table 3; find next hop.

11) Forward the packet.

Bloom filter produce no false negatives, but there is a small probability of false positives. Occasional false positives do not greatly affect our mechanism. The reason is, as shown on Fig.3, that when a false positive occurs, the routing table will be accessed and all entries within the table will be checked. Since this is a false positive, no matching entry will be found, and the packet will be dropped accordingly. This is a minor waste of resources, and, there are enhanced bloom filters that decrease the chance of false positives. The weighted bloom filter is one of an example. As explained in [9], under reasonable frequency models such as step distribution or Zipf's

distribution, the improvement of the false positive probability of the weighted Bloom filter over that of the traditional Bloom filter has been shown in simulations.

3.3. Full function firewalls

The previous example shows how CTS and bloom filters work for our proposed mechanism. Note that this mechanism is able to chain as many tables as needed, and thus it can support fully the functions supported by traditional firewalls. For example, if filtering on TCP flags is desirable, a table containing entries for flags can be added and chained to the CTS.

4. Evaluation

In this section, we evaluate the performance of our proposed mechanism. We focus on routing table storage size and CPU time for route lookups. We estimate routing table storage size based on an assumed network, and evaluate CPU time using appropriate algorithms.

4.1. Routing table storage

We used node D from our example to evaluate routing information usage. Referring to the documentation on the Cisco Nexus 7000 Series [10], we allocated 144 bits for each entry in main table and 72 bits for each entry in chained tables. We also assumed that there were 50 members of the set of source ports. In other words, we allowed only 50 defined port numbers to be used by clients to access service α.

A table driven routing protocol is used. Each node must retain the latest routing table in the local network, and so, when the network topology changes, routing changes must be sent to each node, in order to obtain the consistent and up-to-date network routing information. Since, in our estimated network, every node has full understanding of the entire network, the size of CTS will be very close on every node.

For node D, the calculation of CTS size is shown in Table 1.

Table 1. Calculation of CTS size

Size of CTS				
	Main Table	**Table 1**	**Table 2**	**Table 3**
Number of entries	6	2	4	50
Size of an entry	144bit	72bit	72bit	72bit
Size of table	864bit	144bit	288bit	3600bit
Total size	4896bit			

Let us compare this with a flat routing table containing the same information. Again we allocated 144 bits to each entry. The routing table is abbreviated in Table 2.

Table 2. Regular routing table

Regular Routing Table							
ID	**D IP**	**D Port**	**Netmask**	**S IP**	**S Port**	**Next Hop**	**Metric**
1	E	α	/48	B	S1	E	1
…	…	…	…	…	…	…	…
50	E	α	/48	B	S50	E	1
51	E	α	/48	D	S1	E	2
…	…	…	…	…	…	…	…
100	E	α	/48	D	S50	E	2
101	E	β	/48	C	N/A	E	3
102	E	β	/48	D	N/A	E	3
103	B	N/A	/32	N/A	N/A	B	6
104	C	N/A	/32	N/A	N/A	C	5
105	A	N/A	/32	N/A	N/A	B	4
106	A	N/A	/32	N/A	N/A	C	4

As we can see from table 2, the number of entries is 106. When a same 144 bits are allocated to each entry, the total size of the table is 15264 bits, which is approximately three times that of the CTS.

In this example using a CTS can save memory. However, a CTS is not always smaller than a regular routing table. It depends on how many entries are included in chained tables. The more entries, the greater are the savings. That is to say, the CTS works better in cases where filtering rules are specific and detailed.

4.2. CPU time

A much more dramatic illustration of the improved efficiency of using CTS can be seen when we compare CPU time. There are two reasons why our mechanism conserves CPU time. Firstly, bloom filters conserve CPU time in those cases where there is no route for the desired destination, since in such cases, no table needs be accessed at all, but rather, the only query is against the bloom filter. Packets will therefore be dropped immediately following that single query, since routing tables are only accessed when the query returns true. For example, suppose node D receives a packet, sourced from B and destined to E, using source port number 51 to access service α. With a bloom filter, no table entry needs to be read, and the packet is dropped right after the bloom filter query. On the other hand, with a regular routing table, and without a bloom filter query (or related mechanism), the entire table of 151 entries must be checked, in order that it may be determined that none of those 151 entries match.

Secondly, categorizing information hierarchically makes searches much more efficient. In a regular routing table, information is stored together in a flat table. When a specific entry is queried, all entries

stored before that matching entry must first be read. Whereas with our structured storage, only some number of related entries need to be examined. Let us also illustrate the efficiency of structured routing entry storage with an example. Suppose that node D receives a packet, sourced from C and destined to E to access service β. With a regular routing table like that shown in Table 2, a full 101 entries need to be read prior to finding the route; but when using CTS, this number decreased to 3. Although the number of entries read in CTS is not always less than when using a regular routing table, for example when a packet is sent from node B to access service α on node E, it is quite evident that in an actual MANET with many nodes, the average number of entries read will be far less using CTS than with a traditional routing table.

Let us attempt to give a more precise specification of how our mechanism saves CPU time. We will again use Node D in Figure 1 to illustrate the savings in CPU utilization. As CPU hardware is varied, and since different operating systems handle CPU usage differently, of course we cannot specify an exact number to use in calculating CPU time. However, the performance of longest prefix matching can be determined by the number of dependent memory accesses per lookup [11].

As mentioned previously, a bloom filter uses a bit array in which one or more bits map to one element of some set, and it is an examination of these bits that is used to determine class membership. The number of bits associated with one element of the set is determined, in particular cases, by the number of hash functions. Suppose, for example, that two hash functions are used; this maps two bits from the bit array to each element in the set. In other words, each routing entry, including both those in the main table and those in each sub-table, takes up two bits from the bloom filter bit array. Therefore, and continuing with the same example network, the size of the bit array is 124 bits, since the total number of CTS routing entries is 62.

With this calculation as background, let us now compare CPU time utilization for the three possible results from the bloom filter query. Generally speaking, there are three possibilities with respect to the bloom filter query that will occur when node D receives a packet. One possibility is that a matched entry is found; a second is that no matched entry is found; and, thirdly, the bloom filter query may return a false positive. Recall that, for CTS, the entry size in main table is 144 bits, and is 72 bits in chained tables, while for a regular routing table all entries are 144 bits in size.

Let us consider the first possibility: a matched entry is found, i.e., the bloom filter query returns a true positive. Let us suppose that node D receives a packet from B to access service α on node E using source port number 50. With CTS, a route query will consist of a bloom filter query, one entry read from the main table, and 50 entries read from sub-tables. In contrast, with a regular routing table, 50 entries must be read. The calculation is shown below, in Table 3.

Table 3. Comparation: matched routing entry exist

	CTS			Regular Table
	Bloom filter	Main Table	Chained Tables	Routing Table
Number of Entries Reading	1	1	50	50
Size	1*124	1*144	50*72	50*144
Total	3868 bits			7200 bits

Table 3 shows that, using CTS, a maximum of 3868 bits of data need to be processed, whereas with a regular routing table the number is about two times that.

Let us now consider the second possibility: a packet arrives which does not have a matched entry, and the bloom filter query return value correctly indicates this fact, i.e., the query returns a true negative. When this occurs, CTS requires only that the bloom filter bit array be read, while with a regular routing table the entire table must be read. Here the advantage of CTS is much more drastic. The calculation is shown in Table 4.

Table 4 Comparation: no matched routing entry

	CTS		Regular Table
	Bloom filter	Main Table	Routing Table
Tims	1	0	106
Size(bits)	1*124	0	106*144
Total	124 bits		15264 bits

As shown in Table 4, CTS requires that a maximum of 124 bits be read. With a regular routing table, 15264 bits of data must be read, which is over 123 times the 124 bits read using CTS.

Consider finally the last possibility, where the bloom filter query returns a false positive: a packet without a matched entry in the routing table arrives, but the bloom filter query returns true. Here the result will be roughly the same as when using a regular routing table. With CTS, both the bit array and the entire main table have to be accessed. Table 5 shows the calculation.

Table 5 Comparation: false positive

	CTS		Regular Table
	Bloom filter	Main Table	Routing Table
Tims	1	6	106
Size(bits)	1*124	6*144	106*144
Total	988 bits		15264 bits

As can be seen from Table 5, the case of the false positive requires that, using CTS, 988 bits of data must be processed. This number is still smaller than using a regular routing table. Therefore, even in this worst case scenario, where CTS uses a bloom filter query and, because that query returns a false positive, a full routing table reading is also required, CPU time is still less than with a regular routing table.

From the calculations above it can be seen that CTS normally requires less, and often considerably less CPU time than using a regular table. Of course, were the number of hash functions or number of routing entries different than in our example, the calculation would show different results. It is clear however that, in general, and especially as the number of routes grows, CTS will greatly reduce the usage of CPU time.

5. Conclusion

In this paper, we propose a routing storage and query mechanism for ROFL, a scheme proposed for implementing firewalls in MANETs. Our mechanism combines a chained table structure and bloom filters. This mechanism in principle supports all the functions of traditional firewalls, and has a number of advantages when used in MANETs. In particular, it reduces routing table storage requirements, and reduces CPU time needed for route lookups. This is especially important for MANETs because resource constrained nodes, rather than dedicated devices, must do the routing and firewalling.

6. References

[1] S. M. Bellovin, "Distributed firewalls," Journal of Login, vol. 24, pp. 37-39, 1999.

[2] H. Zhao, C.-K. Chau, and S. M. Bellovin, "ROFL: Routing as the firewall layer," in Proceedings of the 2008 workshop on New security paradigms, 2009, pp. 23-31.

[3] H. Zhao and S. M. Bellovin, "High performance firewalls in MANETs," in Mobile Ad-hoc and Sensor Networks (MSN), 2010 Sixth International Conference on, 2010, pp. 154-160.

[4] R. Bhoraskar, V. Gabale, P. Kulkarni, and D. Kulkarni, "Importance-aware Bloom Filter for managing set membership queries on streaming data," in Communication Systems and Networks (COMSNETS), 2013 Fifth International Conference on, 2013, pp. 1-10.

[5] A. Broder and M. Mitzenmacher, "Using multiple hash functions to improve IP lookups," in INFOCOM 2001. Twentieth Annual Joint Conference of the IEEE Computer and Communications Societies. Proceedings. IEEE, 2001, pp. 1454-1463.

[6] F. Bonomi, M. Mitzenmacher, R. Panigrah, S. Singh, and G. Varghese, "Beyond bloom filters: from approximate membership checks to approximate state machines," in ACM SIGCOMM Computer Communication Review, 2006, pp. 315-326.

[7] F. Bonomi, M. Mitzenmacher, R. Panigraphy, S. Singh, and G. Varghese, "Bloom Filters via d-Left Hashing and Dynamic Bit Reassignment Extended Abstract," in Forty-Fourth Annual Allerton Conf., Illinois, USA, 2006, pp. 877-883.

[8] L. Li, B. Wang, and J. Lan, "A variable length counting Bloom filter," in Computer Engineering and Technology (ICCET), 2010 2nd International Conference on, 2010, pp. V3-504-V3-508.

[9] J. Bruck, J. Gao, and A. Jiang, "Weighted bloom filter," in Information Theory, 2006 IEEE International Symposium on, 2006, pp. 2304-2308.

[10] Cisco, 'Cisco Nexus 7000 Series NX-OS Unicast Routing Configuration Guide, Release 5.x.' Available:http://www.cisco.com/en/US/docs/switches/datacenter/sw/5_x/nx-os/unicast/configuration/guide/l3_manage-routes.pdf. (20 July 2011).

[11] S. Dharmapurikar, P. Krishnamurthy and D.E. Taylor, " Longest Prefix Matching using Bloom filters ," Proc. ACM SIGCOMM, pp. 201-212, 2003.

