This full text paper was peer reviewed at the direction of IEEE Communications Society subject matter experts for publication in the ICC 2007 proceedings.

Mesh network firewalling with Bloom Filters

Leonardo Maccari, Romano Fantacci
Department of Electronics and Telecommunications
University of Florence - Telecommunication Network Lab
tel. : +390554796467 Florence, Italy
Email: {maccari,fantacci}@lart.det.unifi.it

P. Neira, R.M. Gasca
Departament of Languages and Computer Systems
University of Sevilla - QUIVIR LabI2S
tel. +34954559814 Sevilla, Spain
Email: {pneira,gasca}@lsi.us.es

***Abstract*— The nodes of a multi-hop wireless mesh network often share a single physical media for terminal traffic and for the backhaul network, so that the available resources are extremely scarce. Under these conditions it is important to avoid that unwanted traffic may traverse the network subtracting resources to authorized terminals. Packet filtering in wireless mesh networks is an extremely challenging task, since the number of possible connections is quadratic with respect to the number of the terminals of the network; for each connection a rule is needed and the time needed for filtering grows linearly with the number of rules. Moreover nodes can be in possession of end users and the administrator might want to keep the explicit ruleset as much secret as possible while giving the nodes enough data to behave as a firewall. In this article we present a solution for distributed firewalling in multi-hop mesh networks based on the use of Bloom Filters, a powerful but compact data structure allowing probabilistic membership queries.**

I. Introduction

A multi-hop wireless mesh network is a network composed of peer terminals that communicate using a locally shared physical media and some ad-hoc routing protocols. The novelty of mesh networks is that the communication standard used and the routing protocols are highly dynamical so that the network are self-organizing; no administrative action should be required to add or remove a terminal from the network (see [1] for a survey of available technologies). Such a dynamic instrument perfectly suites the requirements of critical scenarios like emergency interventions in hostile environments, but also permits the creation of pervasive networks of services that may enlarge without the need of expensive infrastructures. In any of these scenarios the security of the communications and the availability of the service is essential, and wireless mesh networks have peculiar problems compared to wired or wireless *infrastructured* networks due to the technology involved and to the different management scheme.

In general, wireless networks have no clear geographical border, so that any attacker in the coverage area can receive packets and try to inject packets into the networks. This second behaviour should be discouraged by the use of layer II cryptography and access control techniques, but the complexity of the mesh scenario has so far discouraged the use of security measures such as *IEEE 802.11i* standard; in wireless mesh the most performant solution still seems to be the highly vulnerable WEP standard (see [2]). This might give to the attacker the practical possibility of injecting packets into the network.

Under a management point of view mesh networks may include nodes that are not under the control of a single administrator, such as roaming users. In general, the manager of the network should consider the nodes untrusted, meaning that from an internal node could be produced attacks to the rest of the network. This might be a willing behaviour or, for example, a consequence of an infection of a virus. Nevertheless all the terminals should behave both as end client and as IP routers to create the multi-hop mesh. Since the use of highly dynamic MAC and routing protocols produces a constant signaling traffic reducing the available resources, the generation of unwanted traffic (i.e. denial of service attacks, or SPAM) may hardly hit the performance of the network.

In infrastructured networks unauthorized traffic is normally filtered with the use of a firewall: a firewall is a node that drops or forwards packets based on filtering rules, normally placed on the gateway of the network. In wireless mesh network the main interest is to reduce packets passing *through* the network, so that filtering rules should be present on every node of the mesh. This introduces two critical problems as we will explain in more detail in the following sections:

- The number of rules grows quadratically with the number of the terminals of the network and the complexity of filtering grows linearly with the number of rules.
- Each node of the mesh should be aware of the filtering policies of the network, while the administrator may want to keep this information as much reserved as possible.

In this paper we present a possible approach to solve this problem using Bloom filters. As we will see in following sections, Bloom filters are a compact way to represent a set of elements, the main interesting feature is that a query cannot produce false negatives, that is of great help applied to firewalling. To the authors' knowledge this paper is the first addressing the problem of firewalling in mesh networks.

The rest of the paper is organized as follows: section II introduces the basic concepts of firewalling, section III explains the theory behind Bloom filters, section IV explains in detail the proposed solution and section V reports the results

This work is partially supported by National project Wireless 8O2.16 Multi-antenna mEsh Networks (WOMEN) under grant number 2005093248, Spanish Ministerio de Educacin y Ciencia through a coordinated research project (grant DPI2006-15476-C02-00) and Feder (ERDF) and by the 2006 PRIN project PROFILES (PeeR-to-peer beyOnd FILE Sharing)

1-4244-0353-7/07/$25.00 ©2007 IEEE

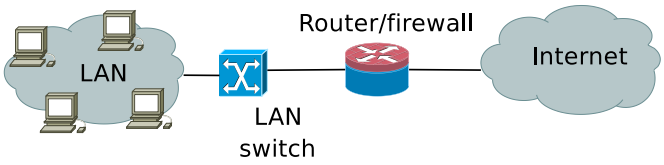

Fig. 1. Typical LAN scenario

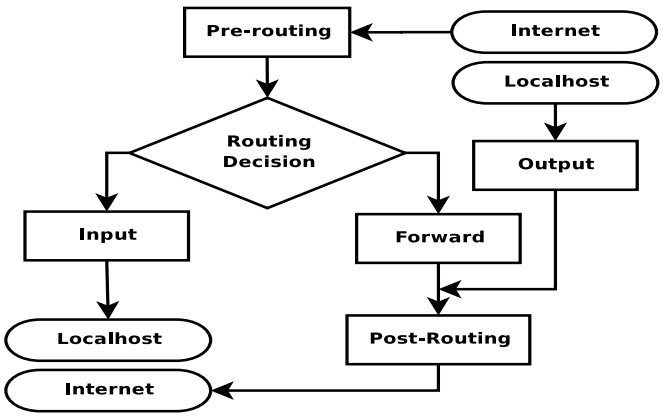

Fig. 2. Iptables filter structure, ovals contain source or destination and blocks contain *chains*

of practical experiments in real testbed. Section VI concludes the work and proposes further developments.

II. Introduction to Firewalls

A firewall is a router that selectively forwards packets based on the filtering rules that have been installed, as an example to review the features of a generic firewall we will describe the Netfilter/Iptables. This appliance is included with the GNU/Linux kernel, being it a client, a server or a transparent firewall and source code is available under a free license, these were the main reasons why we chose this platform for the implementation of our prototype.

As represented in fig. 1 a firewall is normally placed as a bastion host in the gateway of a LAN, its role is to control and limit the traffic coming inside the network and the traffic generated from the network to the Internet.

In fig. 2 it is described the flow of a packet that is traversing the Linux kernel, each of the blocks (named chains) represented in the figure are points in which the packet can be filtered based on the rules contained in the chain. Each chain has a different set of rules and may be executing more complex actions then simply dropping the packets, such as changing its contents (mangling) or marking it with internal flags to be used in successive chains.

When the firewall receives a packet coming from the Internet, it first crosses the *Pre-routing* chain, then, based on the routing decision goes into the *Input* chain; if the packet is not directed to the local host but to some other host the packet is sent to the *Forward* chain. On a bastion host firewall as depicted in fig.1 most of the packets pass through the *Forward* chain, since only a few packets will be directed to the firewall itself. After passing into the *Forward* chain, packets cross the *Post-routing* chain, together with any packet that has been locally generated and are directed to the Internet. It is important to note that packets that are going to the outside network take a partially different path in the kernel if have been locally generated (crossing the *Output* chain) or have been forwarded.

Every chain contains various filtering rules, every rule contains a pattern to be matched against the current packet and an action to be performed (normally: Accept/Deny), basically a rule can be imagined as follows:

```
if tcp destination port = 80 and source
IP = 150.217.10.8, accept the packet
```

Referring to the firewall in fig. 1, such a rule if placed in the *Forward* chain will accept any packet matching the pattern coming or going to the LAN, while if put in the *Input* will allow any packet matching the pattern directed *to* the firewall IP address. Rules are normally expressed in positive logic, that is, the default behavior is set to *drop* and each rule allows packets matching a pattern. If a packet matches anyone of the rules it is allowed, otherwise it is dropped. Positive logic has the advantage of being indifferent to the order of the rules, that makes management easier. Also, generally speaking, if the default policy is set to Accept, then it is easier to commit errors in the configuration and let unwanted traffic pass the firewall.

Iptables is a *stateful* firewall [3], that means that packet filtering is done not only parsing the contents of the packets but also taking into account the flow of data the packet belongs to. As an example a stateful firewall may allow a TCP packet with ACK and SYN flags activated only if there has been a TCP packet with SYN flag allowed before.

Two problems that firewalls have to face in modern networks and that are even more important in mesh networks are the following ones:

- Scalability problems: if each packet must be matched against a list of rules, the time required for filtering grows linearly with the number of rules. If a firewall contains thousands of rules performances drop significantly. Memory consumption is also a concern, therefore the data structure used to represent the filtering policy must do an efficient use of the resources available.
- Complexity of the topology: modern networks are composed of different subnets, each one composed of machines that offer or use complex services; describing such a scenario with filtering rules can be extremely challenging.

In [4], a model for a distributed firewall is proposed. In that scenario only the second issue is approached using policies represented with a high-level language that are then translated into single rules. The policies are distributed to the terminals and applied locally, creating a *distributed firewall*.

A. Mesh networks firewalling

In a mesh scenario, each node of the network may act as a client or as a server and, most of all, will forward traffic that

is not directed to its address to the neighbor machines. While the *Input* and *Output* chains depend on the kind of services the node uses we focus on the *Forward*, that determines which type of services are vehiculated in the network. The problem we want to address is how to perform packet filtering of IP addresses and TCP ports in each terminal, with a scalable solution that takes into consideration the problems introduced by the peculiar nature of mesh networks. We defined two main issues:

- If a positive logic is used, the number of rules adopted grows quadratically with the number of the nodes of the network. If we limit filtering to TCP ports and IP addresses, for each couple of machines into the network there must be at least two rules that explicitly allows bi-directional communications. If the network is composed of N machines, each one should contain $(N-1)(N-2)*s$ rules where s is the number of allowed tcp ports.
- As said, a mesh could be composed of terminals that do not belong to a single administrator: there might be roaming users or leased terminals to temporary end users. Under these circumstances the administrator of the network (we refer to the administrator as the entity administering the link with the external Internet) may want to hide as much as possible the policy used.

Example application are various, from traffic shaping (the administrator may want not to allow certain kind of traffic, i.e. peer to peer) or as reaction to attacks: imagine that a node in the network start performing a flood attack, or sending SPAM mail. The network might be able to detect the attack and reconfigure the firewall in order to isolate the attacker and limit the impact over the network.

Lastly we note that in a mesh network the firewall can not perform stateful filtering, since there is no guarantee that all the packet that constitute a data flow may use the same path to their destination.

III. Bloom Filters

A bloom filter is a space-efficient data structure for representing a set in order to support probabilistic membership queries. The probability of committing an error in a query might be limited choosing an appropriate size of the data structure compared to the size of the set of elements to be represented. The most important feature they present is that they only present false positives, that is, a query of the type *is element A part of the set B* will never produce a negative answer if $a \in B$, but may produce a positive answer if $a \notin B$. Bloom filters were introduced by Burton H. Bloom in [5], for an overview of their theory and their application in networking field see [6]. In the rest of this section we will recall the basic concepts.

A bloom filter is an array of bits of size n, to build the filter, each element of the set is hashed with a number K of distinct hash functions and the result of each hash is used as an index to set the corresponding bit in the filter. Referring to image 3 the filter is of size n and each hash function returns a value of size $ln(n)$, the corresponding subset of bits is turn to 1 in the

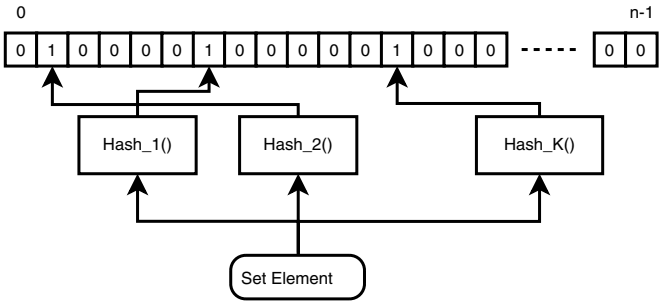

Fig. 3. Bloom filter generation

filter. Each time a new element is inserted in the filter, a new subset of bits is set to 1, with a certain probability of having two distinct elements generate two overlapping subsets. Once populated the filter with all the elements, to check the presence of an element in the set the same procedure is applied, it is hashed with the K hash functions and if all the bits of the subset were set to 1 the element belongs to the set. Since the subsets may be overlapping there is a certain possibility of false positives, but no false negatives are admitted. The probability of having a false positive if the original set is composed of m elements is given by the following equation:

$$f = (1 - e^{Km/n})^K \tag{1}$$

f is minimized if $K = ln(2) * n/m$ which gives:

$$f = (0.5)^k = (0.6185)^{n/m} \tag{2}$$

If we set $f < 0.1\%$ we have:

$$\begin{cases} n/m = \lceil log_{0.6185}(0.0001) \rceil = 20 \\ k = \lfloor log_{1/2}(0.6185^{20}) \rfloor = 13 \end{cases} \tag{3}$$

13 different hash functions and a filter size given by 20 bit per element of the set are needed. Given a set of 6500 elements, the size of the filter will be of approximately 16 kilobyte, and each hash function should produce a number big enough to be used as index for the filter that is $log_2(16KB) = 14$ bit. Instead of using 13 hash functions, different slices of the result of a unique pseudo random number generator applied to the element can be used. There is no need for cryptographically strong hash functions, the only required property is distribution over the range $\{0...n-1\}$.

IV. Mesh Firewalling with Bloom Filters

Figure 4 represents a mesh network scenario, in which different terminals are all equipped with routing capabilities and consequently, of firewall. In the scenario we consider the nodes under control of the network manager are one (or more) gateway to the Internet and some fixed machine offering standard services (WEB, DB ...) under control of the network manager. The rest of the terminals behave as clients, contacting servers placed both outside or inside the network or even as servers, offering services to hosts both outside or inside the network. Our solution is based on the use of Bloom filters

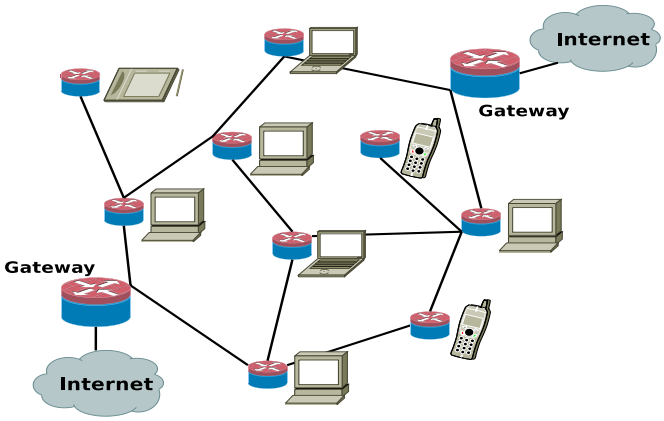

Fig. 4. Mesh network

to implement packet filtering based on source and destination TCP ports and IP addresses, so the administrator has control over the services transported by the network. It is optimized for a class C size network, but the same approach can be extended to wider networks or protocols other than TCP.

Imagine that a firewall rule is represented by 4 values (two IP addresses and TCP ports), so that each rule explicitly allows communications between two IP addresses over certain services, in positive logic. The rule-set is composed of all the allowed combinations, once decided which connections are allowed, it will be composed of a number n of vectors of the form $R = \{Source_{IP}, Dest_{IP}, Source_{port}, Dest_{port}\}$. A Bloom filter is populated using an appropriate dimension and number of hash functions and distributed to each host of the network. Whenever a host receives a packet it has to forward, in the forward chain of figure 2 it will create vector R with the fields of the packets and evaluate it against the Bloom filter in its possession. If R belongs to the set of allowed packets it will forward the packet otherwise it will drop it. Since Bloom filters present only false positives and the firewall rules are expressed in positive logic, the main risk is that a packet that should not be forwarded will be accepted. The likelihood of this event depends on the size of the chosen filter, and could be reduced if different filters are used for different hosts, as explained later. Instead in presence of false negatives, the network would unpredictably discard packets making the system unmanageable.

Such a system presents the following advantages:

- The cost of evaluation for each packet is at most the cost due to the computation of K hashes. When performing a query, each hash is computed separately, and after each computation the corresponding bit is checked in the filter. All the K functions must be computed only if the query result is positive, otherwise the packet can be dropped after the first failure.
- Only the filters are distributed to the nodes, not the explicit rules. Depending on the dimension of the rule-set a brute force attack may be possible, but this is actually unavoidable with any distributed filtering technique.

Traffic	Src_{IP}	Src_{port}	Dst_{IP}	Dst_{port}
EXT→ EC	EXT	low	EC	hi
EXT→ ES	EXT	hi	ES	low
EC→ EXT	EC	hi	EXT	low
ES→ EXT	ES	low	EXT	hi
IC→ IS	IC	hi	IS	low
IS→ IC	IS	low	IC	low

TABLE I

ACCEPTED TRAFFIC STREAMS: EXT IS USED FOR IP ADDRESSES NOT BELONGING TO THE LOCAL LAN

The main disadvantage of this simple approach is represented by the size of the filter. Considering only internal traffic of a class C network with 254 hosts, and the total number of ports, the maximum number of rules is given by all the possible combinations of IP addresses and ports: $(254^2) * (65535^2) \simeq 2.8E14$. Using optimal K, and 20 bit per element the size of the filter is approximately 600 terabyte, that makes the filter unusable. It should be noted that even with such an enormous filter the computational cost would be limited to the cost of the hash, but the filter would be impossible to store or transmit. To keep the size of the filter smaller we need to split the problem into smaller ones, clustering the packets in different streams and applying different filters to each stream. To find a viable solution we fixed the maximum false positive rate to 0.1% (we will show how this is enough for an effective filtering) and a maximum size of the filters to less then 30 kilobytes. Note that the whole filter should be transmitted only at network entrance, whenever a new rule is added to the filter (that may be a consequence of entrance of a new host) the information needed to update the filter are K numbers of size $log_2(n)$ bit.

To separate traffic flows we decided to divide the hosts into classes with specific capabilities:

- **Internal Server (IS):** The host behaves as a server for clients belonging to the network.
- **Internal Client (IC):** The host behaves as a client for servers belonging to the network.
- **External Server (ES):** The host behaves as a server for clients outside the network.
- **External Client (EC):** The host behaves as a client for servers outside the network.

The accepted traffic flows are detailed in table I , classified by IP and TCP port (*low* and *hi* represent TCP port below and over 1024). This is just one of the possible way of clustering packets into streams, we believe this classification should be expressive enough to represent the most common situations.

Note that a packet can be classified without knowing the IP addresses belonging to each class but basing the decision only on TCP ports and the subnet mask of the network, used to distinguish between internal and external hosts.

In the *Pre-routing* chain of Iptables, we introduced six rules in order to mark packets belonging to the different streams, once arrived in the *Forward* chain each stream is filtered with a different Bloom filter. The great advantage is that each filter

Streams	Chain	$\mathrm{Src_{IP}}$	$\mathrm{Src_{port}}$	$\mathrm{Dst_{IP}}$	$\mathrm{Dst_{port}}$
1a) EXT→ EC	P	EXT			hi
	F		low	EC	
2a) EXT→ ES	P	EXT	hi		
	F			ES	low
1b) EC→ EXT	P		hi	EXT	
	F	EC			low
2b) ES→ EXT	P			EXT	hi
	F	ES	low		
3a) IC→ IS	P		hi		low
	F	IC		IS	
3b) IS→ IC	P		low		low
	F	IS		IC	

TABLE II

ACCEPTED TRAFFIC STREAMS: P IS USED FOR RULES IN THE *Pre-routing* CHAIN AND F FOR RULES IN THE *Forward*. THE APPLICATION OF EACH COUPLE OF RULE GENERATES THE STREAMS OF TABLE I

Host Number	IC	IS	EC	ES	filter size		Iptables Rules	
80	20	20	20	20	1000	1000	400	400
160	40	40	40	40	2000	4000	800	1600
240	60	60	60	60	3000	9000	1200	3600
320	80	80	80	80	4000	16000	1600	6400
400	100	100	100	100	5000	25000	2000	10000
320	140	20	140	20	4000	7000	1600	2800
320	120	40	120	40	4000	12000	1600	2800

TABLE III

BLOOM FILTER SIZE ESTEEM AND COMPARISON WITH IPTABLES

will be applied only to a subset of the 4 fields, therefore the sum of the sizes of each filter will be much less than the size of a single filter.

In table II are reported which fields will be used for clustering and which fields will be used for bloom filters, the combination of the two phases leads to bloom filters applied to at most two fields in vector R. Note that a bloom filter is never used to filter external IP addresses or TCP ports over 1024, which are the larger but less relevant sets. Also note that host of the network may belong to more than a single class, and have multiple functionalities.

Each filter is repeated twice applied to different fields, so that the six streams can be filtered with only 3 bloom filters (filters 1a,1b and 2a,2b 3a,3b are equivalent and can be referred to simply as filter 1,2,3 respectively). The size of the bloom filters largely depends on the size of the single classes over the total 254 hosts. In a realistic scenario, the total number of services provided by the network will cover a little part of the 1024 TCP port (we chose 10) so that to have a false positive rates below 0.1%, given equation 3 the sum of sizes of filters 1 and 2 will be $10*(EC+ES)*20$ bit. The third filter is the larger one, its size will be $20*IS*IC$. If all the hosts of the network belong to all sets,that is the worst case, the total size is $10*(254*2)*20+(254*254)*20 \simeq 180Kbytes$, which is a usable size but still far from our target. In table III we propose some realistic scenarios with different distribution of sizes for each class and we compare our solution with the same filtering strategy using Iptables; it should be noted that:

- first column report the sum of columns (2-4) corresponding to the size (number of hosts) of each class. A total size larger than 254 is to be intended as a larger network or as a network in which the host classes are overlapping.
- column 6 reports the sum of the size (bytes) of filters 1,2, while column 7 reports the size of filter 3.
- column 8 and 9 report the number of iptables rules that would be required to substitute bloom filters 1,2 and 3 respectively.
- while in the first lines classes are equal in size, in the last two lines a more realistic scenario is reported, in which clients are more than servers.
- for all the combinations reported, total size of Bloom filters is always below 30 kilobytes.

The filters could be distributed at network entrance, after the operations of authentication to the network, together with other accounting credentials and can be updated with broadcast packets upon changes of topology. If a number F of distinct filters is used, that is, each node at network entrance will receive one over F filters (note that this produces higher traffic only when refreshing nodes), then the probability of having a packet filtered with the same filter along all its path is given by $(1/F)^p$ where p is the length of the path. In an open air scenario, in which each node of the network is placed in a squared grid and may possess at most 4 neighbors, with a two hop path at most 12 hosts are reachable. To reach the remaining 242, at least a three hop path is necessary involving 2 filtering nodes; we assume that two different filters generate different false positives each one with probability P_{false}. Being P_{false}=1/1000, and 242/254 the probability of a two-hop path the chance of having a false positive along the whole path is less then $P_{false}*(1/\mathrm{F})^2 = 4E-5$ and drops exponentially with p, which we consider acceptable.

V. TESTBED RESULTS

The implementation has been realized on a PIII 866 MHz running version 2.6.16 of the GNU/Linux kernel and version 1.3.4 of the firewalling framework *iptables* [7]. Our implementation has been realized as a kernel module implementing the necessary changes to use Bloom filters. Source code is available at http://people.netfilter.org/pablo/bloom-experiments/.

We chose Iptables because it is embedded in many commercial devices (i.e. wireless routers and mobile phones) and for the availability of the source code and free documentation.

Our testbed is composed of three hosts A, B and C connected with a 100Mbit Ethernet link. C acts as gateway between A and B and consequently filters forwarded traffic based on the information contained in three Bloom filters as we described in the previous section. On the other hand, the hosts A and B generate TCP traffic by means of the tool *Netperf* [8]. Host C has been loaded first with Iptables rules and then with Bloom filters, in both cases they represent the first five scenarios of table III. When using Iptables, the generated traffic is matched by rules that are placed in the middle of the list, so that the average case is evaluated.

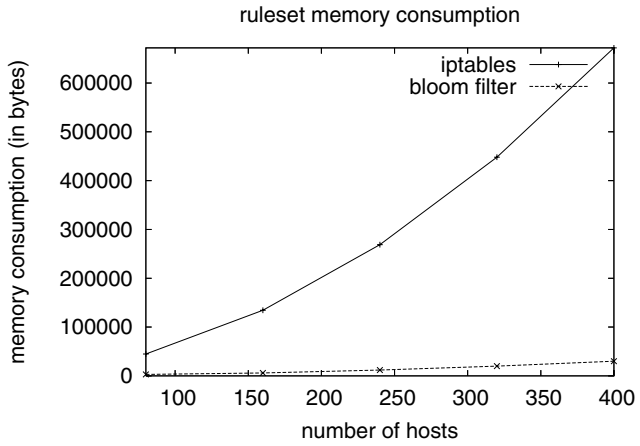

Fig. 5. Memory consumption

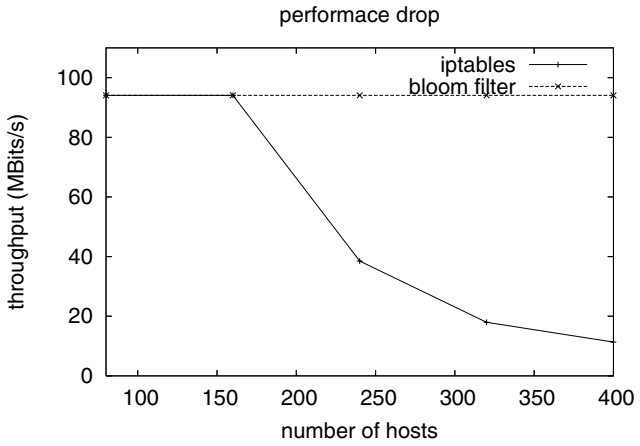

Fig. 6. Performance Drop

Experimental results obtained have been expressed in terms of memory consumption and network performance drop using the settings detailed in the first five rows of the table III.

A. Memory consumption

One of the reasons why the existing filtering solutions does not suite well for mesh networks is memory consumption. Due to the fact that devices that belong to a certain mesh network can be of different sorts, from embedded devices with limited resources available to personal computers, the filtering solution must be memory-efficient. Moreover, it must scale well with regards to the number of hosts that compose the network because new machines means new extra rules.

The size of one iptables rule that only uses the information contained in the TCP/IP header and the associated action (accept or drop) is *56 bytes*, every extra rule requires such amount of bytes. As said memory consumption order for iptables is $O(n)$ where n are the number of rules. On the other hand, the amount of memory required to represent a limited number of rules in a Bloom filter of a fixed size is $O(1)$, although we have to choose the appropriate size depending on the number of hosts that the network will hold.

The results obtained show that Bloom filter approach needs much less memory than iptables (Fig. 5) with better scalability if the number of hosts that join the network increases.

B. Performance

The experimental results show that the cost of issuing 13 hash computations is negligible (Fig. 6), therefore, our solution has no impact on the performance in terms of CPU load, and consequently of throughput. Opposite to this we see how linear complexity impacts hardly the performance of plain Iptables, that drops to around 40% of throughput passing from a 160 to 240 host scenario and about 19% to the 310 host scenario.

VI. Conclusions

In this work we have introduced a novel architecture to enable filtering policies in mesh networks based on bloom filters. The experimental results show that the solution proposed reduces memory consumption and scale better than existing filtering solutions for a relatively large amount of machines. As possible improvements we mention:

- The use of compressed spectral filters that permit to remove objects from the set. With a little overhead at start-up to transmit the filter our analysis is still valid.
- Packet caching, since packets from the same connection will be treated in the same way, keeping a cache of the last accepted packets can reduce time needed to compute the hash functions.

The use of Bloom filters as lightweight packet classifiers can be applied to other fields, our intention for future research is to evaluate their role to achieve QoS in mesh networks and generic packet filters in other kind of networks, such as overlay peer-to-peer distributed networks.

References

[1] W. W. Ian F. Akyildiz, Xudong Wang, "Wireless mesh networks: A survey," *Elsevier Computer Networks The International Journal of Computer and Telecommunications Networking Computer Networks and ISDN Systems*, vol. 47, 2005.

[2] W. A. Arbaugh, N. Shankar, and Y. C. J. Wan, "Your 802.11 wireless network has no clothes," May 15 2001. [Online]. Available: http://citeseer.ist.psu.edu/472552.html;http://www.drizzle.com/~aboba/IEEE/wireless.pdf

[3] G. Van Rooij, "Real stateful tcp packet filtering in ip filter," in *10th USENIX Security Symposium*, Washington, D.C, USA, aug 2001.

[4] Y. Bartal, A. Mayer, K. Nissim, and A. Wool, "*Firmato*: A novel firewall management toolkit," *ACM Transactions on Computer Systems*, vol. 22, no. 4, pp. 381–420, Nov. 2004.

[5] B. H. Bloom, "Space/time trade-offs in hash coding with allowable errors," *Communications of the ACM*, vol. 13, no. 7, pp. 422–426, 1970. [Online]. Available: citeseer.ist.psu.edu/bloom70spacetime.html

[6] A. Broder and M. Mitzenmacher, "Network applications of bloom filters: A survey," 2002. [Online]. Available: citeseer.ist.psu.edu/broder02network.html

[7] N. C. Team, "Iptables: The linux firewalling tool." [Online]. Available: http://www.netfilter.org/

[8] R. Jones, "Netperf: The network performance tool." [Online]. Available: http://www.netperf.org/