Stateful firewalling for wireless mesh networks

P. Neira, R.M. Gasca
Department of Languages and Systems
Quivir Research Group
ETS Ingeniería Informatica
University of Sevilla, Spain
{pneira|gasca}@lsi.us.es

Leonardo Maccari
Department of Electronics
and Telecommunications
Telecommunication Networks Lab (LaRT)
University of Florence, Italy
maccari@lart.det.unifi.it

L. Lefèvre
INRIA RESO - Universite of Lyon
LIP Laboratory
(UMR CNRS, INRIA, ENS, UCB)
ENS de Lyon, France
laurent.lefevre@inria.fr

***Abstract*—Firewalls have been traditionally used to apply filtering policies in wired networks, to divide zones with different level of trust. In wireless distributed networks, such as mesh networks for service delivery, firewalling is a valuable instrument to control the behavior of the clients and avoid certain attacks, such as DoS attacks coming from the inside of the network. In previous works we have outlined the possibility of applying stateless firewalling to distributed mesh networks using Bloom filters. In this paper we will expand this model to perform stateful firewalling in mesh networks, that will allow a more fine-grained control over the traffic passing over the network. Preliminary experimental results are also provided.**

I. Introduction

Wireless mesh networks (WMN) are mobile distributed networks composed of terminals connected using wireless links. In these networks, a set of access points (AP) forms a backbone that provides network access to clients. Standards like IEEE 802.11 or 802.16 can be used to produce WMN and should guarantee that network entrance is limited to authorized terminals by means of layer II access control techniques. However, still many kind of attacks can be performed at higher layers to disrupt services. A firewall is an instrument that can be used to limit the impact of such attacks.

In wired infrastructured networks, firewalls separate network segments and enforce filtering policies that determine what traffic is allowed to enter and leave the network, filtering policies are expressed as access control list (ACL). The ACL is composed of a set of rules which use selectors that match several packet fields, e.g. source and destination address, ports, etc. and an action to be issued, usually accept or deny.

Similarly, firewalls can be used to improve network security in WMN but since there is no well defined concept of perimeter, we would have to enforce the filtering policy in the whole AP backbone in order to deploy an effective firewalling. In our previous works [1] [2], we have proposed a distributed firewalling solution for WMN based on bloom filters [3] whose main concerns are:

1) *Low computational complexity:* needed because the APs are usually embedded devices with limited resources.
2) *Efficient ACL distribution:* since the bandwidth resource in WMN is scarce, ACL updates must require low bandwidth.

In this work, we extend our solution to enhance *stateful firewalling*. Stateful firewalls perform correctness checks upon communication protocols. These firewalls implement a state automaton for every supported protocol to ensure that communication between two peers evolves in a standard complaint manner. This evolution is stored a set of variables $V = \{v_1, v_2, ..., v_n\}$ that represent the current state S_k of the flow F_j. Thus, the security architect can use the stateful capabilities in their ACL to deny the packets that trigger invalid state transitions. The design of a stateful firewalling solution for WMN has to fulfill two requirements:

1) *Low computational complexity:* as in our previous works, the solution must be suitable for devices with limited resources.
2) *Handover support:* since one client C_A can roam from the access point AP_x to AP_y at any moment, the state information of the traffic of C_A should be synchronized to AP_y.

The paper is organized as follows: In Sect. II, we briefly detail our previous works. We detail the implementation of stateful firewalling with bloom filters in Section III. The handover support is described in Section IV. Then, we evaluate our solution proposed in Section V. We conclude with the conclusions and future works in Section VI.

II. Bloom-based stateless firewalling for WMN

A Bloom filter (BF) is a space-efficient structure for an inexact representation of a set that allows false positives but not false negatives. Thus, 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 $a \notin B$ (See [4] for more details on Bloom filters).

In our previous work, we have proposed a bloom-based solution to represent the firewall ACL. The representation is used to classify and filter traffic. We divided the ACL into subsets that contain the rule-set of each client node that can be attached to the backbone. Whenever a client node joins the backbone, the subset of firewall rules that represents its filtering policy is distributed by the authentication authority along the backbone in the form of a bloom filter.

We consider that our approach is expressive enough to profile most of typical traffic streams of common networks. The results show negligible impact in performance terms, comparable to no firewalling at all, while traditional firewalling with standard list-based ACL greatly lowers the throughput

and round trip delay. We have also evaluated a RADIUS based solution to deliver to the AP of the mesh the rule-sets. For details on the stateless firewalling approach, see [1] and [2].

III. Bloom-based stateful firewalling for WMN

WMN are distributed networks composed of terminals connected using wireless radio links. These networks consist of a set of access points (AP) that forms a backbone. The backbone provides network access to static and mobile clients. In certain environments, the APs can also be mobile but, in general, the backbone topology is less susceptible to changes. Anyhow, the backbone must be able to dynamically reconfigure its routes as new links between APs may appear and others may fade away. Also, the clients may change their point of attachment from one AP to hand over to another. Mobility is the main cause of changes in the network topology, however, a failure in one of the APs may also trigger network reconfigurations.

Our solution allows a low rate of false positives but not false negatives. Thus, a low rate of packets may go through the AP while they should not. This might seem problematic in terms of security, however, our solution is better than having no firewalling at all. On the other hand, no packets that belong to a flow that evolves appropriately are ever denied. This is the main difference with regards to [5] as we do not risk to reject packets that must go forward. As said, we consider that our stateful firewalling approach is significant to provide higher level of security for WMN.

A. d-left counting bloom filters

For the purposes of this work, we use an improved construction of Counting Bloom filters [6] based on d-left hashing that gives near-perfect hashing, the so-called *d-left counting bloom filters* (dlCBF). This construction saves memory consumption by a factor of two or more. The idea consists of a table split into d subtables, each subtable has k buckets, and each bucket can keep c cells of size w. The insertion is composed of two steps:

1) The fingerprint of the element x is calculated with a hash function $f(x)$ that generates a fingerprint f_x of size w.
2) We use d pseudo-random permutations (one per subtable) $P_1(f_x), P_2(f_x), ...P_d(f_x)$, the first n bits of f_x are used to determine the bucket where the fingerprint is inserted, and among all buckets the less loaded, i.e. the bucket that keep less fingerprints, is selected. The last $w-n$ bits, so-called reminder, are stored in the selected bucket. If the reminder is already stored, the counter is incremented.

Since fingerprint collisions are rare, we assume that counters can be smaller than the one used in CBF (2 bits vs. 4 bits). Assuming a dlCBF that stores n elements where each subtable has a size of $b = 2^z$ buckets with c cells and fingerprints of size w, the rate of false positives is $n2^{-(z+w)}$. Moreover, to avoid bucket overflows, we calculate the estimated bucket usage as $u = n/dbc$. The authors of dlCBFs affirm that by using $n/12$ buckets the estimated usage is $u = 2/3$ which provides a very rare probability of a bucket overflow.

Thus, a dlCBF composed of 3 subtables, 341 buckets, 6 cells per bucket, fingerprints of 10 bits, and counters of 2 bits, has an approximately rate of false positives of 0.015, and it gives an overall of $3 * 341 * 6 * (18 + 2)/4096 = 19$ bits per element. Therefore, this dlCBF consumes approximately 9 KB versus standard CBF of 21 KB (approximately 2 times less than a CBF with similar rate of false positives).

B. Design

Basically, the proposed model consists of inserting a set of tuples in the dlCBF that represent the set of possible next valid states, the so-called *state expectations* (SE). Thus, every packet p that arrives to the AP has to verify that it triggers a new state S_k that belongs to the set of SE. If p does not fulfill an expectation, it is denied as it does not belong to a flow that follows a sane evolution.

We represent one state expectation of a flow F_j through a tuple, the so-called *state expectationion tuple*, that is composed of five flow selectors and the flow state: $T_i = \{Addr_{src}, Addr_{dst}, Port_{src}, Port_{dst}, Pnum, State\}$. Thus, the set of SE of F_j, $\pi = \{T_0, ..., T_h\}$, represents the finite set of possible expected states $\theta = \{S_\alpha, ..., S_\omega\}$ that the packet p that belongs to F_j must fulfill.

When a state expectation is fulfilled, we may delete it from the dlCBF. However, since the lookup operation has a low rate of false positives, the deletion of a wrong fulfilled state expectation introduces errors that lead to false negatives. For that reason, we use timing-based deletions. Timing-based BFs use memory allocated for the counter bits to store a timer instead [5]. This variation uses the counters as set of *recently used* bits $U_m = \{u_0, ..., u_r\}$ for every bit b_m in the BF. Basically, u_0 is set when b_m is set, and periodically the bitset U_m is shifted. When $U_m = \{0, ..., 0\}$, the bit b_m is unset.

The timing-based deletion provides a way to limit the lifetime of one state expectation, and it also naturally provides a mechanism to ensure that the flows do not evolve abnormally. Thus, we assume that every fingerprint f_p of the dlCBF has m timing bits to delete the state expectations that have not been confirmed after r phases. The length of the phase δ and the duration in phases of SE are tunable parameters that depends on the acceptable state maximum lifetime (in terms of the protocol specification). Thus, every fingerprint f_p is removed from the dlCBF after $r * \delta$ time units.

However, instead of setting the timing bit u_0 of $U_m = \{u_0, ..., u_r\}$ when the fingerprint f_p is inserted in the dlCBF, we store the number of phases r in the set of timing bits after which the state expectation expires. Thus, we can define different maximum lifetimes for each state expectation S_k. This is interesting since we usually have some SE that live longer than others. Moreover, if we try to insert an already existing fingerprint f_p in the dlCBF, we reset the corresponding timing bits to $r * \delta$.

Every phase δ we decrease the timing bits in one, if they reach zero, we remove the corresponding fingerprint from the BF. Thus, we do not delete confirmed SE, instead we let them expire. As said, this timing-based deletion approach ensures

that no false negatives are introduced. The values of r and δ depends on:

1) The level of strictness of the conformance check: low r and δ means that state expectations have a very short lifetime.
2) The maximum state lifetime: these information can be extracted from the protocol specification.
3) The round-trip time (RTT) of the network: if the delay is high, state expectations may expire before the packets fulfill them. Thus, leading to unexpected flow hangs.

C. Operation

When the client C_A sends a packet p, the mesh AP receives it, and it invokes its stateful firewall routine (SFR). The SFR decides if p continues the traversal based on the ACL and the set of existing SE.

```
1  match ← check_ACL_match(packet);
2  if not match then
3  |  deny(packet);
4  end
5  tuples ← infer_tupleset(packet);
6  for each tuple in tuples do
7  |  expected ← state_lookup(tuple);
8  |  if expected then
9  |  |  tuples← infer_nxt_tupleset(packet);
10 |  |  for each tuple in tuples do
11 |  |  |  state_insert(tuple);
12 |  |  end
13 |  |  accept(packet);
14 |  end
15 |  valid ← conformance_check(packet);
16 |  if valid then
17 |  |  tuples← infer_nxt_tupleset(packet);
18 |  |  for each tuple in tuples do
19 |  |  |  state_insert(tuple);
20 |  |  end
21 |  |  accept(packet);
22 |  end
23 |  deny(packet);
24 end
```

Algorithm 1: Stateful firewall routine (SFR)

Specifically, the SFR checks if p matches the ACL, if it does not, then the packet p is denied. On the other hand, if p matches the ACL, the SFR infers from p which are the set of possible SE and checks if there is a matching tuple in the dlCBF, ie. the packet p is valid if it fulfills one of the SE, $\theta = \{S_\alpha, ..., S_\omega\}$. If so, then we assume that p belongs to an existing flow F_j that is evolving according to the standard. Thus, the SFR infers the next set of possible SE tuples and let the packet go through.

The SFR may find a matching for p in the ACL but not in the set of SE. This means that p may be the first packet of a flow. Thus, the SFR performs a conformance check on p to ensure that the packet is well-formed according to the communication standard. If the packet passes the conformance check successfully, then the SFR infers the set of next possible SE tuples and insert them into the dlCBF. This conformance check validates that the packet p follows an acceptable configuration as first packet of a flow according to the specification. We have formalized the SFR in *Algorithm. 1*.

D. State inference

The inference procedure is extracted from the state automaton. We assume that the SFR has a state automaton for every supported protocol so that the set of possible states is finite and deterministic $S = \{S_0, ..., S_n\}$. The state automaton can be represented as a set of state-nodes connected with edges where every state-node has a set of input edges $E_i = \{e_{i_0}, ..., e_{i_h}\}$ and output edges $E_o = \{e_{o_0}, ..., e_{o_g}\}$. Thus, given two different states S_k and $S_k + 1$, the edge e_r can be an output edge of S_k and an input edge of S_{k+1}. Also, every edge e_r is marked with the packet type p_t that triggers the transition to a certain state-node.

The current-state inference consists of obtaining the subset of possible current states that p_t can be in, ie. every state that can be reached from an input edges marked with the packet type p_t is a current state candidate. On the other hand, the next-expected-state inference consists of adding the set of states that can be reached following the output edges that leave the set of possible current states. We provide an example automaton of TCP in Sect. 1.

E. False positives

The rate of false positives depend on three cases that result from the combination of the probability of having false positives in the ACL matching and the state expectation lookup, they are:

- A false positive in the ACL: Then, the false positive rate of the SFR ,$f(SFR)$, is equal to the false positive rate of the ACL matching, $f(ACL)$.
- A malformed packet p_m matches the ACL falsely matches a state expectation: $f(SFR)$ is the rate of false positive of the state expectation matching, $f(SEM)$.
- A malformed packet p_m falsely matches the ACL and one state expectation: this is the worst case, however, the rate of such false positive is low as it is $f(ACL) * f(SEM)$.

The rate of false positives $f(SEM)$ depends on the number of lookups performed in the dlCBF which is the number of current possible states that has been inferred from the packet. Thus, the probability of a false positive depends on how many possible current states can be inferred from a packet p_t. Consequently, $f(SEM)$ depends on the packet type.

F. An example: simple stateful firewalling for TCP flows

For instance, we assume a TCP flow F_{tcp} between two peers C_A and C_B that is filtered by one mesh AP whose behaviour must validate a simple TCP protocol automaton (Fig. 1) with support for connection reopening and keepalive (as described in RFC1122). The communication flow F_{tcp} is composed of a

set of packets $P = \{p_0, p_1, ..., p_j\}$ that are exchanged between the peers C_A and C_B using the mesh AP as gateway. For simplicity, we assume that every state S_k is composed of only one variable which stores the current TCP protocol state.

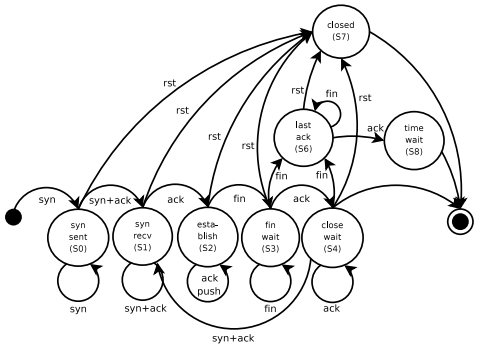

Fig. 1. Simple TCP state automaton with RFC1122 support

When one of the mesh APs SFR receives the first packet p_0 of F_{tcp} (a packet with the *SYN* flag set from C_A to C_B) that matches the ACL, the SFR infers the current tupleset which is, in this particular case, composed of only one tuple: $T_0 = \{C_A, C_B, port(C_A), port(C_B), S_0\}$.

Thus, since the state-tuple T_0 is not in the set of SE stored in the dlCBF, as p_0 is the first packet of F_{tcp}, the SFR performs the conformance check. This check is successfully passed since p_0 has the *SYN* flag set which is a valid combination to initiate a TCP flow. Then, the SFR infers three SE tuples: one for the next expected state *SYN+ACK* in the reply direction $T_1 = \{C_B, C_A, port(C_B), port(C_A), S_1\}$; one for possible *SYN* retransmissions $T_2 = \{C_A, C_B, port(C_A), port(C_B), S_0\}$; and one for flow closure via *RST* $T_3 = \{C_B, C_A, port(C_B), port(C_A), S_5\}$, and it insert them in the dlCBF.

If the next packet p_1 has the *SYN+ACK* flags set, the SFR infers that the current state of the flow F_{tcp} must be S_1. The packet p_1 indeed fulfills the state expecation tuple T_1 so that the SFR infers again the next set of SE: one for the next expected state *ACK* $T_3 = \{C_A, C_B, port(C_A), port(C_B), S_2\}$; one for possible *SYN+ACK* retransmission $T_4 = \{C_B, C_A, port(C_B), port(C_A), S_1\}$; and one for flow closure via *RST* $T_5 = \{C_A, C_B, port(C_A), port(C_B), S_5\}$. The operation with the following packets is similar.

As said, there are three different false positive cases. For the worst case, ie. a malformed packet which does not match the ACL is accepted, the rate of a false positive depends on the type of the packet received. For instance, if an *ACK* packet is received, the SFR infers three possible current states (S_2, S_5 and S_8). This means that we need three lookups to check for SE. Thus, the rate of false positives for *ACK* packets is $3 * f(SEM)$. Assuming $f(ACL) = f(SEM) = 0.01$, the rate of false positives is $f(ACL) * 3 * f(SEM) = 0.0003$ for the worst case.

G. Limitations

Our proposed solution inherits the same limitations of existing stateful firewalling solutions since it is still possible to deploy DoS attacks such as TCP SYN-flood and RST-flood. Nevertheless, these packets are valid combinations of the TCP protocol specification. Therefore, TCP stateful firewalls cannot reject these attacks by means of stateful inspection solely.

Still, our solution predicts the next set of state expectations, it would be possible for an attacker to spoof a packet that can match one state expectation. This is part of the nature of our approximate stateful firewalling solution. Nevertheless, this does not affect the deployment of the stateful firewalling of a certain flow as matching state expectation does not delete other state expectations. This keeps the deployment of DoS attacks against our stateful firewalling approach harder.

IV. Handover support

The handover is a common operation in mobile networks that occurs when one client C_A leaves the AP_x to connect AP_y. Several reasons can trigger an handover, from physical restrictions, e.g. building layouts, signal quality, ... to failures, e.g. the AP_x stops working.

The deployment of stateful firewalling in WMN can lead to flow disruptions during the handover. Let's assume the following scenario: the mobile client C_A with n open flows $F = \{f_1, ...f_n\}$ is connected to the AP_x. Thus, AP_x stores the set of states $S = \{state(f_1), ..., state(f_n)\}$ of C_A's open flows and it deploys the filtering according to the stateful firewall routine exposed in the previous section. Now, the client C_A roams from AP_x to AP_y. However, the current set of states S is not known by AP_y. Thus, according to *Algorithm 1*, the packets that belong to existing flows will be denied.

In order to solve the handover problem, we have to define a state replication solution so that neighbour mesh APs can know which is the current set of states of C_A.

The handover support is based on FT-FW [7] which is one of our previous works. FT-FW is cluster-based reactive fault-tolerant software solution at application level for stateful firewalls in infrastructured networks. Although such work is focused on fault tolerance, we consider that its contribution is significant to solve the handover problem. Basically, FT-FW guarantees that the flow states are known by all the stateful firewall replicas that compose the cluster. Thus, one of the stateful firewall replicas can recover the filtering if a failure arises. In infrastructured networks, the utility of the state replication is usually to enable fault tolerance. However, in the case of WMN, FT-FW guarantees that states are known by all stateful firewall replicas which is what we need to solve the handover problem. We have adapted our previous works to the WMN scenario.

A. FT-FW overview adapted to WMN

The FT-FW architecture and replication protocol keeps in mind simplicity, transparency and negligible delay in client responses that are desired properties for the WMN scenario.

We assume that every AP_x has a set of neighbour APs $\gamma = \{AP_1, ..., AP_m\}$ where $m \geq 1$. We consider that two whatever APs, AP_y and AP_z, are neighbours if they have a direct established wireless link. We assume that AP_x filters a set of flows $F = \{F_1, F_2, ...F_n\}$ and has set of clients $C = \{C_1, ..., C_j\}$ where $j = 0$ implies $n = 0$. Every flow F_i in F is in a state S_k. The flow states are a finite set of deterministic states $S = \{S_1, S_2, ..., S_n\}$.

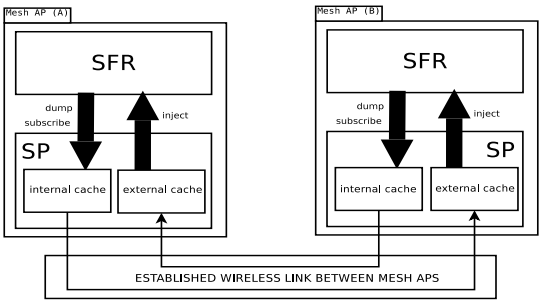

Fig. 2. FT-FW architecture for WMN handover support

The FT-FW architecture follows an event-driven model (EDM) whereby any new state expectation is propagated through an event. These events are produced by the stateful firewall and consumed by the state proxy (SP). The SP is an application that runs in the stateful mesh AP and propagates the SE to neighbour mesh AP nodes. The EDM suits well for distributed systems since share many of the same characteristics such as modularity and loose-coupling, and whose asynchronous nature suits well for the performance requirements of stateful firewalls. Also, as there is no concept of dedicated wired link in the WMN scenario, we assume that two neighbour APs, AP_x and AP_y, use the existing link between them to transfer the state-expectation changes. We have represented the FT-FW architecture adapted to WMN and the information replication flow in Fig. 2.

The SFR provides a framework to manipulate the dlCBF that stores the set of state expectations. Basically, the framework offers a method to subscribe to new state-expectation events; one to dump the full dlCBF that store the state expectations; and another to inject state expectations to the SFR to enable the handover. The SPs use this framework to interact with the SFR. We have modified the SFR routine to send state-expectation events to the SP whenever a non-existing state expectation is added to the dlCBF. Thus, we notify every new state expectation that is inserted in the dlCBF through an event. The event is composed of the state expectation tuple and the coordinates $(subtable, bucket, cell)$ plus the fingerprint inserted in the dlCBF. We assume that there are two kind of events: *new* that represents a new state expectation; and *destroy* that notifies that a fingerprint of the dlCBF has expired via timing-based deletions.

- *Internal cache*, that is a cache that stores the set of local SE, ie. those states that correspond to flows that are being filtering by this mesh AP. This can be a subset of the SE that are held in the dlCBF as we may not allow the handover of certain clients between two different subset of mesh APs.
- *External cache*, that are a set of caches that store the foreign states, ie. the state expectations of its mesh AP neighbours. We assume that AP_x has a number of external of caches equal to the sum of clients that each neighbour AP belonging γ have.

At startup, every SP dumps the existent SE and stores them in the internal cache; and it also subscribes to events of SE to keep the cache up to date. The SP also maintains another cache to store foreign SE that comes from other mesh APs. When an event of state expectation occurs, the SP updates its internal cache and it propagates the state expectation to other SPs that run in the neighbour mesh APs to update their external cache.

The state expectation table is compactly distributed in the form of a dlCBF and new state expectation are distributed as incremental differences. Assuming the example dlCBF of 9KB, we can represent the position of a fingerprint in the dlCBF as a coordinates of three parameters $(subtable, bucket, cell)$, requiring 1 bit to describe the kind of update (add, delete), $log_2(d)$ bits for the subtable axis, $log_2(b)$ bits for the bucket and $log_2(c)$ bits for the cell plus w bits of the fingerprint and the counter.

For instance, assuming the example dlCBF of 3 subtables, 341 buckets and 6 cells, every new state-expectation message consumes 1+2+9+3+10=25 bits per update (plus the packet header size). On the hand, the message to notify that one state expectation has been removed from the dlCBF (destroy messages) consists of three parameters $(subtable, bucket, cell)$ which is 1+9+3=13 bits plus the packet header (which is 20 bytes for multicast UDP). Therefore, the replication messages require very low network bandwidth. We assume that the SP implements a reliable multicast replication protocol that exploits the stateful firewall semantics to perform an efficient replication as described in [7].

When a client C_A roams from mesh AP_x to AP_y, the AP_y's SP that invokes the inject method to insert the state expectations stored by the external cache into the SFR. This process consists of merging two dlCBF: the one used by the SFR and the one stored in the external cache. We assume that during the handover the AP_y requests to the client C_A which was the former mesh AP that it was connected. Thus, the mesh AP_y can inject the appropriate external cache.

V. Evaluation

In order to evaluate our proposed solution, we have measured computational complexity by means of the CPU and memory consumption metrics. We also have measured the bandwidth and CPU consumption to enable the handover support via state replication. We have compare the results obtained of our stateful firewalling approach with a reduced version of Netfilter/iptables stateful firewall (μiptables) that provides similar features than our solution. The μiptables solution uses 24 bytes to stores per-flow states instead of the 168 bytes that it requires in a Linux kernel 2.6.25. Thus,

we can provide a fair comparison between our solution and an existing Open-Source stateful firewall implementation with similar features. Of course, the non-reduced version of Netfilter/iptables provides much more stateful firewalling features than our solution but it would not scale up for the WMN scenario as we justify in this section.

As we only want to evaluate the stateful part of the firewalling, not the packet classification, we assume that the μiptables default policy is accept and the ACL is: iptables -I FORWARD -m state –state invalid -j DROP.

Our testbed is composed of two mesh AP nodes and two clients running GNU/Linux which use the UU-AODV. We have implemented the dlCBF using Jenkins hash, which provides a good distribution without degrading performance. The SFR is a kernel module for GNU/Linux that handle packets in the *forward* hook of the Netfilter framework. We have also adapted our Open-Source implementation of the SP [8] to replicate state expectations of the SFR. The mesh nodes are two laptops PIII 800 MHz with 128 Mbytes of memory and wireless links are standard 11 Mbps 802.11b wireless links

A. Computational complexity

The CPU consumption our the stateful firewalling solution is low, reaching up to 16% of CPU with 30 connections per second (cps). Our solution slightly outperforms μiptables (Fig. 3). The memory requirements of our solution also outperforms μiptables that consumes 96KB to store 4096 flow-states versus 8KB of our dlCBF-based solution with 3 subtables, 6 cells, 341 buckets (Fig. 4). We observed similar RTT for both solutions. Still, μiptables with the simplistic rule-set is suitable for the WMN scenario since 48KB is a easy-to-fulfill memory requirement.

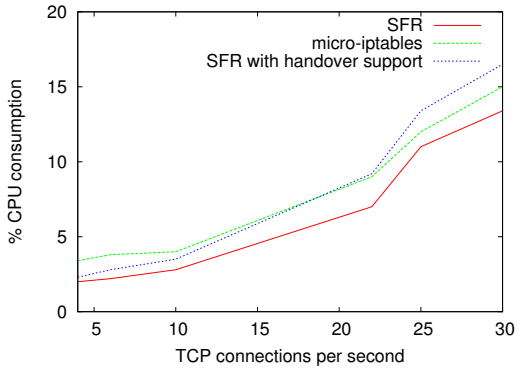

Fig. 3. CPU consumption

B. Handover

The amount of memory that the stateful firewalling requires to store the flow-states is an important metric to evaluate the scalability of the solution in terms of the handover. In the case of μiptables, assuming that AP_x filters 2000 flows and MTU is 1500, the initial synchronization between two mesh nodes, ie. when mesh AP_x node initially establishes a link with AP_y, requires 33 packets. However, our approach only requires the

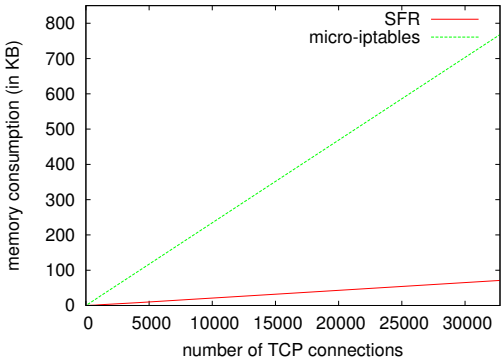

Fig. 4. Memory consumption

transfer of 3 packets to fully resynchronize the mesh node (See Fig. 4, divide the size of the dlCBF of the SFR by 1420 which is the size of the packet payload without the link layer headers). Also, every state-change message of μiptables requires 33 bytes: 12 bytes to identify the flow by means of the source and destination, and 1 byte to store the current TCP protocol state, plus the 20 bytes of an IP header. Our solution only requires 24 bytes, 1 byte to encode the change of a cell through the tuple $[subtable, bucket, cell, fingerprint]$.

We have also evaluated the CPU consumption of the state replication in order to evaluate the feasibility of the handover support. The results show that the state replication requires approximately 5% extra CPU (Fig. 3).

VI. Conclusions and Future Work

In this work, we have proposed an approximate stateful firewalling solution adapted to wireless mesh networks which fulfills the computational complexity limitations and it solves the handover problem by means of replication techniques. As future work, we plan stateful UDP-based protocols used in real-time applications such as VoIP. Also, we expect to further validate the proposed solution in a more realistic testbed.

References

[1] L. Maccari, R. Fantacci, P. Neira, and R. M. Gasca, "Mesh network firewalling with bloom filters," in *Communications, 2007 IEEE International Conference*, 2007, pp. 1546–1551.

[2] L. Maccari, P. Neira, R. Fantacci, and R. Gasca, "Efficient packet filtering in wireless ad-hoc networks," *IEEE Communications Magazine*, feb 2008.

[3] 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

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

[5] F. Bonomi, M. Mitzenmacher, R. Panigrah, S. Singh, and G. Varghese, "Beyond bloom filters: from approximate membership checks to approximate state machines," *SIGCOMM Comput. Commun. Rev.*, vol. 36, no. 4, 2006.

[6] F. Bonomi, M. Mitzenmacher, R. Panigrahy, S. Singh, and G. Varghese, "An improved construction for counting bloom filters," *ESA 2006*, 2006.

[7] P.Neira, R.M.Gasca, and L.Lefevre, "Efficient failover for cluster-based stateful firewalls," in *16th Euromicro International Conference on Parallel, Distributed and network-based Processing*, Toulouse, France, Feb 2008.

[8] P. Neira, "conntrack-tools: The netfilter's connection tracking userspace tools," http://people.netfilter.org/pablo/.