Modeling and Analysis of Firewalls by (Tissue-like) P Systems

Alberto Leporati, Claudio Ferretti

Dipartimento di Informatica, Sistemistica e Comunicazione
Università degli Studi di Milano – Bicocca
Viale Sarca 336/14, 20126 Milano, Italy
{leporati,ferretti}@disco.unimib.it

Summary. We propose to use tissue-like P systems as a tool to model and analyse the security properties of firewall systems. The idea comes from a clear analogy between firewall rules and P systems rules: they both modify and or move objects (data packets, or symbols of an alphabet) among the regions of the system. The use of P systems for modeling packet filters, routers and firewalls gives the possibility to check — and possibly mathematically prove — some security properties.

1 Introduction

Firewalls are devices that allow to control network traffic. Depending on the protocol layer they operate at, firewalls can be classified into packet filters, circuit proxies, and application level proxies. Since they allow to enforce security policies, firewalls are essential for organizations that are connected to the Internet, and/or whose networks are divided in a number of segments. In fact, they operate like filters that selectively choose what data packets are allowed to cross the boundaries between network segments, and thus ensure that information flow between those segments only in the intended ways.

When deploying firewalls in an organization, it is essential to verify that they are configured properly. Unfortunately, firewall configurations are often written in a low-level language which is hard to understand. Thus, it is often quite difficult to find out which connections and services are actually allowed by the configuration. Indeed, it is well recognized that writing a correct set of rules is a challenging task. Hence, network administrators would benefit greatly using a tool that helps them to analyze the behavior of firewall rules.

Membrane systems (also known as *P systems*) are a distributed, parallel and synchronous model of computation inspired by the functioning of living cells [15]. The basic model consists of a hierarchical structure composed by several membranes, embedded into a main membrane called the *skin*. Membranes divide the Euclidean space into *regions*, that contain some *objects* (represented by symbols of an alphabet) and *evolution rules*. Using these rules, the objects may evolve and/or move from a region to a neighboring one. At least two ways to apply the rules are considered in the literature: the *maximally parallel* and the *sequential* way. When two or more (sets of) rules can be applied in a given computation step, a nondeterministic choice is performed. A *computation* starts from an initial configuration of the system and terminates when no evolution rule can be executed. *Tissue* P systems [10, 11] can be viewed as an evolution of P systems, corresponding to a shift from cell-like to tissue-like architectures, based on intercellular communication and cooperation between cells. In this model cells are usually composed of a single membrane, and the interconnection structure forms an arbitrary graph. The cells are the nodes of the graph, and objects may either evolve by means of *evolution rules*, or move between cells (alongside the edges of the graph) as a result of the application of *symport/antiport* or *uniport* rules. In what follows we assume the reader is familiar with the basic notions and the terminology underlying P systems. For details, and a systematic introduction on the subject, we refer the reader to [16, 17]. The latest information about P systems can be found in [14].

By looking at firewall rules as filters that selectively choose what data packets are allowed to cross the boundary between two regions of the network, it is apparent that a firewall operates like a semi-permeable membrane that separates the two regions. Hence, membrane systems are easily seen as a natural tool that allow to model and analyze firewall systems.

In this paper we apply the membrane computing paradigm to the problem of properly configuring a collection of firewall systems. The scenario we imagine is the following: a network administrator has to devise the rules that allow to control traffic in a large network, composed of several segments. Each segment delimits an area, or zone, that contains hardware equipments such as PCs, servers, printers, etc. Each pair of adjacent areas is separated by a firewall, that for each direction selectively filters what data packets are allowed to cross the boundary in that direction. We assume that firewalls operate as packet filters, which are allowed to examine and possibly modify the fields of IP packets. This means, in particular, that they are also able to operate as *routers*: for example, they can modify the destination address of all packets having a specified source address and destination port. The aim is to help network administrators in testing and analyzing firewall rules before implementing them; in fact, usually the implementation must be performed quickly, since as soon as all network equipments are mounted the organization wants to start using the network. Moreover, it is always desirable to test a configuration change before putting it at work in a real network.

Another goal of our work — not addressed in this paper — is to give the possibility to *mathematically prove* security properties (such as, for example, the impossibility for a certain kind of TCP packets to reach a given region of the network). This will be obtained by considering *reachability* problems on the P systems that simulate the functioning of the firewalls. So doing, we will be able to find answers to questions like:

- What is the action for a specified IP packet?

- What packets are permitted by this list of rules?
- From which sources are packets to this destination permitted?
- Which services are accessible on a given host?
- Is a given host/network accessible from another given host/network?
- What kind of traffic is allowed between two networks?
- From which networks is a given host accessible?

Of course, some attempts have already been made in the literature to provide tools that help network administrators to test and analyze firewalls and packet filters before implementing them. To the best of our knowledge, no one of these attempts uses membrane systems. Moreover, much of the work currently present in the literature focuses on the configuration of a single firewall, and proposes algorithms that detect common configuration mistakes, such as rule shadowing, correlation, generalization and redundancy. The work whose spirit is most similar to ours has been done by Eronen and Zitting [1]; in their paper they describe an expert system whose knowledge base contains a representation of a firewall configuration. The tool allows to model a single firewall, but it has the advantage that it can also compare the firewall configuration against a list of known vulnerabilities and attacks, and warn the user whenever a match is found. Mayer, Wool and Ziskind [12] propose a firewall analysis engine based on graph algorithms. Similar work based on a logic background has been done by Hazelhurst et al. [4, 5, 6], where ordered binary decision diagrams have been used to analyze routers' access control lists. This representation allows for efficient handling of the lists: for instance, finding redundant (shadowed) rules is easy. Several researches have also implemented tools for describing and generating the contents of an access list. For example, Guttman [2] describes an approach for generating filters' rules starting from a desired security policy, and verifying that a packet filter correctly implements a given security policy.

The paper is organized as follows. In section 2 we briefly describe the features of packet filters we want to model, and consequently we derive some properties and constraints of the P systems we will use. In sections 3 and 4 we present the model of tissue-like P systems that results from this analysis, and we show how this model can be simulated by a single-membrane P system. Section 5 contains the conclusions, and gives some directions for future research.

2 Preliminary Analysis

We focus our attention to *stateless* firewalls with packet inspection, also known as *packet filters*. We assume the reader has some background in IP-based networking [19].

As told in the Introduction, firewalls function as routers which connect different network segments together. Based on their configuration, they may restrict the traffic flowing between the different segments. Despite being so common, firewalls, routers, and many other simple packet filters usually lack good user interfaces

```
1 permit udp any host 192.168.1.1 eq 53
2 deny udp any host 192.168.1.2
3 permit udp any 192.168.1.0 0.0.0.255 eq 123
4 permit udp any host 192.168.1.2 eq 177
5 deny ip any any
```

Fig. 1. An example of a Cisco router access list. Note that the fourth rule is never matched because of the second rule

for specifying the desired security policy. Hence it is very easy to make mistakes when writing the lists of filtering rules, especially when these lists are long (several hundreds rules is not uncommon). In particular, it is possible to make four kinds of errors when implementing a security policy as a set of rules: *shadowing*, *correlation*, *generalization*, and *redundancy*. Briefly, a rule r_1 *shadows* a rule r_2 when r_1 is always executed before r_2, and r_2 operates on a subset of the IP packets which are processed by r_1. In this situation r_2 can never be executed and hence is useless. A pair of rules r_1 and r_2 are *correlated* when the sets of packets processed by them are not one the subset of the other but have a nonempty intersection; the problem arises in particular when the actions (either `accept` or `reject`) specified in the two rules are different. A rule r_2 is a *generalization* of a rule r_1 if r_2 follows r_1 in the order, and r_2 matches a superset of the packets matched by r_1, and the actions of r_2 and r_1 are different. Generalization may not be a configuration error, since it is generally accepted that more specific rules are followed by more general rules; however, since the actions performed are different it is a good practice to warn the administrator, so that he is aware of the situation. A *redundant* rule performs the same action on the same packets as another rule such that if the redundant rule is removed, the security policy will not be affected. Rule r_2 is redundant to rule r_1 if r_1 precedes r_2 in the order, and r_2 matches a subset (or the same set) of packets matched by r_1, and the actions of r_1 and r_2 are the same. If r_1 precedes r_2 and r_1 matches a subset of the packets matched by r_2, and the actions of r_1 and r_2 are the same, then rule r_1 is redundant to rule r_2 provided that r_1 is not involved in any generalization or correlation anomalies with other rules preceding r_2.

Since the administrator may write several rules that may affect the same set of IP packets, when one of these packets arrives the firewall must choose what rule to apply. Since a nondeterministic choice is not desirable, the firewall disambiguates by considering the rules in the same order as they have been specified, and applies the first rule whose parameters match the packet. For instance, Figure 1 shows an example of a Cisco router access list. When a packet is received, the list is scanned from the start to the end, and the action (either `permit` or `deny`) associated with the first match is taken. If a packet does not match any of the rules, the default action is `deny`. Often a "deny all" rule is included at the end of the list to make it easier to verify that a list has not been truncated. Separate lists can be specified for each network interface. As told above, it is very easy to make mistakes when

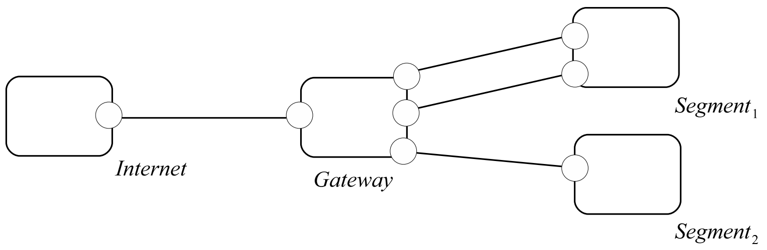

Fig. 2. The structure of a firewall P system. Each node of the graph represents a membrane in our tissue-like P system, that on its own represents a segment or zone of the network. The edges show the connections between zones. The small circles located on the membranes represent the lists of rules that filter the packets incoming to the membranes through the corresponding edges

writing access lists. For instance, the fourth rule in Figure 1 is never matched because it is shadowed by the second rule.

A crucial observation is that IP packets are processed one at the time, as soon as they arrive. Stated otherwise, network traffic is seen as a *stream* of packets. No cooperation among packets and no form of parallelism exist in current firewalls. If we sum all these observations, we can see that our model of P systems should apply *non-cooperative* rules in the *sequential* way; moreover, we should either impose that they are deterministic, or otherwise we should associate a linear (i.e., total) ordering $<$ to rules, so that if two rules $r_1 < r_2$ can be applied to a given object then r_2 (the rule which has the hightest priority) will be applied. However the assumption that our P systems are deterministic is not realistic, since this would mean that the above mentioned problems in writing firewall rules (shadowing, correlation, generalization, redundancy) have been solved *before* modeling the firewall by using the P system. Since we would rather like to use our P systems to solve these problems (as well as to answer questions concerning other security properties) we will assume that each rule has an associated priority.

Our rules will be an abstraction of the `accept` (also `permit`) and `reject` (also `deny`) rules which can be found in many packet filters such as, for example, Cisco routers [7], IPCHAINS and IPTABLES [13]. The rules will use the following fields from the IP protocol header: next level protocol (e.g., TCP, UDP or ICMP), source and destination IP addresses. In addition, some fields for upper level protocols, such as TCP and UDP port numbers, can be used. It will also be possible to specify entire subnets in place of single IP addresses, and to use wildcards when specifying the protocol or port numbers.

Since a firewall checks packets that try to pass the barrier in both directions, we should provide a separate list for the packets that try to enter and those trying to leave each of the regions separated by the firewall.

Another aspect that we have to take into account is that the segments of a network may be interconnected in an arbitrary way. Further, two segments may possibly be connected by two (or even more) network interfaces, that is, two segments may have multiple connections. This means that if we want to associate a membrane to each segment of the network we have to use a model similar to tissue P systems. In what follows we refer to this model — that will be formally defined in the next section — as *firewall P systems.* Figure 2 represents a sketch of the structure of a firewall P system. Nodes are labelled with the name of the segment or zone of the network, and edges show the existing connections between zones. The small circles located on the borders of the membranes represent the lists of rules that filter the packets incoming to the membranes through the corresponding edges. So, for example, $Segment_1$ is a zone connected to *Gateway* by means of two network interfaces. The two small circles located on the membrane of $Segment_1$ represent the two lists of rules that control the *incoming* traffic in the zone named $Segment_1$. The corresponding small circles located on the membrane named *Gateway* represent the lists of rules that filter the packets that come from $Segment_1$ and try to enter the zone *Gateway.* The precise form of the rules that compose these lists, as well as of the objects upon which these rules operate, is the subject of the next section.

3 The P Systems Model

In this section we give a precise definition of all the ingredients of our model of firewall P systems.

The *objects* represent IP packets. For the purposes of this paper, IP packets are represented as six-tuples

(*protocol, src_IP, dst_IP, src_port, dst_port, gateway*), where:

- *protocol* ∈ {TCP, UDP, ICMP};
- *src_IP* and *dst_IP* represent the source and destination address in the usual form (a quartet of integer numbers, each in the range 0..255);
- *src_port* and *dst_port* are integer numbers in the range 0..65535 that represent the source and the destination port associated with the source and destination address, respectively. These fields are meaningful only when *protocol* ∈ {TCP, UDP};
- *gateway* is an identifier of the edge to be followed at the next hop. It is used for routing purposes.

Note that in real IP packets there are other fields (such as *flags*) that we do not consider in this paper.

A *rule* is a quadruple of the form

(*priority, action, in_fields, out_fields*), where:

- *priority* is a non-negative integer that specifies the priority of the rule;
- *action* ∈ {`accept, reject, drop`} specifies how the packet filter should treat packets matched by the rule;

- *in_fields* is a six-tuple of the form

  (*protocol, src_IP/subnet, dst_IP/subnet, src_port, dst_port, gateway*),

  where:
  - *protocol* $\in$ {TCP, UDP, ICMP, `any`}. The value `any` is treated as a wildcard;
  - *src_IP* and *dst_IP* represent the source and destination address in the usual form (a quartet of integer numbers, each in the range 0..255). The subfield *subnet* is an integer in the range 0..32, and allows to specify subnets (that is, sets of IP addresses) in the customary way: it indicates that the specified number of bits of the IP address — starting from the most significant bit — are fixed (as specified in the address), whereas the others may assume any value. So, for example, the subnet 192.168.1.0/30 is composed by the IP addresses 192.168.1.x, where $x \in \{0, 1, 2, 3\}$.
    Alternatively, *src_IP* and/or *dst_IP* may assume the special value (wildcard) `any`. In such a case, the subfield *subnet* is not specified;
  - *src_port* and *dst_port* are integer numbers in the range 0..65535 that represent the source and the destination port (if *protocol* $\in$ {TCP, UDP}) associated with the source and destination address/subnet, respectively. Also in this case, *src_port* and/or *dst_port* may assume as a value the wildcard `any`;
  - *gateway* is an identifier (that is, a label) of the edge to which the rule is associated;
- *out_fields* is a six-tuple of the form

  (*protocol, src_IP, dst_IP, src_port, dst_port, gateway*),

  whose fields and values are defined exactly as those appearing in the objects of the system. Additionally, each field may assume the special value `same`, which indicates that the rule does not change the value of the field (that is, the value already present in the analyzed object is kept).

A rule r *matches* an object o if the fields specified in *in_fields* match. The fields *protocol*, *src_port* and *dst_port* match if they are equal (in the object and in the rule), or if the field in the rule contains the wildcard `any`. The field *src_IP* of o matches the field *src_IP/subnet* of r if the former IP address is contained in the subnet (set of IP addresses) of the latter. This definition includes the case in which the rule specifies a single IP address. The same criteria are applied when matching the field *dst_IP* of o with the field *src_IP/subnet* of r. The fields *gateway* match if and only if they are equal (in the object and in the rule).

The lists of rules are associated to the membranes, rather than to the regions enclosed by them, as is customary in membrane computing. Each list of rules checks the packets coming from a specific region, seeking to enter into the region enclosed by the membrane that contains the rules. As stated above, each list of rules is processed in the order given by priorities, until a match is found. The first matching rule specifies the action taken by the filter on the given object. In case two or more rules having the same priority match (a situation that should not occur, since it denotes a misconfiguration), a nondeterministic choice is made between such rules.

The application of an `accept` rule to an object o has the effect of letting the object enter the zone delimited by the membrane that contains the rule. Precisely, the object is removed from the system and a new object is created, with the same values of the fields as those of o but for the fields of *out_fields* different from `same`, which are rewritten with the new values specified in the rule. So doing it is possible to simulate *port forwarding*, an important feature of firewalls that allows to redirect the incoming traffic towards the appropriate server, which is supposed to be in a specific zone of the network (unknown to anyone located outside). The application of a `reject` rule is similar: the incoming object is removed from the system and a new special object, representing an ICMP error packet, is created. The source and destination IP addresses of the new packet are exchanged with respect to the packet given as input, so that the new packet goes back to the sender to signal that its previous IP packet has been rejected. The destination port is set equal to the source port of the original packet, whereas the source port is simply put to 0. Since all these fields of the resulting packet are so determined, when writing a `reject` rule the only argument of *out_fields* which is meaningful is *gateway*, which indicates the direction to be followed to go back to the sender; all the other fields are ignored. The application of a `drop` rule simply removes the object from the system, without producing any new object. In this case, the argument *out_fields* can be omitted.

When a new object has been created as the result of the application of a rule, it is put in the region enclosed by the membrane, ready to be processed by the next list of rules. Such a list is located at the end of the edge which is uniquely determined by the *gateway* parameter; the presence of this field thus allows to simulate also *routing tables*, and is particularly useful when two regions are connected by two or more edges. Of course there may already be other objects waiting to be matched against these rules; at the next computation step, one of these objects will be chosen in a nondeterministic way and will be processed, according to the *sequential* mode of applying the rules. Since no object may be processed by two or more lists of rules in the same computation step, each list can be operated *in parallel* with the others. This situation is similar to what happens with spiking neural P systems [8], where each neuron applies its rules in the sequential way but all neurons work in parallel.

We conclude this section by giving the formal definition of our model of P systems. A *firewall P system*, of degree $m \geq 1$, is a tuple $\Pi = (O, Z_1, \dots, Z_m, conn)$, where:

- O is the set of *objects*, which represent IP packets as described above;
- $Z_1, \dots, Z_m$ are the *membranes* (cells) of the system, each representing a zone of the network. Each membrane is a tuple $Z_i = (w_i, L_{i_1}, \dots, L_{i_k})$, where w_i is the multiset of objects initially present in the membrane, and k is the number of incoming edges (in-degree) of Z_i. For every $j \in \{1, 2, \dots, k\}$, L_{i_j} is a set of

rules, associated to Z_i and to the j-th incoming edge, having the form described above:
  - (*priority,* `accept`*, in_fields, out_fields*)
  - (*priority,* `reject`*, in_fields, out_fields*)
  - (*priority,* `drop`*, in_fields*)
- $conn = \big\{\{i,j\} : i,j \in \{1,2,\ldots,m\}$ and $i \neq j\big\}$ is the multigraph (that is, a multiset of edges) of *connections* between the membranes.

A *configuration* of a firewall P system Π is described by the multisets of objects contained in its membranes. The *initial configuration* is the one in which membrane Z_i contains the multiset w_i, for all $i \in \{1,2,\ldots,m\}$. A *computation step* changes the current configuration by applying the rules as described above. In particular, we recall that at every computation step each list of rules chooses in a nondeterministic way an object among those which have to be processed by such rules. Moreover, the lists operate with *maximal parallelism*: if at least one object exists which has to be processed by a given list of rules, then one of these objects *must* be chosen and matched against the rules. A *final configuration* is a configuration in which no rule can be applied. As usual, a *computation* starts from an initial configuration and produces configurations by applying computation steps. The computation *halts* if it reaches a final configuration. By identifying an *input membrane* Z_{in} and an *output membrane* Z_{out}, with $in, out \in \{1,2,\ldots,m\}$, we can define a computation device that transforms input multisets into output multisets: the input of the computation is w_{in}, whereas the output is the contents of membrane Z_{out} in the final configuration, if it is reached. Non-halting computations produce no output. By considering the Parikh vectors associated with multisets, we immediately obtain also a computation device that transforms vectors of natural numbers into vectors of natural numbers.

4 One Membrane Suffices

Let us give now some insight on the computational power of firewall P systems.

As stated in the Introduction, we envision that our systems will be used to mathematically prove some security properties. Such proofs will be obtained by considering *reachability* problems. So, for example, we could prove that a certain kind of TCP packets will never reach a specified zone of the network by showing that no configuration which can be reached from the initial configuration contains that kind of packets in that zone. This means that our P systems should *not* be Turing-complete, otherwise the reachability problem would be undecidable. However, it is easily proved that firewall P systems are not universal: since objects are never created (rules can only modify an object or remove it from the system, and no object can enter the system from the environment during the computation, as it happens with tissue P systems), no output which contains more objects than those given in the initial configuration may be produced.

Establishing the precise computational power of firewall P systems is left as an open problem. In this section we just prove that firewall P systems can be simulated by single-membrane transition P systems using non-cooperative rewriting rules with priorities and catalysts. If the simulated firewall P system would operate in the *sequential* mode (meaning that lists of rules do *not* work in the maximally parallel way) this would mean that they would generate at most the Parikh images of ET0L languages [18]. However, in order to correctly simulate the maximally parallel application of the lists of rules — while maintaining sequentiality between the rules of the same list — we have to use a catalyst for each list of rules.

Theorem 1. *Firewall P systems can be simulated by single-membrane transition P systems using non-cooperative rewriting rules with priorities and catalysts.*

Proof. Let $\Pi = (O, Z_1, \ldots, Z_m, conn)$ be a firewall P system, where $Z_i = (w_i, L_{i_1}, \ldots, L_{i_k})$. We build a transition P system $\Pi' = (A, \mu, w, R)$ that simulates Π as follows. The alphabet A is composed of objects which can be seen as seven-tuples

$$(\textit{region, protocol, src\_IP, src\_port, dst\_IP, dst\_port, gateway})$$

where (*protocol, src_IP, src_port, dst_IP, dst_port, gateway*) is an object of Π, and *region* keeps track of the region which contains the object. Moreover, alphabet A contains also a symbol $\sharp \notin O$, and a symbol $c_{i,j} \notin O$ for each set of rules L_j of simulated membrane Z_i. The membrane structure μ is composed by a single membrane, the skin. The multiset w of the objects initially present in the skin membrane is built from the multisets w_i by adding to each object $o \in w_i$ the new value of the *region* component (that, for w_i, is equal to i). This initial multiset also contains a single copy of each symbol $c_{i,j} \in A$.

The set R of rewriting rules is obtained from the lists of rules of Π as follows. Let $r_{i,j} = (\textit{priority}, \texttt{accept}, \textit{in\_fields}, \textit{out\_fields})$ be an `accept` rule of Π, associated with membrane i and its incoming edge j. This rule produces a set $R_{i,j}$ of rewriting rules in Π', where each rule is obtained by specifying a source and a destination IP address, taken from the subnets specified in *in_fields* in all possible ways. For every pair (*src_IP, dst_IP*) of IP addresses a rule $ac_{i,j} \to bc_{i,j} \in R_{i,j}$ is generated, where a is the object that represents the IP packet being analyzed, and b represents the packet modified according with the values of *out_fields*. Object $c_{i,j}$ is a catalyst, unique to membrane i and its incoming edge j, which is used to make the application of rules from the same list sequential; indeed, for all i and j, system Π' contains a single copy of $c_{i,j}$. The value of *region* in b is i, while the value of *region* in a is put equal to the membrane which is connected to membrane i through its incoming edge j. The priority of the rule is set equal to the value of *priority* from $r_{i,j}$. Each `reject` rule produces an analogous set of rewriting rules, whereas all `drop` rules generate rewriting rules in which the output object is $\sharp$, that does not appear in the left hand side of any rule. The set R of rules is obtained as the union of all sets $R_{i,j}$ thus generated.

The system Π' thus generated operates in the maximally parallel way. Due to the presence of catalysts, the rules occurring in the same list of Π are simulated by Π' in the sequential way. It is not difficult to see that each computation step of Π corresponds to a computation step of Π', and hence that Π' simulates Π. $\square$

Please note that in firewall P systems we can have rules with wildcard patterns when the fields *src_IP/subnet, dst_IP/subnet* have a subfield *subnet* of value less than 32, or anywhere a value `any` is used in a field, but these wildcards will have matches only over a predefined finite set of values (valid IP addresses, protocols, ...). Ours is therefore just a syntactic short notation for finite sets of usual P rules, and this comes into play also in the previous proof, when building the set of rules of the simulating transition P system.

Remark.

The possibility of modeling a network of firewalls by an equivalent single formal element has been studied also in a completely different technical context, when using SAT instances as a representation of the system [9]. Moreover, this property suggests the interesting perspective of having actual network systems where all the subnets are seen as a single region. In the actual implementation of such a system, we could consider to write in the header of IP packets the value of the *region* component used by our single membrane model, and anywhere a packet would arrive, it would be filtered/transformed by active elements of the network. This technological approach could enhance network security, since filtering would no longer happen only in firewalls on the borders, with the trouble of them being "single points of failure", but consistently anywhere on the network.

5 Conclusions and Directions for Future Work

In this paper we have modeled the functioning of firewalls, routers, and other simple rule filters by a tissue-like model of P systems. After a description of the features of this model, we have formally defined it and we have shown that it can be easily simulated by a single-membrane P system. We can thus argue that what is seen as an important and difficult problem in the practice of computer networks (analyzing and understanding long lists of filtering rules) is indeed a very simple task in the theory of membrane systems.

Future work includes further analysis of the features of packet filters and of the properties of the corresponding P systems. In particular, it should be interesting to see how the algorithms currently proposed in the literature (such as those in [3, 4, 5]) to solve common firewall misconfiguration problems map themselves to the model of P systems we have proposed. Also some new algorithms may be devised, based upon the new point of view given by P systems. Since many simulators of P systems already exist, an interesting development is to test the application of P systems described in the current paper against real cases.

Acknowledgements. This work was partially supported by the Italian project FIAR 2008 "Modelli di calcolo naturale e applicazioni alla systems biology".

References

1. P. Eronen, J. Zitting. An expert system for analyzing firewall rules. In *Proceedings of the* 6th *Nordic Workshop on Secure IT Systems (NordSec 2001)*, 2001, pp. 100–107.
2. J.D. Guttman. Filtering postures: local enforcement for global policies. In *Proceedings of the 1997 IEEE Symposium on Security and Privacy*, 1997.
3. A. Hari, S. Suri, G. Parulkar. Detecting and resolving packet filter conflicts. In *Proceedings of IEEE INFOCOM 2000*, 2000, pp. 1203–1212.
4. S. Hazelhurst. Algorithms for analysing firewall and router access lists. *Technical Report TR-Wits-CS-1999-5*, Department of Computer science, University of the Witwatersrand, South Africa, 1999.
5. S. Hazelhurst, A. Attar, R. Sinnappan. Algorithms for improving the dependability of firewall and filter rule lists. In *Proceedings of the International Conference on Dependable Systems and Networks (DSN 2000)*, IEEE Computer Society Press, 2000, pp. 576–585.
6. S. Hazelhurst, A. Fatti, A. Henwood. Binary decision diagram representations of firewall and router access lists. *Technical Report TR-Wits-CS-1998-3*, Department of Computer Science, University of Witwatersrand, South Africa, 1998.
7. K. Hundley, G. Held. *Cisco access lists field guide.* McGraw-Hill, 2000.
8. M. Ionescu, Gh. Păun, T. Yokomori. Spiking neural P systems. *Fundamenta Informaticae* **71**(2-3):279–308, 2006.
9. A. Jeffrey, T. Samak. Model checking firewall policy configurations. In *Proc. of the 2009 IEEE Symposium on Policies for Distributed Systems and Networks*, 2009, pp. 60–67.
10. C. Martín Vide, J. Pazos, Gh. Păun, A. Rodríguez Patón. A new class of symbolic abstract neural nets: tissue P systems. *Computing and Combinatorics,* 8th *Annual International Conference, COCOON 2002*, LNCS 2387, Springer, Berlin, 2002, pp. 290–299.
11. C. Martín Vide, J. Pazos, Gh. Păun, A. Rodríguez Patón. Tissue P systems. *Theoretical Computer Science*, **296**(2):295–326, 2003.
12. A. Mayer, A. Wool, E. Ziskind. Fang: A firewall analysis engine. In *Proc. of the 2000 IEEE Symposium on Security and Privacy*, 2000, pp. 177–187.
13. The NETFILTER/IPCHAINS/IPTABLES web page: `http://www.netfilter.org/`
14. The P systems web page: `http://ppage.psystems.eu/`
15. Gh. Păun. Computing with membranes. *Journal of Computer and System Sciences*, **61**:108–143, 2000.
16. Gh. Păun. *Membrane computing. An introduction.* Springer, 2002.
17. Gh. Păun, G. Rozenberg. An introduction to and an overview of membrane computing. In Gh. Păun, G. Rozenberg and A. Salomaa (eds.), *The Oxford Handbook Of Membrane Computing*, Oxford University Press, 2010, pp. 1–27.
18. D. Sburlan. Non-cooperative P systems with priorities characterize PsET0L. In *Membrane Computing,* 6th *International Workshop, WMC 2005*, LNCS 3850, Springer, 2006, pp. 363–370.
19. A.S. Tanenbaum. *Computer networks.* Prentice Hall, 2002.