A trust model applied to e-mail servers

Leonardo Oliveira and Carlos Maziero
Graduate Program in Computer Science
Pontifical Catholic University of Paraná, Brazil
{*bispo,maziero*}*@ppgia.pucpr.br*

Abstract

E-mail services are essential in the Internet. However, the basic e-mail architecture presents problems that opens it to several threats. Alternatives have been proposed to solve some problems related with e-mail services, offering reliability and scalability to those systems. This work presents a distributed trust model, allowing to create dynamic and decentralized trusted server lists, through the exclusion of servers used as spreaders of malicious messages. Many techniques were used to build it, like a social network model, message filters, message management, and a trust information storage and propagation model.

1 Introduction

E-mail systems are commonly used by its simplicity, flexibility, and low costs for implementation and usage. However, this kind of system suffers problems caused by fragilities of protocols involved on communication process. Amongst them are lacks of a robust mechanism for authentication, lacks of a confidentiality and integrity mechanism for message delivery, and also lacks of a reputation mechanism of users and e-mail servers.

E-mail systems researches are trying to solve this kind of fragilities using filtering and classification algorithms, which look for malicious patterns inside the message content, using symmetric keys on e-mail clients, to sign and cypher sent messages, and methods for e-mail servers authentication. These researches resulted in important advances in the attempt to solve problems related with authentication, but does not provide mechanisms to measure how trustful a server or domain is. In other words, authentication mechanisms by themselves are not able to minimize the sending of malicious messages (spam).

This work presents a distributed trust model for e-mail servers that uses e-mail classification techniques, a sender authentication model, and social networks, to create an environment able to keep information about legitimate/malicious e-mail servers using a decentralized method. Section 2 brings a short review of main threats in e-mail services. Section 3 describes the main techniques used to authenticate the e-mail senders; section 4 describes the use of social networks in a distributed environment, section 5 shows the proposal architecture and details its functional aspects; section 6 presents the model implementation and discusses some experimental results; finally, section 8 concludes this work and delineates some perspectives of continuity.

2 Threats in e-mail systems

Today e-mail systems were designed to be simple, their main target was restricted to a small and trustful environment, constituted basically by the academic community. The SMTP protocol, responsible for e-mail transferences between servers [4, 8] does not provide robust mechanisms for authentication and access control. Amongst the current problems present in e-mail systems are *spams*, virus and *scams*, which compromises their performance, robustness, security, and usability.

Many techniques are being used for *spam* control. The main techniques are based on trustful and non-trustful servers lists, or on screening received messages to find suspect content:

- *Black Lists*: distributed *RBL* (*Realtime Blackhole Lists*) servers keep lists of IP address from *spam* spreaders or sources, which can queried by DNS to verify the sender trustworthiness [7].
- *White Lists*: each e-mail server can keep a trusted senders list; this list is commonly kept through a web-based acknowledge mechanism [6].
- *Anti-spam Filters*: Programs that filter e-mail according to its content, using statistical techniques, Bayesian classification, neural networks, etc. [10, 15].

A second form of threat are virus and *worms*, which are malicious programs that can spread though e-mail. An e-

mail virus normally is constituted by an e-mail with an executable file attached (or a HTML code able to load an executable file stored in a remote server). This code, which can be activated by the receiver or automatically loaded, aims to reproduce the virus and to do other local actions, like damaging system files, installing spyware applications, and so on. Finally, the *phishing scams* or simply *scams* are fake messages that use the SMTP protocol fragilities to build social engineering attacks. Such attacks aim to deceive e-mail receivers, convincing them to inform personal data like bank informations, credit card numbers, etc.

3 Sender authentication

E-mail senders can be authenticated, in order to guarantee the origin of a message. Many techniques have been developed with this goal, like *PGP* (*Pretty Good Privacy*), *SPF* (*Sender Policy Framework*), *SenderID* and *DKIM*. *PGP* [14] is a cryptography package that makes use of public keys and session keys to cypher documents. Thus, only the person owning the private key can decrypt the session key, guaranteeing then the privacy and integrity of a message.

SPF [13] makes use of *DNS* servers to provide to a receiver server informations about the sender server. These informations are described in a proper language and stored in a specific *DNS* record. When the server receives a message, it makes a DNS query to get information about the origin domain, to verify the sender authenticity. *SenderID* [12] is a *SPF*'s extension that improves the *SPF* language and extends the *SMTP* protocol, to solve problems related with forwarded messages authentication.

DKIM (*Domain Keys Identified e-Mail*) [5, 1] uses the public key system to validate the authenticity of an e-mail server. The private key is stored in a local database, and it is used to generate a signature for every e-mail sent, included in its header. The public key is stored in a *DNS* record and can be requested by the receiver server, to validate the signature.

All these techniques provide sufficient information to identify the sender of an e-mail, but they do not allow to judge about his/her trustworthiness. Our proposal in this paper is to combine one of such techniques to a trust model, in order to provide more information about e-mail senders.

4 Trust relationships

In human relations, trust is the base for the construction and maintenance of a group of individuals. Trust relations define how a person acts regarding other known persons and strangers. Trust relations can be classified as *Hierarchic trust*, *Social groups* and *Social networks* [11].

Hierarchic trust considers all relationships as an hierarchic model, for instance the trust of a father on his son. This trust relation can be represented as a tree, where all nodes are individuals and the edges define the trust degree between each pair of participants. Through transitivity, any two participants can define a trust degree between them, event if they have no direct connection.

Social groups define sets of individuals whose activities are systematically related with a well-defined goal. Participants of social groups are able to share common interest information, propagating it to all other participants. Participants can also make use of *trust groups* to start and keep relationships with other participants. Social groups can be represented as a graph, where all nodes are participants and the edges are links between individuals.

The *Relationship* word will define the interaction between individuals, based on reciprocal perceptions. *Social networks* are built from all relationships established by a person. It is possible to walk on the social network to find relationship paths between persons.

Computational systems can use the same principles of interpersonal relationships to define trust levels between its diverse elements [9]. In the following we will present a proposal that applies concepts of trust relationships among e-mail servers, aiming to diminishing the quantity of malicious e-mails received by the servers in the group.

5 A trust model for a group of e-mail servers

This work defines an architecture able to create dynamic and decentralized lists of trusted servers, using e-mail classification and e-mail server authentication techniques. We consider that e-mail servers are organized in trust groups $\mathcal{T}$, defined by their administrators. For example, considering the e-mail servers present in figure 1, the trust group of s_4 has s_2, s_3, s_5 and s_9 ($\mathcal{T}_4 = \{s_2, s_3, s_5, s_9\}$).

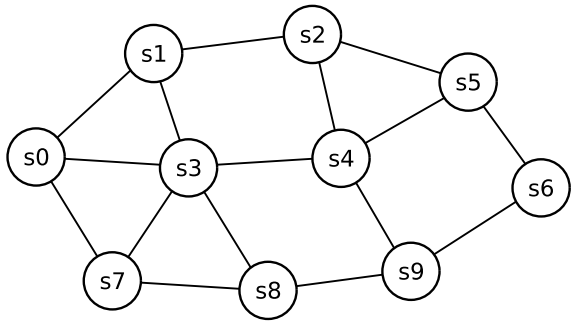

Figure 1. Trust groups

Servers that constitute a trust group cooperate between each other to share information about outside servers known

by the group, using trust network techniques. As many trust groups are distinct and partially overlapped, trust information will be gradually propagated, passing from a trust group to another.

5.1 Architecture

The system architecture use concepts of trust networks, anti-spam tools, anti-virus tools, and an authentication model to build a trust management system among e-mail servers (*Mail Transport Agents* - MTAs). Figure 2 represents all components that will be implemented/integrated in each participating MTA.

- **SMTP Server**: responsible for messages reception; implements the SMTP protocol.
- **Sender authentication**: implements a domain authentication method, as *SPF* or *DKIM*.
- **Anti-spam and anti-virus**: analyses if a message is legitimate or malicious. The filter results are used by the trust management system.
- **Trust management system**: analyses and keeps the trust on a given server, according to the messages received from it.

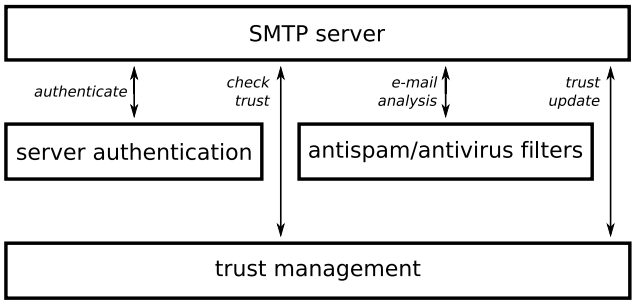

Figure 2. Architecture model

When a SMTP communication is established between two MTAs, the receiver server should execute the following steps for each received message (as indicated in figure 2):

1. Authenticate the server: if the server is correctly authenticated, go to next step, otherwise, reject the message and close the connection;
2. Verify if the sender server can send messages, querying the trust management system (the rules for this will be defined in section 5.2). If the response is positive, the system receives the message, otherwise the connection is closed;
3. Submit the received e-mail to anti-spam and anti-virus filters. These filters will inform whether the received e-mail is legitimate or malicious;
4. Send the filters results to the trust management system, to update the local trust information.

The architecture was divided in three modules: *management*, *storage* and *trust propagation*, presented in the following.

5.2 Trust management

Each server from a trust group executes actions to maintain trust information about other e-mail servers external to its group. For each external server, it maintains a *local trust*, defined by internal information (received messages), and a *global trust*, calculated using information provided by other members from its trust group. The combination of these two trusts define a *final trust*, used to determine the maximum number of messages that can be received from this server in during a given period. An external server exceeding this limit will be *banned*, no more messages will be received from it until the beginning of a new period. The period duration is defined by the system administrator (in minutes, hours, days, etc.). Main attributes involved on trust calculation are described in Table 1 (all trust values are between 0 and 1, or 0% and 100%).

Table 1. Trust calculation information

Entity	Description
$\mathcal{T}_i$	Trust group of the SMTP server s_i.
$\mathcal{K}_i$	Set of external SMTP servers known by s_i. Servers belonging to its trust group are not external: $\mathcal{K}_i \cap \mathcal{T} = \emptyset$.
$tl_i^j(x)$	Local trust: last value known by s_i of individual opinion of s_j about a server $x \in \mathcal{K}_j$ (each server s_i locally stores values $tl_i^*(x)\ \forall x \in \mathcal{K}_i$.
$tg_i(x)$	Global trust: shows the opinion of the trust group about a server x, calculated by s_i using all local trusts from its trust group servers.
δ_t	Trust variation stepping.
t_0	Initial trust to be attributed to new servers, empirically defined as 50%.
$age_i(x)$	age of data about server x kept by s_i.
age_{max}	maximum value for $age_i(x)$.
$mm_i(x)$	Number of malicious messages received from x server by s_i in the current period.
$ml_i(x)$	Number of legitimate messages received from x server by s_i in the current period.
mm_{max}	Number of malicious messages received during a period to reduce by δ_t the trust on a given server, considering a initial local trust of 100%. This value is defined by the system administrator.
$banned_i(x)$	indicates if server x was banned by s_i; if true, all messages from x will be rejected by s_i until the current period finishes.
lim	Number of legitimate/malicious messages necessary to increase/decrease by δ_t the local trust about a server.
tc	Current trust, used on the lim calculation. Defined as the geometry mean of local trust tl and global trust tg.

The procedure 1 describes all actions realized by a server $s_i \in \mathcal{T}_i$ when receive a connection of a SMTP server $x \notin \mathcal{T}_i$ to deliver an e-mail.

Procedure 1 when s_i receives a connection from server x:

1: **if** $x \notin \mathcal{K}_i$ **then**
2: $\mathcal{K}_i \leftarrow \mathcal{K}_i \cup \{x\}$ // x becomes a known server
3: $banned_i(x) \leftarrow$ FALSE
4: $tg_i(x) \leftarrow t_0$ // initial trust
5: $ll_i^i(x) \leftarrow t_0$
6: $age_i(x) \leftarrow 0$
7: $mi_i(x) \leftarrow 0$
8: $mm_i(x) \leftarrow 0$
9: **end if**

The local trust is reevaluated for each e-mail reception. If the number of malicious messages received from a given server exceeds a maximum limit, the server will be banned until the end of current period. Procedure 2 describes the actions performed by a server $s_i \in \mathcal{T}_i$ when it receives a message m from a SMTP server $x \notin \mathcal{T}_i$:

Procedure 2 when s_i receives a message m from server x:

1: **if** $banned_i(x)$ **then**
2: rejects the message m
3: **else**
4: accepts the message m
5: $age_i(x) \leftarrow 0$
6: $tc \leftarrow \sqrt{tg_i(x) \times tl_i^i(x)}$
7: analyzes m contents
8: **if** $msg\_legitimate(m)$ **then**
9: $mi_i(x) \leftarrow mi_i(x) + 1$
10: $lim \leftarrow tc \times mm_{max}$
11: **if** $(mi_i(x) \geq lim) \wedge (tl_i^i(x) < 1)$ **then**
12: $tl_i^i(x) \leftarrow tl_i^i(x) + \delta_t$
13: $mi_i(x) \leftarrow 0$
14: send $notify(x, tl_i^i(x))$ to trust group $\mathcal{T}_i$
15: **end if**
16: **else**
17: $mm_i(x) \leftarrow mm_i(x) + 1$
18: $lim \leftarrow (1 - tc) \times mm_{max}$
19: **if** $(mm_i(x) \geq lim) \wedge (tl_i^i(x) > 0)$ **then**
20: $tl_i^i(x) \leftarrow tl_i^i(x) - \delta_t$
21: $mm_i(x) \leftarrow 0$
22: $banned_i(x) \leftarrow$ TRUE
23: send $notify(x, tl_i^i(x))$ to the trust group $\mathcal{T}_i$
24: **end if**
25: **end if**
26: **end if**

Procedure 2 shows that, when a local trust on x decreases, it is banned until the end of current period. The adoption of dynamic thresholds (using the tc and lim attributes) allows to decrease faster the trust on lowly-trusted servers than increasing it, and to increase faster the trust on highly-trusted servers than decreasing it. The limit lim on the number of messages that the server s_i can accept from the server x per period is calculated in line 10 (if the message m is legitimate) or in line 18 (if not). In both cases, the limit is calculated as a function of s_i' current trust tc on the server x.

Finally, when a period finishes, each server should execute the actions described in procedure 3. Such actions will re-enable banned servers, restart counters and "forget" information about servers that no more sent e-mails to s_i for a while. This forgetting mechanism is important, as it allows to consider only the recent behavior of external servers.

Procedure 3 when s_i finishes a period:

1: **for all** $x \in \mathcal{K}_i$ **do**
2: $banned_i(x) \leftarrow$ FALSE
3: $mi_i(x) \leftarrow 0$
4: $mm_i(x) \leftarrow 0$
5: $age_i(x) \leftarrow age_i(x) + 1$
6: **if** $age_i(x) = age_{max}$ **then**
7: $\mathcal{K}_i \leftarrow \mathcal{K}_i - \{x\}$ // "forget" server x
8: remove all local information about x
9: send $notify(x, undef)$ to the trust group $\mathcal{T}_i$
10: **end if**
11: **end for**

5.3 Trust storage

The trust information is locally stored in each e-mail server belonging to a trust group. The following information is maintained in s_i local database, for each known server $x \in \mathcal{K}_i$:

- domain of x;
- domain name of x (*FQDN - Fully Qualified Domain Name*);
- x IP addresses;
- local trust $tl_i^*(x)$ and global trust $tg_i(x)$;
- counter of legitimate messages $mi_i(x)$ and malicious ones $mm_i(x)$ received from x during the current period;
- Age $age_i(x)$ of the local information about x.

5.4 Trust propagation

The trust propagation mechanism spreads out information about servers known by s_i to all members of its trust group $\mathcal{T}_i$. Trust groups are defined by the e-mail service administrator and will indicate which servers should be informed about trust updates on external servers (in this proposal, trust groups are manually defined).

As defined in procedure 2, when a server updates its local trust about an external server, it notifies its trust group servers, using a $notify$ message. When a server s_i receives a message $notify(x, t)$ from s_j, it updates its local information about x, as shown in procedure 4. Global trust is defined as the mean of local trusts on x for the servers belonging to $\mathcal{T}_i$; it is reevaluated by s_i after each period (procedure 5). If any server $s_j \in \mathcal{T}_i$ did not inform its opinion about x ($tl_i^j(x) = undef$), its value will not be considered.

Procedure 4 when s_i receives $notify(x, t)$ from s_j:

1: **if** $(j \in \mathcal{T}_i) \wedge (x \in \mathcal{K}_i)$ **then**
2: $tl_i^j(x) \leftarrow t$
3: **end if**

Procedure 5 when s_i finishes a period:

1: **for all** $x \in \mathcal{K}_i$ **do**
2: $tg_i(x) \leftarrow mean(tl_i^j(x) \neq undef, \forall s_j \in \mathcal{T}_i)$
3: **end for**

6 Implementation and results

We developed a prototype of our proposal (called *TrustMail*) using *Linux*, the *Postfix* e-mail server, the *SpamAssassin* anti-spam filter, and the *Clamav* anti-virus filter. In the following, we will describe our implementation, the communication system between e-mail server and other components, and a short evaluation of the system's behavior.

6.1 *TrustMail* prototype

The prototype was implemented using C language and the SQLite database to store local trust information. Communication between *TrustMail* and other parts of the system (e-mail reception and filtering) is made through POSIX message queues.

Postfix 2.2 was chosen as MTA due to its easiness of integration with other tools. The authentication system chosen was SPF, because its implementation is simple and easy to integrate with *Postfix*. The communication between the MTA and SPF, and the communication with *TrustMail*, are done by *policyd*, an open source SPF implementation (*Policyd* was modified to support communication with *TrustMail*).

SpamAssassin was chosen as anti-spam filter, because it is the *de facto* standard on Linux. The same applies to the *Clamav* anti-virus, which has a huge virus signature list. Both run as *daemons*, turning easy the integration with other tools. The integration between *Postfix* and the messages filters was done by the *Clamav-Filter* script, modified to support malicious message notifications. The current prototype implementation is illustrated in figure 3. In this figure is possible to see that *TrustMail* does not depend on the e-mail server implementation, authentication system, nor messages filters. This way, the proposed model could be easily integrated with other tools.

The following procedure is performed when receiving an e-mail:

1. When receiving a `RCPT TO` SMTP command, *Postfix* invokes *policyd* with the following parameters: sender address, sender IP and domain.

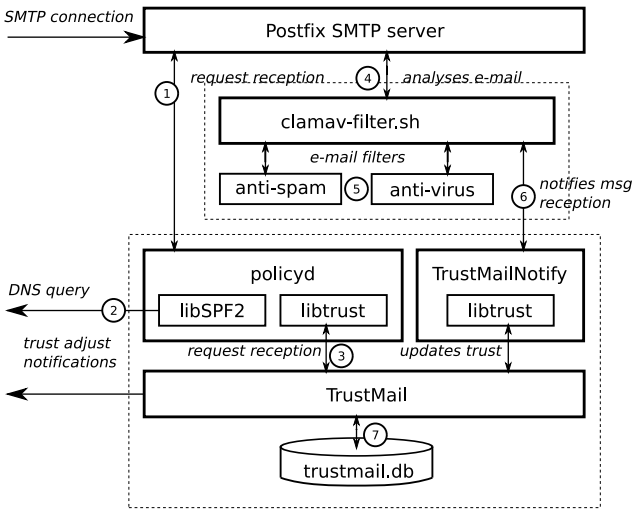

Figure 3. The prototype architecture

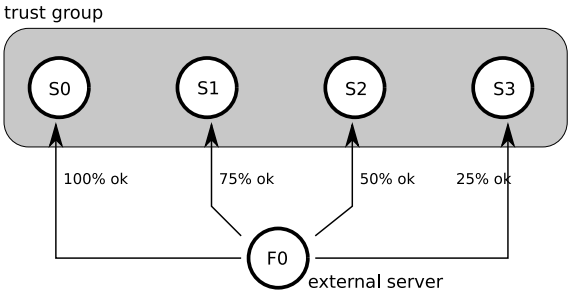

Figure 4. Experimental topology

2. *Policyd* makes a query to the DNS server of the e-mail sender domain, in order to authenticate it. The query return can be:
   - *Pass*, *Soft Fail*, *Neutral*, *Unknown* or *None*: the procedure can continue;
   - *Error*: Returns the message "450 Temporary failure" to *Postfix*;
   - *Fail*: Return an error and the connection should be closed by *Postfix*.
3. If the server is authenticated, *policyd* calls *TrustMail* to request if the connection can continue. *TrustMail* calculates the maximum number of e-mails that can be received from that server and responds to *policyd*. *Policyd* then sends back to *Postfix* the response received.
4. After the e-mail is received, *Postfix* invokes the *Clamav-Filter* script, with the following parameters: IP address, name and domain of the sender server.
5. *Clamav-Filter* then passes the e-mail through the *Clamav* and *SpamAssassin* filters.
6. Next, the *Clamav-Filter* calls the *TrustMailNotify* module, which notifies *TrustMail* that a malicious or legitimate message was received.
7. Finally, *Trustmail* updates the trust information in stored in a *Trust_Mail.db* database.

6.2 Experimental results

The prototype was implemented and tested in a virtual machine environment (using UML (*User-Mode Linux*). The experiment used the topology shown in figure 4, in which four virtual machines implement the trust group ($s_0 \ldots s_3$). Another virtual machine was used to simulate an external e-mail server (e_0), which sends e-mails to the servers in the group

On the experiment, the external server sends e-mails continuously and randomly (on average one e-mail each 15 seconds), uniformly distributed among group servers. The external server has a distinct behavior concerning each server in the group: it never sends a malicious e-mail to s_0, but 25% of e-mails sent to s_1, 50% sent to s_2 and 75% sent to s_3 are malicious. The chosen period duration was 30 minutes, $age_{max} = 10$ periods, $t_0 = 50\%$, $\delta_t = 10\%$ and $mm_{max} = 10$ messages. The total execution duration was 24 hours.

To evaluate the group influence on the decisions of each each server, we observed the evolution of local and global trust about e_0 ($tl_i^i(e_0)$ and $tg_i(e_0)$) for each group member, on two circumstances: without trust update notifications (that is, without group cooperation) and with such notifications. This evaluation is presented in figure 5. It is possible to see that, if there is no cooperation between group members (curves $tl(s_i, x)$), each server will build its own opinion (trust level) about the external server e_0. When notifications are used, cooperation between group members occurs (curves $tg(s_i, x)$), and trust opinions on e_0 converge to similar values.

In the same experiment, we evaluated the quantity of messages rejected by the banishment of non-trusted servers (line 22 of procedure 2. The results are presented in figure 6. It is possible to see that, excepting s_0 (that did not receive spam), all the other servers reject significant amounts of the e-mails sent to them. This show that the banishment mechanism limits the quantity of messages that an external server can send to each group member, when the trust in it is not 100%.

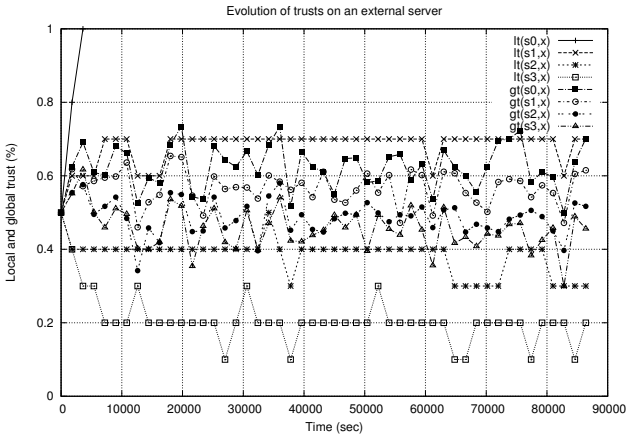

Figure 5. Local and global trust evolution

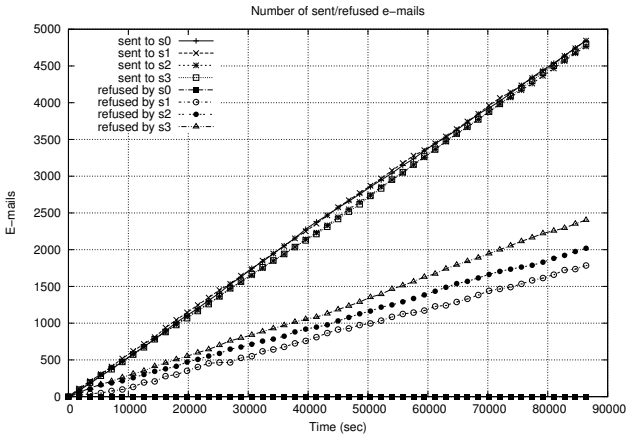

Figure 6. Number of messages sent/rejected by servers

7 Related works

Not much research was proposed using social networks to fight spam. The following works are considered by us most relevant.

The *MailRank* [3] system is a tool for global *white-lists* construction. Data about user activity are evaluated and grouped in a global social network. Each message sent by an user to another is transformed in a trust vote, used to build the social network. The system is compatible with the current e-mail structure, and is composed by two basic elements: the *MailRank Proxy*, used as a proxy between each e-mail client and its e-mail server, to extract information from e-mails sent and received by each user, and the *MailRank Server*, a central server that collects data from all MailRank proxies, to create a global classification of users based on their sent and received e-mails.

By other hand, the work presented in [2] shows the construction of an anti-spam tool that extracts relationship information between e-mail users through the analysis of e-mail headers (*From*, *To*, *Cc*, *Bcc*, etc.). This information is used to build a huge graph of relationships among users. This graph is then used to build *white-lists* for all users, using a social network property called *agglomeration*. Finally, the agglomeration groups are analyzed to find spamming behavior patterns.

Our work differs from both in some aspects: when they classify users as spammers or not, our work concentrates on e-mail servers behavior, allowing a better scalability; moreover, our proposal presents a fully decentralized architecture, while the other works need some centralized server for the construction and analysis of the social network. Finally, our proposal is compatible and can coexist with the standard e-mail systems in use today.

8 Conclusion

This work proposed and evaluated a trust model for e-mail servers. It defines trust groups whose members interact to exchange "opinions" about external e-mail servers. Each server in a group uses the other members' opinions to build a global trust, and use this information to limit the quantity of e-mail received from an external server. Local trust information is propagated to the trust group using a social network model, decentralizing the maintenance and evolution of trust information.

Other aspects of this work that can be explored in future works are a) the definition of reputation between the members of a trust group, allowing the automatic inclusion or exclusion from members in a group; b) the definition of optimal values for the model's constants; and c) a deeper study of the trust spread mechanisms among distinct groups with common servers, in a scalable way.

References

[1] E. Allman, J. Callas, M. Delany, M. Libbey, J. Fenton, and M. Thomas. RFC 4871: Domainkeys identified mail (DKIM) signatures, May 2007.

[2] P. O. Boykin and V. P. Roychowdhury. Leveraging social networks to fight spam. *IEEE Computer*, 38(4):61–68, 2005.

[3] P. A. Chirita, J. Diederich, and W. Nejdl. Mailrank: Using ranking for spam detection. *ACM International CIKM Conference*, 2005.

[4] D. Crocker. RFC 822: Standard for the format of arpa internet text messages, Aug. 1982.

[5] M. Delany and Yahoo. Domain-based email authentication using public-keys: Advertised in the DNS (DomainKeys). Internet Draft, 2004.

[6] R. J. Hall. How to avoid unwanted email. *Communications of the ACM*, 41(3):88–95, 1998.

[7] J. Jung and E. Sit. An Empirical Study of Spam Traffic and the Use of DNS Black Lists. In *Internet Measurement Conference*, Taormina, Italy, October 2004.

[8] J. Klensin. RFC 2821: Simple mail transfer protocol, Apr. 2001.

[9] V. Krebs. The social life of routers: Applying knowledge of human networks to the design of computer networks. *Internet Protocol*, 3(4), Dec. 2000.

[10] M. Sahami, S. Dumais, D. Heckerman, and E. Horvitz. A bayesian approach to filtering junk E-mail. In *Learning for Text Categorization: Papers from the 1998 Workshop*, Madison, Wisconsin, 1998. AAAI Technical Report WS-98-05.

[11] S. Wasserman and K. Faust. *Social Networks Analysis: Methods and Applications*. Cambridge University Press, 1994.

[12] Wong, Microsoft, and Lentczner. The SenderID record: Format interpretation. http://www.ietf.org/internet-drafts/draft-ietf-marid-protocol-02.txt, 2004.

[13] M. Wong. Sender Policy Framework (SPF): A convention to describe hosts authorized to send SMTP traffic. http://db.org/drafts/internet/mengwong/spf/00/, 2004.

[14] P. R. Zimmermann. *The Official PGP User's Guide*. The MIT Press, 1995.

[15] C. C. Zou, W. Gong, and D. Towsley. Code Red worm propagation modeling and analysis. In *9th ACM conference on Computer and Communications Security*, pages 138–147, 2002.