Artificial Immune System For Collaborative Spam Filtering
Technical Report LCA-REPORT-2007-008, EPFL, September 2007.

Artificial Immune System For Collaborative Spam Filtering

Slavisa Sarafijanovic and Jean-Yves Le Boudec

EPFL, Switzerland
{slavisa.sarafijanovic, jean-yves.leboudec}@epfl.ch

Summary. Artificial immune systems (AIS) use the concepts and algorithms inspired by the theory of how the human immune system works. This document presents the design and initial evaluation of a new artificial immune system for collaborative spam filtering[1].

Collaborative spam filtering allows for the detection of not-previously-seen spam content, by exploiting its bulkiness. Our system uses two novel and possibly advantageous techniques for collaborative spam filtering. The first novelty is local processing of the signatures created from the emails prior to deciding whether and which of the generated signatures will be exchanged with other collaborating antispam systems. This processing exploits both the email-content profiles of the users and implicit or explicit feedback from the users, and it uses customized AIS algorithms. The idea is to enable only good quality and effective information to be exchanged among collaborating antispam systems. The second novelty is the representation of the email content, based on a sampling of text strings of a predefined length and at random positions within the emails, and a use of a custom similarity hashing of these strings. Compared to the existing signature generation methods, the proposed sampling and hashing are aimed at achieving a better resistance to spam obfuscation (especially text additions) - which means better detection of spam, and a better precision in learning spam patterns and distinguishing them well from normal text - which means lowering the false detection of good emails.

Initial evaluation of the system shows that it achieves promising detection results under modest collaboration, and that it is rather resistant under the tested obfuscation. In order to confirm our understanding of why the system performed well under this initial evaluation, an additional factorial analysis should be done. Also, evaluation under more sophisticated spammer models is necessary for a more complete assessment of the system abilities.

Key words: Artificial, immune, collaborative, email, spam, filtering, representation, hashing, similarity signatures.

[1] The authors are with EPFL, Lausanne, Switzerland. The work presented in this paper was supported (in part) by the National Competence Center in Research on Mobile Information and Communication Systems (NCCR-MICS), a center supported by the Swiss National Science Foundation, under the grant number 5005-67322.

1 Introduction

1.1 Related Work

One of main problems not solved by the existing similarity-hashing based [2] and other collaborative content filtering methods [3, 7, 11] is that the representation of the email content used for spam filtering is vulnerable to the random or aimed text additions and other text obfuscation, which leaves many obfuscated spam emails undetected.

Also, the results of the queries to the collaborative-filtering databases have usually very constrained impact to the detection decision. For example, when computing an email spamminess score, the product offered to the students and employees at EPFL (our university), which is based on SpamAssassin [9], weights the Razor's result [7] approximately the same as the rule "subject is all big letters", and much less then the Bayesian score [4]. This is probably done in order to avoid false detection of good emails, because the used similarity-hashing representation is not precise enough for distinguishing well spammy patterns from normal email content.

Although the general idea of exchanging the exact or similarity signatures derived from the emails for spam bulk detection has been well known for years [1, 5], we do not find solutions that successfully address the above explained problems.

Damiani et al. [2] investigate the vulnerability of the Nilsimsa representation [5] (used by DCC [3]) and show the results that suggest that the representation becomes completely non-useful if enough random text is added by the spammer to the different copies of the same original spam message. They also find that, in case when the hashing function used by the filters is known, the spammer can defeat Nilsimsa representation by adding a much smaller (20 times smaller) amount of text.

Interestingly, though their results show major weaknesses of Nilsimsa, the authors comment the results only in the region of small random additions for which the representation is still good, i.e. the additions being up to 3 times longer then the spammy message. Their comments were later misinterpreted by many people who cited their work as proof of the representation's strength. Nothing prevents the spammer from adding more text and moving into the region where the representation does not work well, which could happen already with the added random text 5 times longer then the spammy message. The problem here is that the signature is computed from all, or predefined but variable in length, parts of the email. This gives enough room to the spammer for effective add-random-text and/or add-chosen-text obfuscation. Our system is designed to avoid such problems.

Regarding the existing use of the artificial immune system algorithms for spam filtering, we find that both the email representation and the algorithms are crucially different from our solution. Oda and White [6] use a word-based representation. They compute scores based on both good and bad words present in the email, which is, the same as Bayesian filtering methods, vulnerable to the additions of good words or phrases.

The representation used by Secker et al. [8], another artificial immune systems based approach, is also word based and not resistant to the letter-level obfuscation because the exact matching is used. As their method takes into account bulk evidence per user bases, i.e. it uses accumulated emails of one user as the training set,

it discovers the repeated spam patterns and is good at finding repeated spam. On the contrary, their system has no built-in mechanisms to detect new spam content based on the bulkiness of spam messages, although bulkiness offers strong spam evidence that should surely be exploited.

Another type of content-based filtering is Bayesian filtering, originally proposed by Graham [4]. A good feature of Bayesian filters is that they adapt to the protected user's profile, as they are trained on the good and bad email examples of the protected user. The disadvantages are vulnerability to the addition-of-good-words attack and absence of mechanisms to exploit bulkiness of new spam. The system has only a "local" view of a new ongoing spam bulk.

Usually the Bayesian filtering and collaborative filtering are done separately, and then the results are combined, along with results from other methods, for the final decision making. It might be advantageous for collaborative filtering if some local spamminess processing is done before the information is exchanged for the collaborative filtering, which the existing systems do not take into account.

The only solution known to us that uses the signatures on the strings of fixed length is the work by Zhou et al. [11], a peer to peer system for spam filtering. However, their signatures are exact and are not similarity signatures, as required by the rest of their system to work. Even modest spam obfuscation is able to alter some of the bits of such generated signatures, which prevents their system from detecting spam bulks. Their analysis results in a different conclusion, because they use rather unrealistic obfuscation (which alters the created signatures with a very small probability) to test their solution.

1.2 Our approach

We design the antispam system using some analogies to the workings of the human immune system. The system consists of the "adaptive" part, which is used for collaborative content processing to discover spammy email patterns, and the "innate" part. Although the innate part is not discussed and evaluated in this paper, it is still important. It is assumed to consist of predefined and quick mechanisms, such as white lists, black lists, and rules, which could be used to instruct to the adaptive part to not process the email or to process it more intensively then usual. In this paper we explain and evaluate the adaptive part, as it can work alone.

One instance of our system is added to an email server and it protects email accounts of that server. The system preferably (but not necessarily) collaborates with a few other such systems.

The adaptive part produces so-called detectors that are able to recognize spammy patterns within both usual and heavily obfuscated spam emails. This is made possible by processing emails on the level of so called "proportional signatures": the text strings of the predefined length are sampled at random positions from the emails. They are further transformed into the binary strings using our custom similarity-preserving hashing, which enables both good differentiation of the represented patterns and their easy and robust similarity comparison.

The adaptive processing looks at the bulkiness of the proportional signatures and at the same time takes into account the users' profiles and feedbacks from standard users' actions, thus using a maximum of the available information for this so-called

collaborative content processing. The profile of the user is taken into account by excluding from further processing the proportional signatures that show similarity to the examples of the “good signatures”. Good signatures are created from the good emails received or sent by the user. Similar “processing” exists in the human immune system and is called negative selection. Then the local processing is done on the remaining signatures, the processing that takes together into account both the local bulkiness of the signatures and the feedback from the users deleting their emails as spam. Based on the results of this local processing, some of the signatures may be decided to be exchanged with other collaborating systems.

We assume that some of the users have and use the “delete as spam” button when they read their email, tough the system may work even if the assumption is released. Similar so-called “danger signal” feedback exists in the human immune system when there is damage to the body’s cells, and is used similarly as in the presented antispam system, to help activating the detection. For creating and activating the detectors, apart from the above explained evidence, the signatures obtained from other antispam systems are also accounted for when evaluating the bulkiness. Similar clustering of the chemical matches on the surface of the virus infected cells happens in the human immune system.

Thanks to the combination of the custom representation and the local processing, many good parts of the emails are excluded from further processing and from the exchange with other collaborating systems. This enables the bad (spammy) parts to be represented more precisely and better validated locally before they are exchanged. This increases the chances for the bad patterns to form a bulk and thus create a detector. They cannot be easily hidden by the spammer within the added obfuscation text as in the case with the classical collaborative filtering schemes.

The local clustering of the signatures makes the so-called recurrent detection feasible: the new emails are checked upon arrival, but also a cheap additional checking is done upon creation of new active detectors during the pending time of the email (before the user’s email client comes to pick it up). This further decreases non-detection of spam. The randomness in sampling and user-specific processing ensure the detectors to be diverse and unpredictable by spammers.

2 Description of the System

2.1 Where Do We Put the Antispam System

The antispam system, which filters the incoming e-mails for the users having their accounts on the same e-mail server, is placed in front of that e-mail server towards its connection to the Internet (Figure 1). This is the logical place of the filter, though the deployment details might differ a bit.

The antispam system designated to one e-mail server and its users can be an application added to the e-mail server machine, or it can be a computer appliance running such an application. A few such antispam systems can collaborate with each other, and each of them is also interfaced to the email server and accounts it protects. The collaboration to other antispam systems can be trusted, like in the case of few antispam systems administered by the same authority, or evaluated by the antispam

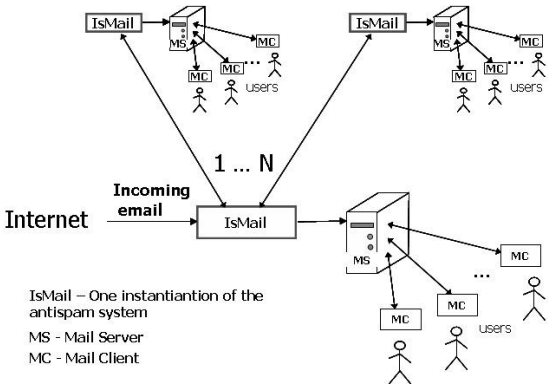


Fig. 1. The position of an antispam system with respect to other antispam systems, the protected email server and Internet.

system and correspondingly adapted, as it would probably be the case in a self-organized collaboration of antispam systems with no inherent mutual trust.

2.2 What the system does, inputs, outputs.

The antispam system decides for the incoming emails whether they are spam or not. If enough evidence is collected that an e-mail is spam, it is either blocked or marked as spam and sent to the e-mail server for easy sorting into an appropriate folder. Otherwise, upon a maximum allowed delay by the antispam system or upon a periodic or user-triggered send/receive request from the user's email client to the email server, the email is passed unchanged to the e-mail server.

The first-type inputs into the antispam system are incoming e-mail messages, before they are passed to the e-mail server.

The second-type inputs to an antispam system come from the access by the antispam system to the user accounts it protects. The antispam system observes the following email-account information and events for each protected email account: text of the e-mails that the user sends; text of the e-mails that the user receives and does an action on them; the actions on the e-mails processed by the antispam system and received by the user, i.e. not filtered as spam, including deleting a message, deleting a message as spam, moving a message to a folder; the actions on the e-mails processed by the antispam system and filtered as spam, which could happen very rarely or never depending on the user's behavior and performances of the antispam system; the send/receive request from the email client of the user to the e-mail server; email addresses from user's contacts. We assume that some of the users protected by the antispam system have "delete" and "delete-as-spam" options available from its e-mail client for deleting messages and use them according to their wish, but this assumption could be released and another feedback could be incorporated from the user actions on his emails, like moving the emails to good folder for example or simply deleting the emails. Here "delete" means move to "deleted messages" folder, "delete-as-spam" means move to "spam messages" folder. We also assume that all the e-mails that the user still did not permanently delete are preferably on the e-mail server, so the antispam system can observe the actions taken on them. Here "per-

manently delete" means remove from the e-mail account. The messages could be all moved to and manipulated only on the e-mail client, but then the client should enable all the actions on the e-mails to be observed by the antispam system.

The third-type inputs to the antispam system are messages coming from collaborating antispam systems. The messages contain useful information derived from the strings sampled from some of the e-mails that have been either deleted-as-spam by the users having accounts on the collaborating antispam systems or found by local processing as being suspicious to represent spammy part of an email from a new spam bulk. The third-type inputs to the antispam system are especially useful if there is small number of the accounts protected by the system. One of the factors that determine the performances of an antispam system is the total number of the active accounts protected by the antispam system and its collaborating systems.

The main output from the antispam system are the decisions for the incoming emails whether they are spam or not.

Another output are the collaborating messages sent to other antispam systems. These messages contain useful information derived from the strings sampled from some of the e-mails that has been deleted-as-spam by the users having accounts on the antispam system, or are locally found to be bulky. If the collaboration is self-organized and based on evaluated and proportional information exchange, the antispam system has to create these outgoing collaborating messages in order to get similar input from other antispam systems.

2.3 How the System Does Its Job - Internal Architecture and Processing Steps

Internal architecture and processing steps of the antispam system are shown on Figure 2. Each block represents a processing step and/or a memory storage (database). All the shown blocks are per user and are shown for only one user on the figure, except the "Maturation" block which is common for all the users protected by the same antispam system. The following processing tasks are done by the system.

Incoming emails are put into the pending state by the antispam system, until the detection process decides if they are spam or not, or until they are forced to an Inbox by pending timeout, by periodic request from the mail client, or by a request from the user. The innate processing block might declare an email as non-spam and protect it from further processing by the system. If an email is found to be spam, it is quarantined by the antispam system or it is marked as spam and forwarded to the email server for an easy classification. Otherwise it is forwarded to the email server and goes directly to the Inbox. The user has access to the quarantined emails and can force some of them to be forwarded to the Inbox, but is not required to do so.

A pending email that is not protected by the innate part is processed in the following way. First, the text strings of predefined length are sampled from the email text at random positions. Then, each sampled text string is converted into the binary-string representation form called proportional signature ("binary peptide"). The details on creating the proportional signatures are given in Section 2.3. To continue reading this section, you just need to know that similar strings generate similar proportional signatures, i.e. their signatures have small hamming distance, and that unrelated strings with very high probability result in not similar proportional signatures (big hamming distance). This explains why the term proportional signature is used.

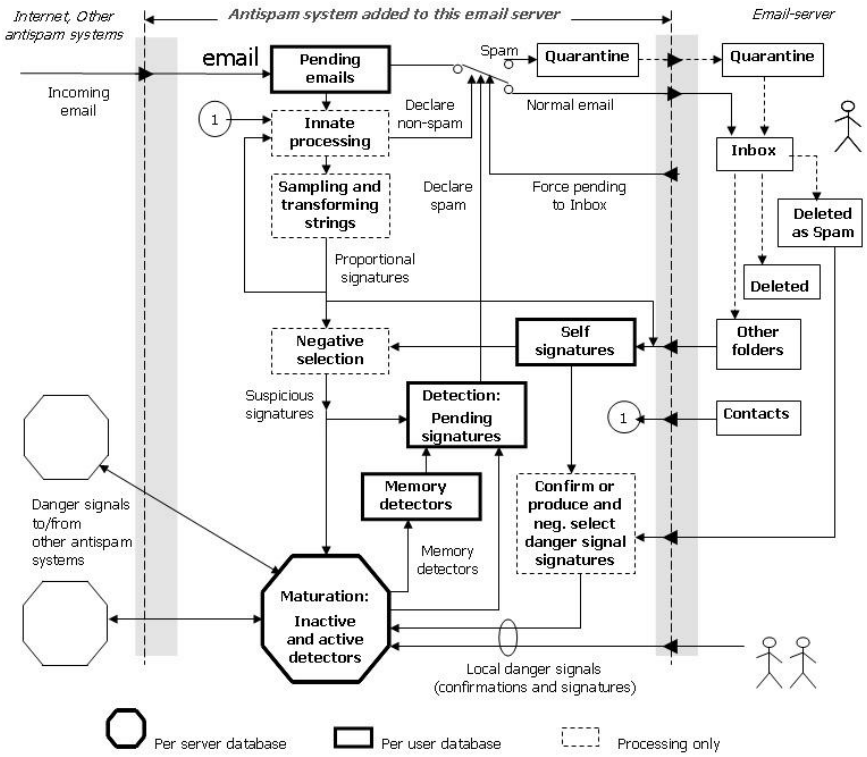


Fig. 2. Internal architecture of the antispam system.

Each proportional signature is passed to the negative selection block. Another input to the negative selection block are so called self signatures, the signatures obtained in the same way as the proportional signatures of the considered incoming email, but with the important difference that they are sampled from the e-mails that the user implicitly declared as non-spam (e.g. outgoing emails). In the negative selection block, the proportional signatures of the considered incoming email that are within a predefined negative-selection-specific similarity threshold of any self signature are deleted, and those that survive become so called suspicious signatures.

Each suspicious signature is duplicated. One copy of it is passed to the maturation block, and another to the detection block. Each suspicious signature passed to the detection block is stored there as a pending signature. It is compared against already existing memory and active detectors and against the new active and memory detectors potentially made during the email pending time. If a suspicious signature is matched (found to be within a predefined detection-specific similarity threshold) by an active or memory detector, the corresponding email is declared as spam. The pending signatures are kept only as long as their corresponding email is pending.

The active detectors used in the detection process are produced by the maturation (block) process. The inputs to this process are the above mentioned suspicious signatures, local danger signatures and remote danger signatures. The local danger signal signatures are created in the same way like the suspicious signatures, but from

the emails being deleted as spam by the users protected by the antispam system. The remote signatures are obtained from collaborating antispam systems.

Except upon start of the system, when it is empty, the maturation block contains so called inactive and active detectors. When a new suspicious signature is passed to the maturation block, it is compared using a first maturation-similarity threshold against the signatures of the existing inactive detectors in the maturation block. Syntax of a detector is shown on the Fig 3. If the signature is not matching any of the existing inactive detectors signatures, it is added as new inactive detector to the maturation block. If it is matching an existing inactive detector, the status of that detector (the first that matched) is updated, by incrementing its counter C1, refreshing its time field value T1, and adding the id of that user. The same happens when a local danger signature is passed to the maturation block, the only difference is that, if matching, C2 and T2 are affected instead of C1 and T1 and DS bit is set to 1. Upon refreshing, the T2 is typically set to a much later expiration time then it is the case with T1. The same happens when a remote danger signature is received from a collaborating system, with a difference that id and DS fields (see next paragraph for the explanation of different fields) are not added and the affected fields are only C3, C4, T3, T4. Local suspicious and danger signatures are passed to the maturation block accompanied by id value, and remote danger signatures do not have the id value but have its own C3 and C4 fields set to real number values (could be binary too), so the local C3 and C4 counters may be incremented by one or by values dependant on these remote incoming signature counters.

signature	ACT	C1	C2	T1	T2	C3	C4	T3	T4	id1	id2	...	idn
										DS	DS		DS

Fig. 3. Syntax of a detector. ACT stands for activated/non-activated and this bit shows the state of the detector. C1 is counter of clustered local suspicious signatures. C2 is counter of clustered local danger signal signatures, i.e. signatures generated from emails deleted as spam by users and negatively selected against user specific self signatures. Ti is time field for validity date of counter Ci. “id” is a local (server wide) identification of the protected user account that received email from which the signature originates, and is useful when deciding how and which users this signature might impact once it becomes activated (explained later). DS is so called danger signal bit of a local clustered signature. It is set to 1 if its corresponding signature comes from an email deleted as spam, else it is set to 0.

Whenever an inactive detector is updated, a function that takes as input the counters of this detector is called that decide about a possible activation of the detector (in the current simple implementation we use a threshold for each counter independently). If the detector is activated, it is used for checking the pending signatures of all the local users’ detection blocks (1 per user). We call this recurrent detection of pending email messages. Optionally, only the detection blocks could be checked for which id is added to the detector.

Upon the activation of a detector, its signature is copied to the memory detectors databases of those users that had their id added to the detector and appropriate DS bit set to 1. Memory detectors are also assigned a life time, and this time is longer then for the activated detectors.

Whenever a new detector is added or an existing is updated by the local suspicious or danger signature, a function is called that takes as inputs C1 and C2 and decides if a signature should be sent to a collaborating system (in a simple implementation the counters may trigger the actions independently).

Both the inactive and active detectors live until all the lifetimes (T1-T4) are expired. The old proportional signatures and detectors in different blocks are eventually deleted, either because of expired life time or need to make space for those newly created.

Transforming the strings into the proportional signatures

There are several reasons and goals to transform the sampled text strings into binary representation. First, in order to preserve privacy, it is important to hide the original text when exchanging the information among the antispam systems. To achieve this we use one way hash functions when transforming text string into its binary equivalent. Second, it is important that the similarity of the strings, as it would be perceived by the reader, is kept as similarity of the corresponding binary patterns that is easy to compute and statistically confident. Similarity might mean small hamming distance, for example. “Statistically confident” means that the samples from unrelated emails should with very high chance have the similarity smaller than a given threshold, while the corresponding samples from the different obfuscations of the same spam email, or from similar spam emails, should with high chance have the similarity above the threshold. “Corresponding” means that they cover similar spammy patterns (expressions or phrases) that exist in the both emails. Third, the binary representation should be efficient, i.e. it should compress the information contained in the text string and keep only what is relevant for comparing the similarity. Last, but not least important, the binary representation should provide possibility to generate the random detectors that are difficult to be anticipated and tricked by the spammers, even if the source code of the system is known to the spammers.

To achieve the above listed goals, we design the representation based on so called similarity hashing. Our custom hashing is similar to the Nilsimsa [5], with important differences. It is illustrated on the Fig 4. The input is a string of the fixed length (sampled at a random position from the email). The sliding window is applied through the text of the string. The window is moved character by character. For each position of the window 8 different trigrams are identified. A trigram consists of three characters taken from the predefined window positons. Only the trigrams containing the characters in the original order from the 5-character window and not spaced more then by one character are selected. Then a parametric hash function is applied that transforms each trigram into the integer from 1 to M, where M is the size of the binary representation that must be the same for all the collaborating systems. The bit within the binary string ”proportional signature” indexed by the computed integer is set to 1. The procedure is repeated for all window positions and all trigrams.

Unlike the Nilsimsa method that accumulates the results within the bins of the proportional signature, and then applies a threshold to set most populated beans to 1 and other beans to 0, we just do overwrite a bit if it is already set, i.e. we fill the proportional signature as if we would fill a Bloom filter. In the used transformation,

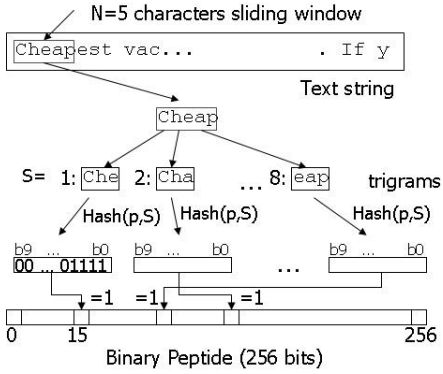


Fig. 4. Hashing of a sampled text string into the proportional signature (also called binary peptide). Collaborating systems must use the same signature length (typically 256, 512 or 1024 bits).

M is determined as the smallest value that provides desirable small contention in the Bloom structure. It is important to notice that the hash function could be any mapping from the trigrams on the $1 - M$ interval, preferably with a uniform distribution of values for randomly generated text. The parameter *p* on the figure controls the mapping. Preferably, the hash function produce the same values for the trigrams containing the same set of characters, in order to achieve robustness against obfuscations that reorder letters within words.

Use of the Bloom filter rule for setting of the signature bits prevents from deleting (by text additions) the bits that correspond to the spammy patterns. Contrary, with a method like Nilsimsa it is possible to add text that will overweight the spammy phrase trigrams and prevent them of being shown up in the signature.

3 Evaluation

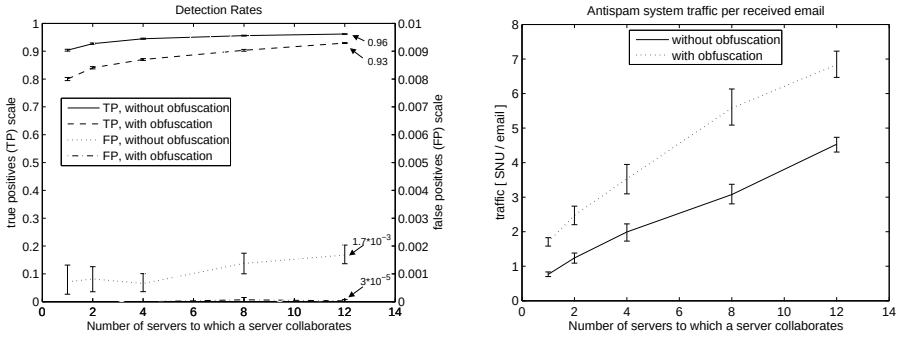
Evaluation of the system is done using a custom simulator made in C programming language. It simulates configurable number of servers and email users per server, and implements the main functions of the proposed antispam system (some details are missing, like detector timers for example, wich were not important for the performed short simulations). User's behavior of sending and reading emails is simulated using a random (uniform) distribution of time between reading new emails and sending random number of emails. The recipients are chosen at random. Network delay is also a parameter (though its impact is small).

We tested how number of systems to which a system collaborates impacts the detection results. We did it for two cases: without obfuscation, when spammer sends many identical copies of the same spammy message, and with obfuscation, when spammer changes each copy from the spam bulk. We tested the system only for one obfuscation model in which letters inside words are rearranged completely randomly for each new copy (such text is still readable by humans). Spammer sends spams in bulks, as that is standard spamming-business model.

The length of sampled strings is 64, the length of binary peptides and detectors is 256. We used spam and ham (not easy or hard ham) sets from SpamAssassin

Corpus [10] of emails for the evaluation. The length of simulation we used was 2h, as constrained with the number of messagess from the used corpus.

4 Results Discussion



(a) (b)

Fig. 5. (a) Detection results; (b) Traffic exchanged for collaboration. SNU stands for Standard Nilsimsa Unit = 256 bits, i.e. the traffic is measured relatively to the amount of traffic produced by a Nilsimsa query to centralized database upon receiving an email.

From the Figure 5(a) we can see that collaboration to up to 10 other antispam systems already gives good results, and that the systems copes well against the tested obfuscation. We were surprised that False Positives is bigger with non-obfuscated messages, but we found that this detections happen with detectors that correspond to header fields of emails. This can be explained with the fact that we did short simulations and during that time self examples are still not learned well as number of good email examples we start with is limited. Obfuscation of messages lessen this artifact. We expect that this artifact will go away in longer simulations and with larger initial number of emails in Inboxes.

From the Figure 5(b) we can see that the traffic created by an antispam system upon receiving an email is only few times larger then for making one nilsimsa database query, which is very moderate usage of the traffic. Upon inspecting number of created candidate detectores and number of exchanged detectors, we found that less than 10% is is exchanged from those created. This is due to the control done by negative selection and maturation processes that put away the detectors that correspond to the parts of the emails that are likely to be normal text (or obfuscation component that is usually random) and allow the collaborating systems to concentrate on further processing of the suspicious patterns.

5 Conclusions

The initial evaluation shows that the system achieves promising detection results under modest collaboration, and that it is rather resistant to the tested obfuscation.

The use of the artificial immune system approach to represent observed and processed information (suspicious signatures) and a learned state (signatures that became active or memory detectors) enables efficient information exchange among

collaborating systems. Only the relevant (locally processed) and independent small units of information (the proportional signatures) need to be exchanged to achieve a distributed and collaborative detection of email spam. These units are independent and contain a **summarized partial information** observed and processed by a local system. With the use of classical neural networks as a competitive approach, the locally learnt information specific to one input pattern is distributed among many links of the local neural network and only the classification output from the local network is available - which does not allow for the simple exchange of summarized information learned locally from multiple strongly correlated observed input patterns.

Due to the complexity of the system and many parameters that might affect the results, the proven conclusions about what the key things are that explain whether and why the system is really working well or not would require a lot of additional testing (simulations) and the use of factorial analysis methods.

Another important point that remains to be evaluated experimentally is the **dynamic** of the response to a new spam bulk by the network of antispam systems. According to the design, we know that during a response to a new spam bulk more resources are used, and that upon creating enough detectors in response to this spam bulk and distributing them within the antispam network, all (or huge majority) of the collaborating users are protected from the remaining spams from that bulk and from repeated similar spams (note that this is very similar to the inflammation and win over a virus by the human immune system). So, it is important to determine the resources needed and the ability and limits of the system to cope with spam under "stress" spamming conditions, when maybe the goal of the attacker is not only to get spam through into the Inboxes, but also to defeat the antispam system(s) (and its reputation) by putting it out of its normal working mode. A better understanding and control of the mechanisms that start and stop the "inflamation" (reaction to spam bulks) in the network of antispam systems is thus crucial.

References

1. Cotten W (2001), Preventing delivery of unwanted bulk e-mail, US patent 6,330,590.
2. Damiani E, et al (2004), An open digest-based technique for spam detection. In Proc. of the 2004 International Workshop on Security in Parallel and Distributed Systems, San Francisco, CA USA.
3. DCC project web page (Jan 2007), http://www.rhyolite.com/anti-spam/dcc/
4. Graham P (2002), A plan for spam, http://www.paulgraham.com/spam.html
5. Nilsimsa project web page (Sep 2006), http://lexx.shinn.net/cmeclax/nilsimsa.html
6. Oda T, White T (2003), Developing an immunity to spam. In: Genetic and Evolutionary Computation Conference, Chicago(GECCO 2003), Proceedings, Part I. Volume 2723 of Lecture Notes in Computer Science, 231-241.
7. Razor project web page (Sep 2006), http://razor.sourceforge.net/
8. Secker A, Freitas A, Timmis J (2003), AISEC: An Artificial Immune System for Email Classification. In Proceedings of the Congress on Evolutionary Computation, Canberra, IEEE, 131-139.
9. SpamAssassin project web page (Sep 2006), http://spamassassin.apache.org/
10. SpamAssassin email corpus (Sep 2006), http://spamassassin.apache.org/publiccorpus/
11. Zhou F, et al. (2003), Approximate object location and spam filtering on peer-to-peer systems. In Proc ACM/IFIP/Usenix Int'l Middleware Conf., LNCS 2672, pp. 1-20.