A Multi-Agent Approach to Testing Anti-Spam Software

Nathan Friess
Lyryx Learning
Calgary, Alberta T2N 3P9
Email: nathan@lyryx.com

Heather Crawford
Department of Computer Science
University of Calgary
Calgary, Alberta T2N 1N4
Email: crawfoha@ucalgary.ca

John Aycock
Department of Computer Science
University of Calgary
Calgary, Alberta T2N 1N4
Email: aycock@ucalgary.ca

***Abstract*—SpamTestSim is a multi-agent based simulator in which agents represent spammers and legitimate email users. The purpose of SpamTestSim is to provide a realistic environment for testing anti-spam software. The agents send and receive mail (both ham and spam) and create and maintain contact lists. We demonstrate the simulation by testing four well-known spam filters.**

I. Introduction

There are a wide range of approaches to filtering spam [1], such as statistical content filters that determine how close a new message is to known spam using words in the message [2], or reputation filters that track past behavior of mail servers and use that to predict future behavior [3]. New approaches to filtering spam are continually being developed, and the methods used to evaluate these new ideas and analyze their effectiveness need to keep pace.

Experimental methods that have been used in spam filter evaluation can be prone to influence by too many variables, as is the case in gathering results from real deployments. Experiments can also be too simplistic by not providing enough breadth in the types of email messages and their origin. It is important that evaluations strike a balance between realism and controlled experimentation. To this end, a simulator should preserve the behavior patterns of real email; thus a more appropriate approach would be to simulate a network of email senders, focusing on sending behavior (who, when, and how much) rather than the content of messages.

This paper introduces a new spam filter evaluation framework called SpamTestSim to fill this gap. SpamTestSim is designed to evaluate spam filter accuracy and performance by using a multi-agent paradigm to simulate the behavior of individual email users, spammers, and other automated email systems. The simulation focuses on emulating behaviors exhibited by humans with the goal of exposing weaknesses in spam filters that result in misclassifying legitimate email as spam.

The rest of this paper is organized as follows: Section II outlines the previous work that our approach is based upon; Section III discusses the SpamTestSim framework in detail, and finally Sections IV and V outline some results and conclusions.

II. Related Work

An extensive evaluation of spam filters was performed by Cormack and Lynam [4]. Cormack and Lynam were interested in performing repeatable experiments with a common baseline to compare both existing filters and potential new filters. Their method of accomplishing this goal was to compile realistic corpora of both ham and spam email and mimic the real operating environment of the tested filters as closely as possible. The experiments performed by Cormack and Lynam on spam filters used their published evaluation framework [5].

Sarafijanovic et al. [6] have taken the concepts from Cormack and Lynam's framework and moved them into a distributed implementation, overcoming one of the limitations of Cormack and Lynam's framework: the lack of support for distributed filtering systems. Their simulated users were simply an extension of Cormack and Lynam's feedback system, and were used to provide training data to filters.

Garcia et al. [7] performed a wider analysis of spam filters similar to Cormack and Lynam's work, but created a framework to test filters in a somewhat realistic environment. The important difference here is that rather than gathering both spam and ham messages in a corpus and using that to test filters directly, these authors simulate the behavior of both legitimate senders and spammers, and use the ham/spam content from a variety of sources.

The simulation performed by Garcia et al. provides a groundwork for the work described in this paper. Our work differs in that it creates an actual social network to realistically simulate an environment where both spam and ham are prevalent. It is in this realism that the real benefit – as well as the most significant differences between our work and the work cited above – may be best seen. Furthermore, our work uses a multi-agent system approach to simulate the various actors in a standard email environment in order to further enhance the realism of the spam filter testing. Thus, with the addition of realism and a novel approach to creating it, our work extends the work of previous spam research.

III. Implementation

SpamTestSim is a simulator that is used to evaluate the efficacy of spam filters. It is designed as a multi-agent system, where agents simulate the behavior of legitimate email users,

spammers, or automated mailing systems such as mailing lists. The implementation for SpamTestSim is a Python program that runs on one or more networked computers. Theoretically, SpamTestSim can scale to include tens of thousands of agents, although simulations have been limited to no more than one thousand agents in order to prevent bottlenecks caused by the spam filters themselves. Each agent follows hard-coded algorithms with randomization in areas such as message selection and speed of an agent's reply to a message. Agents communicate with each other using the Simple Mail Transfer Protocol (SMTP) [8] and Post Office Protocol Version 3 (POP3) [9] protocols, and RFC 2822 message format [10] in order to conform to the standard message format expected by a typical spam filter.

Randomization plays a key role in agents' behavior in order to avoid predictable sequences of actions. To this end, all agents in SpamTestSim use Python's *random* module, which, like many other random number generators, is not truly random. SpamTestSim seeds the "main" pseudo-random number generator with the current system time and then provides each agent with its own pseudo-random number generator object that is seeded using the "main" generator. This allows agents to generate pseudo-random numbers independently of each other, which is necessary to maintain independent states for different numerical distributions of the random numbers, such as normal and gamma distributions. Since each agent has its own generators for these distributions, each agent will not affect the distribution of other agent's generators. Furthermore, it is possible to recreate a previous simulation by re-using the seed of the main pseudo-random number generator in order to allow the agents to follow very similar – but not exactly the same – behavior. Unfortunately, agents' behavior cannot be precisely replicated, because external factors such as the operating system's CPU scheduler and the spam filter's latency in providing verdicts cannot be (easily) controlled. However, recreating small simulations can be useful for verification or debugging purposes.

A. Simulation Phases

Simulation in SpamTestSim consists of two phases: the initialization phase and the simulation phase. In the initialization phase, the simulation controller creates all required agents and sets the parameters that govern the agents' behavior. It is during this phase that a social network is generated (see Section III-D for more details), thus creating the contact lists for each agent. User-configurable parameters are loaded from a configuration file and applied to each agent, and any randomly generated parameters are initialized. Agents are then assigned to simulator hosts on the network.

After this initialization is completed, the simulation phase begins, where all agents are given a signal to indicate that they may begin running. During this time, each agent runs as a separate thread or process, and follows the algorithm that has been defined for the agents' class. This implies that the agents' behavior is not synchronized; they make decisions and act independently of each other during the simulation, which adds to the realism. Each agent continues to run until the simulation controller signals the simulation's end.

B. Underlying Infrastructure

SpamTestSim's infrastructure has two components: the simulation controller and simulation hosts. Simulation hosts consist of a remote procedure call [11] server that accepts requests from the simulation controller, which allows the simulation code to be independent from the underlying structure of the simulation controller and hosts. Once the simulation controller has initialized the agents, it uploads the agents to the simulation hosts, where the agents are run when the simulation starts.

The simulation controller oversees the entire simulation from start to finish, and is the program that the user invokes to run the simulation. The user controls the simulation through a configuration file, although the simulation cannot be influenced (steered) by the user once it has begun. The simulation controller creates all instances of agent classes and initializes agent parameters, as well as creating the global social network. Once this is complete, the controller's main task is to accept requests from agents and assist in coordinating the simulation, as well as ending the simulation at the time defined in the configuration file.

Figure 1 shows the high-level structure of the simulation. The simulation controller controls n agents (recall that these agents may be located on multiple different physical computers); the agents interact with a common POP3 and SMTP server, and the SMTP server communicates with the spam filter being tested. Mail is passed from the SMTP server to the POP3 server through a common mail store.

C. Agents

Agents in SpamTestSim are implemented as a hierarchy of classes. All agents inherit basic functionality from a BaseAgent class; for instance, the ability to create an email message. There are currently three subclasses of BaseAgent: HamAgent, SpamAgent, and MailingListAgent. During the initialization phase of the simulation, one of these three classes will be instantiated for each simulated user. This structure allows for the easy creation of new agent classes. The following sections describe each of the existing agent classes in more detail.

1) BaseAgent: BaseAgent provides common functionality for the implemented agents. One of the most useful features of BaseAgent is the flexible configuration management that is automatically used by all subclasses of BaseAgent. Via the configuration file, it is possible to change various parameters for existing agents, such as the location of the spam or ham corpuses used, or how often a particular agent will check for new messages and respond to other agents. The parameters in the configuration file can be set by the user in order to simulate various email environments. When the class is instantiated at run time, the BaseAgent class automatically reads all applicable parameters from the configuration file and builds the environment accordingly.

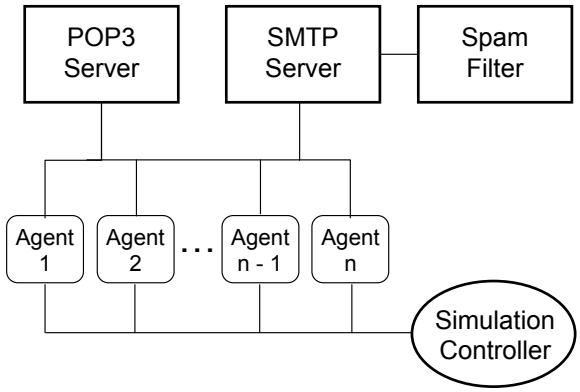

Fig. 1: Simulation structure.

Another important feature of BaseAgent is automatic email message formatting, which is made available to all subclasses. This is especially useful for agents simulating humans that use mail clients, such as HamAgent. BaseAgent provides a variety of simulated mail clients for such agents to use. For example, one agent may use a particular version of Mozilla Thunderbird, while another may use a version of Microsoft Outlook. Both of these agents can build the content of messages that they will send (recipient list, subject, and body), and let the common message formatting functions handle the exact layout of the email message (User-Agent header, reply quoting style, etc.) based on which mail client the current agent is using. Currently, each agent only uses one simulated mail client for the entire simulation. Using the BaseAgent class as a starting point, the other various types of agents such as SpamAgents and HamAgents may be produced in order to simulate the desired environment.

2) SpamAgent: SpamAgents simulate humans that directly send spam rather than human-controlled botnets that send spam. This distinction is important because a single spammer who sends messages serially behaves differently than a botnet that sends messages in parallel; some spam filtering techinques are more effective for one of these situations over the other. However, since the focus of this work is on the HamAgent behavior, the SpamAgent implementation only covers the simple case of a human that sends spam messages directly.

The algorithm used by SpamAgents is as follows: a SpamAgent starts by sleeping for some period of time, then chooses a random message that has not been used previously from the spam corpus and sends a copy of the message to a randomly chosen list of HamAgents. The spam message is sent to one recipient at a time, with a small random delay introduced between sending to each recipient. Once the SpamAgent has attempted to send the spam to all of the intended recipients, it sleeps for a randomly chosen period of time, and the entire process repeats. Note that SpamAgents are not part of the HamAgent social network. Instead, SpamAgents have knowledge of all HamAgents, and can send spam to any of them. The number of HamAgents to which spam is sent is defined as a percentage of the number of HamAgents. In order to add to the realistic nature of SpamTestSim, agents have also been created that simulate legitimate mailing list software, although they are not spammers in the sense of the SpamAgents.

3) MailingListAgent: MailingListAgents behave like typical (legitimate) mailing list software. That is, they accept messages from other agents (HamAgents or SpamAgents), and copy the message to all of its subscribers. The subscribers are chosen during the initialization of the agent, and are fixed throughout the simulation. The subscriber list is a randomly chosen subset of all of the HamAgents in the simulation, with the size of the subscriber list being randomly chosen with a normal distribution. Similar to the SpamAgent, MailingListAgent limits the speed at which messages are delivered to each recipient by inserting a delay in between the delivery to each recipient. Note again that MailingListAgents are not considered spammers *per se*; they are merely an implementation of a tool that real-life spammers use in order to send spam quickly and (hopefully) anonymously.

4) HamAgent: The core agent in SpamTestSim is HamAgent, and so HamAgents have the most complex behavior and the largest number of user-controllable parameters. The basic algorithm of a HamAgent is to download messages from its own POP3 mailbox at mostly regular intervals, determine whether the incoming message is spam, and decide whether to respond, forward, or ignore a message. HamAgents also create new messages at randomly chosen regular intervals and sends them to one or more contacts. In order to determine whether a given message is spam or ham, the HamAgent checks the envelope of the message against the simulator's list of all agents - if the agent is a spammer, the message is considered spam; otherwise the message is considered ham. It is important to note that this means that the simulator must have an accurate view of which agents are spammers and which are not, and also that SpamAgents do not forge SMTP envelope sender addresses (outside of the simulation, both of these assumptions do not hold because spammers routinely forge sender addresses, otherwise it would be relatively trivial to filter spam based on a blacklist of known spam senders). Once a HamAgent determines that an incoming message is spam, it will always ignore the message. This assumes the simplest case that humans are not interested in email messages that they do not recognize (spam), and will always delete them. If a HamAgent receives a message that it determines is not spam, it picks a uniformly distributed pseudo-random number and will reply to or forward that message based on two parameters that set the probability of each action, as defined in the configuration file.

The contact list of each HamAgent can evolve throughout the simulation. When a HamAgent receives a message that is addressed to multiple recipients, it will randomly add recipients to its contact list if they are not already present, and will remove contacts from its list if it has not received or sent a message to that contact within a user-defined amount of time. The body of a message sent by a HamAgent is chosen randomly from a ham corpus, and can be sent as a new

message, a reply, or a forward. If the message being sent is a reply or forward, the MailClient class assigned to the current HamAgent will format the reply or forward as appropriate for the simulated mail client, and insert the new text in the correct location. Currently, all simulated mail clients either insert the new text above or below the quoted text, but never interleave the new text. This means that HamAgents never trim text out of the quoted text of a forward or reply; humans are known to do this at times.

In order to simulate the behavior of humans, HamAgents have the ability to switch between "working" mode and "sleep" mode. Working mode simulates humans that are awake and at work, so HamAgents tend to send and receive more messages while in this mode. In sleep mode, HamAgents do not download their received messages, nor will they send any new messages. The behavior of moving between working and sleep mode is guided by two factors. First, SpamTestSim has a clock that runs at a user-defined rate, which can be equal to or faster than wall clock time depending on what is appropriate for the filter being evaluated. As part of the creation of the social network of HamAgents, every HamAgent is situated in a time zone and thus every HamAgent has a notion of time that it can rely on. The second factor in the work/sleep cycle is at what times the agent chooses to sleep or wake up, which can be set to any time, to the nearest minute.

These features combined allow HamAgents to perform a basic simulation of how humans use email on a daily basis. HamAgents can perform the basic tasks of sending, receiving and replying to messages, as well as forwarding messages to contacts. Contact lists are built from a larger social network, but can also evolve as agents learn about each other or lose contact over time. The contents of messages are copied verbatim from a ham corpus, but the formatting follows one of the available mail client formats such as Thunderbird or Gmail. Although much of this behavior is coded in an algorithm, the code makes judicious use of random number generators and user-defined parameters to allow agents to behave in a less deterministic manner.

D. Generating the Social Network

Before the agents can communicate with each other, they gather the email addresses of other agents from a contact list. If we combine all of the contact lists for all of the agents, we will have the social network of all agents in the simulation. The social network can be shown as a graph where each node represents an agent and nodes are connected by an edge when one agent has the other in its contact list. The graph can be either a directed or undirected graph, but this choice does not greatly affect the generality of the social network since when one HamAgent sends an email to another, the recipient now knows about the sender and could add the sender to its own contact list, thus forming an undirected edge. SpamAgents and MailingListAgents are not considered to be part of the generated social network, and choose their contacts by selecting a random subset of all HamAgents.

The basic Barabási-Albert algorithm [12] was chosen for this work due to its simplicity and broad understanding in the research community. By extending the Barabási-Albert scale-free network generation algorithm we built social networks that not only mimic real-world social networks in terms of structure, but also in terms of geography (time zones). We created two extended scale-free network generation algorithms: a top-down and a bottom-up algorithm. The top-down algorithm mimics a large organization that crosses time zone boundaries by building one large scale-free social network and partitioning it into time zones. The bottom-up algorithm mimics small independent organizations by building several smaller social networks and connecting them in a way that maintains the scale-free structure in the larger network. Full details can be found in [13].

IV. Results

The experiments for the work described in this paper use an example run of SpamTestSim to evaluate four well-known spam filters: BogoFilter, DSPAM, CRM114, and SpamAssassin. These experiments are *not* intended to provide a comparison of the accuracy of the spam filters tested since a proper comparison of different spam filters would involve a careful analysis of the ham and spam corpuses used to ensure that there is no contamination in either corpus. Instead, the intent of these experiments is to demonstrate SpamTestSim, and to compare SpamTestSim against itself with different parameters. Therefore, the results presented here should be viewed as relative comparisons instead of a search for the "best" spam filter.

A. SpamTestSim: Base Configuration

For each of the simulation runs with SpamTestSim in the base configuration, a social network of 100 HamAgents was generated, and two SpamAgents were used to send spam. The ham database used by the HamAgents was drawn from Usenet reply messages using a hand-selected newsgroup list [14]. SpamAgents chose random messages from the TREC 2006 Spam Track Public Corpus [15]. Each simulation was run for a total of 30 minutes, after which the simulation automatically quit and the number of detected spam, ham, false positive and false negative messages were tallied. A total of five separate simulations were performed with each of the three configurations and all results shown are an average of the five runs.

The full results from running SpamTestSim against each of the four spam filters using the base configuration are shown in Table I. For a 30 minute simulation, SpamTestSim classified only a small amount of spam because only two SpamAgents were instantiated. Increasing this number overwhelmed the spam filters and caused the mail queue of the SMTP server to continually grow throughout the simulation, although using more powerful computing hardware in a load balancing setup would overcome this limitation. However, while the amount of spam classified was artificially low, the results of classified

TABLE I: Base results against spam filters.

		Spam	Ham
BogoFilter	Correct	1086	5549
	Incorrect	1903 (63.7%)	485 (8.0%)
CRM114	Correct	641	5966
	Incorrect	2333 (78.4%)	0 (0.0%)
DSPAM	Correct	2355	5997
	Incorrect	655 (21.8%)	45 (0.7%)
SpamAssassin	Correct	2681	5979
	Incorrect	158 (5.6%)	18 (0.3%)

TABLE II: Results of running SpamTestSim with more replies.

		Spam	Ham
BogoFilter	Correct	1248	16389
	Incorrect	1451 (53.8%)	484 (2.9%)
CRM114	Correct	365	16761
	Incorrect	2463 (87.1%)	0 (0.0%)
DSPAM	Correct	2371	15923
	Incorrect	553 (18.9%)	882 (5.2%)
SpamAssassin	Correct	2226	9606
	Incorrect	334 (13.0%)	22 (0.2%)

ham still show interesting variations with different configuration settings, as later experimental results will demonstrate.

Note that BogoFilter's false positive rate was the worst of all four filters at 8.0%, and also was poor at detecting spam. CRM114 was overly conservative about what it classified as spam with its spam detection rate at only 78%. DSPAM's false positive rate was 0.7% and detected only 2355 spam messages. SpamAssassin's results were adequate with a 0.3% false positive rate and 2681 spam messages detected.

B. SpamTestSim: Replying More Often

In this experiment, HamAgents are configured to reply to each other more often by increasing the probability that a HamAgent will reply to a received message. The results of this experiment, shown in Table II show a significant change in false positive rates for BogoFilter and DSPAM. BogoFilter's false positive rate improved by 5.1% from the base SpamTestSim configuration. Based on this result, we hypothesize that BogoFilter can better identify messages that are replies to other messages, possibly by looking for identifiers such as "Re:" in the subject line or the "> " marks in front of lines containing quoted text. DSPAM, on the other hand, performed 4.5% worse at identifying ham, which implies that the same identifiers have caused DSPAM to misclassify more email. CRM114 and SpamAssassin performed about the same as before. Looking at the SpamAssassin table, we can see that fewer ham messages were classified as spam than the other filters in this experiment due to the fact that SpamAssassin was unable to keep up with the flow of messages from the agents in the simulation. The message queue backed up significantly about 10 minutes into the simulation, and continued to grow until the simulation was complete, which led to a delay in message delivery of about five minutes on average, providing fewer opportunities for agents to reply to each other. In this experiment, SpamAssassin is clearly the slowest filter.

V. Conclusion

SpamTestSim was used to evaluate four spam filters: BogoFilter, CRM114, DSPAM, and SpamAssassin. Each of the spam filters tested is known to rely heavily on the contents of classified messages rather than looking at the behavior of the users (agents) themselves. It would be interesting to perform more comparisons of spam filters, such as those that use sender reputation or related distributed filtering techniques, although it is left for future work.

This work is a first step towards a new approach of using a multi-agent system to evaluate spam filters and highlight issues with false positives. As it stands, SpamTestSim already shows promising results and has laid the groundwork for many possibilities of further research into the evaluation of spam filters. The full source code for SpamTestSim will be made publicly available for other researchers to use and extend.

Acknowledgment

Some of the authors have their research supported in part by grants and scholarships from the Natural Sciences and Engineering Research Council of Canada (N.F., J.A.) and the Informatics Circle of Research Excellence (N.F.). The first author's work was done while at the University of Calgary.

References

[1] Y. Guo, Y. Zhang, J. Liu, and C. Wang, "Research on the Comprehensive Anti-Spam Filter," in *IEEE INDIN 2006*, 2006, pp. 1069–1074.
[2] P. Graham, "A Plan for Spam," http://www.paulgraham.com/spam.html, Aug. 2002, accessed September 17, 2008.
[3] V. V. Prakash and A. O'Donnell, "Fighting Spam with Reputation Systems," *ACM Queue*, vol. 3, no. 9, pp. 36–41, Nov. 2005.
[4] G. V. Cormack and T. R. Lynam, "Online Supervised Spam Filter Evaluation," *ACM Transactions on Information Systems*, vol. 25, no. 3, p. article 11, Jul. 2007.
[5] T. R. Lynam and G. V. Cormack, "TREC 2006 Spam Evaluation Kit," http://plg.uwaterloo.ca/~gvcormac/jig/, 2006, accessed December 19, 2007.
[6] S. Sarafijanovic, L. Hernandez, R. Naefen, and J. Le Boudec, "Anti-spamLab - A Tool for Realistic Evaluation of Email Spam Filters," in *Fourth Conference on Email and Anti-Spam (CEAS)*, 2007.
[7] F. D. Garcia, J. Hoepman, and J. van Nieuwenhuizen, "Spam Filter Analysis," in *19th IFIP International Information Security Conference*, 2004, pp. 395–410.
[8] J. Klensin, "Simple Mail Transfer Protocol (RFC 2821)," http://www.ietf.org/rfc/rfc2821.txt, Apr. 2001.
[9] J. Myers and M. Rose, "Post Office Protocol - Version 3 (RFC 1939)," http://www.ietf.org/rfc/rfc1939.txt, May 1996.
[10] P. Resnick, "Internet Message Format (RFC 2822)," http://www.ietf.org/rfc/rfc2822.txt, Apr. 2001.
[11] A. S. Tanenbaum, *Computer Networks*, 4th ed. Pearson, 2003.
[12] A.-L. Barabási and R. Albert, "Emergence of Scaling in Random Networks," *Science*, vol. 286, no. 5439, pp. 509–512, 1999.
[13] N. Friess, "SpamTestSim: A Tool for Evaluating Spam Filters," Master's thesis, University of Calgary, 2009.
[14] N. Friess and J. Aycock, "A Kosher Source of Ham," in *MIT Spam Conference 2009*, to appear, 2009.
[15] G. V. Cormack and T. R. Lynam, "TREC 2006 Spam Track Public Corpora," http://plg.uwaterloo.ca/~gvcormac/treccorpus06/, 2006, accessed April 17, 2008.

This work was supported in part by NSERC ISSNet, the Internetworked Systems Security Network.