A Trust Based System for Enhanced Spam Filtering

Jimmy McGibney and Dmitri Botvich
Telecommunications Software & Systems Group, Waterford Institute of Technology, Waterford, Ireland
Email: {jmcgibney, dbotvich}@tssg.org

Abstract—**The effectiveness of current anti-spam systems is limited by the ability of spammers to adapt to filtering techniques and the lack of incentive for mail servers to filter outgoing spam. A new approach, based on decentralised trust management, is described in this paper. An architecture and protocol, called TOPAS (Trust Overlay Protocol for Anti Spam), are presented. Each mail server records trust measures relating to each other mail server of which it is aware. Trust by one mail server in another is influenced by direct experience as well as recommendations issued by collaborating mail servers. The TOPAS protocol specifies how these experiences and recommendations are communicated between each spam filter and its associated trust manager, and between trust managers of different mail servers. A technique for improving mail filtering performance and the TOPAS protocol using these trust measures is also described. Finally, experimental work is presented that illustrates use of the protocol in a simulated network scenario. Initial results illustrate the dynamics of this system, and indicate the potential of this approach to significantly improve rates of false positives and false negatives in anti-spam systems.**

This is an extended version of a paper presented at the Availability, Reliability and Security Conference in April 2007 [1].

Index Terms—**trust management, trust protocols, spam filtering, e-mail security**

I. Introduction

Spam is probably the greatest single nuisance for users of the Internet. As well as being annoying, spam also introduces many serious security risks and is often used to conduct fraud, as a conduit for malicious software and to carry out denial of service attacks on mail servers. Despite significant anti-spam efforts, the development of powerful spam filtering technologies, and even new legislation in many countries, the incidence of spam remains stubbornly high.

The main current anti-spam techniques focus on filtering incoming email, either based on parsing message content, Bayesian filtering, maintaining Domain Name System (DNS) blocklists, and/or the use of collaborative filtering databases. Spammers tend to be resourceful though, and quickly find ways to get around most countermeasures. There are various attempts to also make anti-spam more adaptive, with the availability of reporting services that allow spam filters and blocklists to keep up to date with new spam techniques. There is a feeling though that spammers are always a step ahead, with the anti-spam community following with countermeasures some time afterwards.

In addition to technical countermeasures, several governments have introduced legislation outlawing spam and providing for stiff penalties. Despite some prosecutions, such legislation has had little or no effect, for several reasons such as the inter-jurisdictional nature of email traffic and spam.

Internet email is transferred using the Simple Mail Transfer Protocol (SMTP) [2]. SMTP was first introduced [3] at a time when the total number of Internet hosts was just a few hundred and trust between these could be assumed, and was designed to be lightweight and open. Attempts to introduce authentication via extensions or with new protocols that can sit on top of SMTP (e.g. Pretty Good Privacy, PGP [4]) have proven useful in some situations but are very far from universal adoption.

In this paper, we explore the use of decentralised trust to provide a more robust spam filtering system. A well-known definition of trust is "a particular level of the subjective probability with which an agent will perform a particular action" (Gambetta, [5]). Trust is primarily a social concept and, by the above definition, is personalised by the subject. In this paper, we make use of trust by taking the socially-inspired notion of trust based on experience and recommendation.

The peer-to-peer nature of the Internet mail infrastructure suggests that it should be well suited to a distributed approach to trust management.

This paper proposes an architecture and protocol for establishing and maintaining trust between mail servers. The architecture is effectively a closed loop control system that can be used to adaptively improve spam filtering. In this approach, mail servers dynamically record trust scores for other mail servers; trust by one mail server in another is influenced by direct experience of the server (i.e. based on mail relayed by that server) as well as recommendations issued by collaborating mail servers. As well as modelling trust interactions between mail servers, we explore how mail filtering can combine trust values with existing mail filtering techniques.

The remainder of this paper is organised as follows. The next section (section II) outlines the reasons that spam is prevalent and the main anti-spam techniques. Next (section III), requirements for a suitable trust

management system are discussed. A trust overlay architecture is presented in section IV, followed by the new TOPAS protocol in section V. Section VI describes some algorithms for handling trust measures received with TOPAS. Section VII describes how trust measures can be used to enhance spam filtering. Section VIII presents the results of simulation experiments that demonstrate the use of our approach. Finally, section IX discusses related work and section X concludes the paper.

Note that we focus on mail transfer agents (MTAs) rather than user clients in this paper. We interchangeably use the terms "mail server", "mail domain" and, simply, "node" in place of "MTA" throughout this paper.

This is an extended version of a paper presented at the Availability, Reliability and Security Conference in April 2007 [1]. This extended version provides more detailed background, explicitly elaborates requirements, describes the TOPAS protocol in more detail, and provides additional experimental results and analysis.

II. Anti-spam Background

The Internet mail infrastructure, based on SMTP, has in general been very successful. Much of this success has been due to its simplicity. With SMTP, mail is transferred from a client system to a server system using a limited set of text commands. The focus of SMTP is on effective mail transport across heterogeneous systems, with the assumption that security or other desired features are provided at a higher layer or by means of extensions. The main security effect is in a lack of requirement for authentication of the source of an email or the path that it has taken.

The main anti-spam techniques in practical use are based on message content and DNS blocklists, and/or use of collaborative filtering databases. *SpamAssassin* [6], for example, processes each incoming mail and assigns a score to it based on a form of summation of values attributed to possible spam indicators. The higher the score, the more likely it is that the mail is spam. A threshold is then used to filter mail – a mail that scores below the threshold is accepted and one that scores above the threshold is flagged as spam. The alternative, or complementary, approach is blocklisting – mail coming from unreliable SMTP clients is simply blocked.

These techniques have significant limitations though. With content filtering using a threshold, some genuine mail messages may score above the threshold and be flagged as spam (false positives) and some spam may score below the threshold and be accepted (false negatives). Careful tuning of the threshold can optimise the balance between the incidence of false positives and false negatives. For example, many email users will tolerate a modest level of spam to reduce the risk of some important mails being blocked. In practice, spammers are quite resourceful and adapt to content filtering advances - by, for example, using images rather than text, mutating text to avoid keywords that cause high filter scores, or spoofing source address to make it more acceptable.

DNS blocklists are also problematic. These effectively identify spam sources in a binary fashion (i.e. a mail source is either on the list or it is not). This becomes problematic when "good" mail servers are attacked and exploited, or when they fall somewhere in between – for example, where servers are well managed but client machines are not patched frequently enough and may be hijacked by spammer rootkits. Even with good system administration, this can happen and may not be noticed immediately.

Several innovative new anti-spam techniques have been proposed. An example is the use of micro-payments for sending mail. The idea is that cost is negligible for normal users but punitive for bulk mail [7]; in one variation, the cost is more significant but is refundable if not spam [8]. Another idea is to require the sending client to solve a computational challenge for each message, greatly slowing bulk mail generation. Other techniques include issuing of cryptographic challenges that rely on message history between the parties (to get around address spoofing) [9] and the use of prechallenges [10]. Additional tricks, such as obfuscation of published email addresses and the use of human interactive proofs [11] in email account creation systems, try to frustrate spammers by making it harder to automate their activities.

In summary, though, it is fair to say that spam is still a substantial (and growing) problem for most mail users and administrators despite efforts so far to counter it.

III. Requirements for Trust-based Anti-spam System

Almost any transaction between entities requires the establishment of trust between them. The decentralised nature of many Internet services means that a model of trust is necessary for effective operations. The scope for hierarchical "top-down" solutions is limited due to the lack of centralised control, the desire for privacy and, in some cases, anonymity, and the increased use of services in a one-off ad hoc fashion. The Internet mail infrastructure is peer-to-peer in nature and thus should be well suited to a distributed approach to trust management.

We can identify some specific requirements for designing a trust-based anti-spam system:

- *Compatibility with existing infrastructure*
  The existing email system should not require changing.
- *The meaning of trust*
  Trust is defined as being between two nodes, in this case mail servers – i.e. node *i* has a certain level of trust in node *j*. Each node has access to a measure of level of trust in each other node. Each node should manages its own trust level independently.
- *Trust updates based on experience*
  It is possible for trust to be built up by experience. Although there may initially be little or no trust between node *i* and node *j*, it must be possible to establish trust based on interactions between them.
- *Trust updates based on recommendations*

Node *i*'s level of trust in node *j* may be influenced by node *k*'s level of trust in node *j* (communicated by node *k* to node *i*).

- *Robustness against attack, including collaboration between spammers*
  Spammers tend to adapt to new anti-spam systems. For a new trust-based approach to be effective, it should be difficult for spammers to render it ineffective. This could include the possibility of spammers actively participating in the trust system, possibly in concert, issuing false recommendations.
- *Stability*
  The system should be stable – i.e. consistent treatment of mail, few oscillations in state, and rapid convergence to new state on changes. There should be a way to allow a poorly behaved domain to regain trust following recovery from hijack or following a change in its administration policy.

IV. TRUST OVERLAY ARCHITECTURE

In this section, we propose the overlay of a distributed trust management infrastructure on the mail infrastructure, with a view to using trust information at nodes to assist with spam filtering. Fig. 1 illustrates the relationship between SMTP and this new infrastructure. Mail transport operates as normal – we do not propose any changes or extensions to SMTP or other existing mail protocols. With our proposed trust overlay architecture, a trust management layer operates separately from mail transport. Two message passing interfaces are defined between the mail transport layer and the trust management layer and another between the trust managers of individual nodes. The interfaces are as follows (Fig. 1):

(1) Experience reports: Mail host → Trust manager
(2) Trust recommendation: Trust manager ↔ Trust manager
(3) Policy updates: Trust manager → Mail host

A. Data structures: representing trust between mail servers

The Internet mail infrastructure is made up of a (large) number of SMTP-capable hosts. For the remainder of this discussion, we refer to these mail servers as *nodes*.

Each node records a set of trust-related parameters for each other node. In the general case, there could be many such parameters corresponding to a variety of services for which trust measures are used. In our experiments, trust in another mail server is represented by a score in the range (0,1) – each node then records this score for each other node about which it is aware. A trust score of 0 means that there is no trust in this node and a trust score of 1 means that this node is fully trusted. Some trust scores will be very stable and may result from plenty of experience and corroboration by several peers and thus there may be a high level of confidence in these scores. Other trust scores may be less reliable. Thus we also record a confidence level for each trust score, again in the range (0,1). A third parameter that we use is recency,

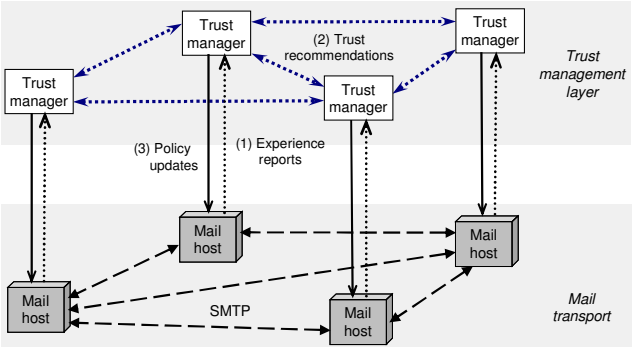

Figure 1. Mail infrastructure trust overlay

indicating how recently this trust score was updated. Recency is important as mail servers come and go; good mail servers become bad (e.g. due to a hijack) and bad mail servers become good (e.g. due to removal of malware). Recency can be recorded in any appropriate units of time.

This can be implemented as a table that might look something like that shown in Table I.

Nodes can control the size of this table by choosing to "forget" a trust score and deleting the corresponding entry. For scalability reasons, trust scores might just be recorded for nodes that have been encountered recently.

B. Mail host identification

Trust scores are recorded per IP address (IPv4 or IPv6). Table I shows hostnames only for reasons of clarity. A trust score thus is just meaningful when applied to mails from the IP address to which it is attached. As SMTP does not protect against modification of sending SMTP server hostname or IP address by intermediate mail relays, processing of trust values is always based on the last hop. Mail received from an SMTP relay affects the trust score of that SMTP relay rather than the originating host. This provides the relay with a strong incentive to filter outgoing spam. The IP address of the last hop is collected from the source IP of packet(s) containing the SMTP HELO/ELHO message (to which a reply will have been sent).

C. Modelling centralised trust references

The architecture described here is highly distributed, with each node autonomously managing its own view of the trustworthiness of other nodes. In practice, however, some nodes might prefer to defer to a centralised trust reference. Such a centralised trust reference can be modelled as a virtual node with a very high trust value (perhaps 1.0).

V. TOPAS PROTOCOL

This section describes a protocol that we call TOPAS (Trust Overlay Protocol for Anti Spam). The TOPAS protocol is for collecting spam statistics to calculate trust, sharing this trust information between nodes and feeding it back to mail hosts to allow them to more effectively filter spam.

The protocol executes using asynchronous message passing in the case of interfaces (1) and (2). Interface (3) uses a simple synchronous request-reply. Underlying services are assumed, including message delivery, failure detection and timeout. It is assumed that each process receives queued messages of the form (`tag`, Arg_1, …, Arg_n). This outline style of protocol specification is influenced by the Generic Aggregation Protocol (GAP, [12]).

(1) Experience report: Mail host → Trust manager

The mail host has an associated spam filter. Each item of mail that arrives at the server is processed by the spam filter, with the result that it is either accepted or flagged as spam. This information needs to be available to the trust manager. The following two messages, from the mail host to the trust manager, provide this:

- (`singleMail`, *i, s*) may be sent from the local mail host to the trust manager to report on a single mail from node *i*. The value of *s* is 1 if the mail has been determined by the mail host to be spam and 0 otherwise. *i* identifies the sending node.
- (`bulkMail`, *i*, *n*, *s*) may be sent from the local mail host to the trust manager to report on a sequence of mails from node *i*. *n* is the number of mails received from node *i* (since the last report) and *s* the number of these determined by the mail host to be spam.

Note that in both messages above, the node identifier *i* could be an IP address, a hostname, a mail domain name or any other identifier supported by the mail host. It is up to the trust manager to process this.

Sending a `bulkMail` message conveys the same information as would several `singleMail` messages. It is included in the protocol for performance reasons. With heavy mail volumes, it is more efficient to send periodic `bulkMail` updates rather than burden the server with generating a `singleMail` message per mail received.

(2) Trust recommendation: Trust manager ↔ Trust manager

Nodes collaborate to share trust information with one another. This is done by the trust managers, using the primitives outlined here. Trust managers may issue recommendations spontaneously (for example on occurrence of some event like sudden appearance of spam) or in response to a request from another node. Note that a request may be issued by one trust manager to another, but a reply is not guaranteed. As mentioned earlier, message passing is asynchronous and the protocol requires no state information to be maintained by the corresponding entities. The following five messages may be sent from one trust manager to another:

- (`getTrust`, *k*) allows one node *i* to ask another node *j* to report its trust in node *k*. This message should be interpreted as an indication from node *i* of a desire to receive a recommendation regarding node *k*. Node *j* may respond with a trust report. Nodes are not obliged to respond, even if they have knowledge or experience of *k*. Nodes could have other reasons (e.g. lack of trust in the requester, *i*) to not reply. Nodes that have no information are expected to stay silent.
- (`getTrustResponseRequested`, *k*) is a variation on `getTrust` that expects the recipient to respond (with a null value) even if it has no trust information on node *k*. Again, there is no strict obligation to respond.
- (`getTrustAll`) allows one node *i* to ask another node *j* to report its trust in all known nodes. This message should be interpreted as an indication from node *i* of a desire to receive a recommendation regarding all nodes of which the recipient is aware. There is no obligation on the recipient to respond. Nodes that have no information on any nodes should issue a null reply.
- (`setTrustReportingPreferences`, *t, c, r, f*) allows node *j* to specify to node *i* how spontaneous trust reports are sent to it (see discussion on "push" model later). This message also indicates a desire by node *j* to receive trust advertisements from node *i* (i.e. to be included in its neighbourhood).
  t is a list of zero or more trust level thresholds. Trust updates are requested whenever trust exceeds or falls below any of these threshold values.
  c is a confidence level threshold. Trust updates are only desired if the confidence level of the sender is at least *c*.
  r is a recency threshold. Trust updates are only desired if the trust information has been updated by the sender within the previous *r* time units.
  f indicates the maximum frequency of update.
- (`trustReport`, *k, T*) allows node *j* to send a recommendation to another node *i*, in relation to node *k*. *T* is an object that encapsulates sender *j*'s trust in node *k*. t is set to null in the case where the sender *j* has no trust information regarding node *k*. Note that a `trustReport` may be issued either spontaneously or in response to a `getTrust` or `getTrustResponseRequested` message.
- (`bulkTrustReport`, *l*) allows node *j* to send a recommendation to another node *i*, in relation to a set of nodes. Parameter *l* is a set of pairs (*k, T*) where *k* is the node identifier and *T* is an object that encapsulates sender *j*'s trust in node *k*. *l* is the empty set in cases where the local node has no trust scores to share. Note that a `bulkTrustReport` may be issued either spontaneously or in response to a `getTrustAll` request.

TABLE I.
REPRESENTATION OF TRUST SCORES, FROM PERSPECTIVE OF A SPECIFIC NODE (smtp1.foo-inc.com)

Node name	Trust score	Confidence	Recency
smtp2.foo-inc.com	0.99	0.99	100
mail.barfoo.org	0.95	0.95	200
labserver.uxy.edu	0.80	0.80	700
webmailprov.com	0.50	0.50	30
lazyconfig.net	0.25	0.25	10
phishysite1342.com	0.01	0.01	8000

(3) Policy update: Trust manager → Mail host

The third part of this collaboration architecture is responsible for closing the loop. Direct experience is recorded by nodes and shared among them. The result of this experience and collaboration then used to inform the mail host to allow it to operate more effectively. Specifically, the mail host, on receiving a mail from node *i* needs to be able to access the current trust information that its trust manager has on node *i*. Although the mail host may be able to use local storage to, for example, cache trust values, we do not place any such requirements on it. Thus we need the ability for the mail host to request trust information from the trust manager and receive a timely reply.

The request message:

- (`getTrustLocal`, *i*) allows the mail host to request the trust score for a particular node, *i*.

The reply message:

- (`trustReportLocal`, *i, T*) allows the trust manager to respond to a `getTrustLocal` request. *T* is an object that encapsulates the trust manager's trust in node *i*. t is set to null in the case where the trust manager has no trust information regarding node *i*.

VI. Using the TOPAS protocol

A. Trust Score Initialisation

The Internet mail architecture is highly dynamic – new nodes appear all the time, and existing nodes will quite frequently receive mail from previously unknown senders. In distributed trust management generally, choosing an initial value of trust to assign to such new arrivals is a non-trivial task, and in practice depends on the application.

There are essentially two options (where the range of trust is (0,1)):

1) Initialise the trust level at zero. The idea here is that correspondents have no trust initially and must earn it. The motivation for this approach is to avoid the risk of the so-called *Sybil attack* [13] – this is where attackers (spammers) try to gain advantage by repeatedly creating new identities.

2) Assume some "default trust" exists. Initialise the trust level at some value greater than zero. Subsequent behaviour may cause this trust level to either rise or fall. The motivation for this approach is to distinguish bad guys from the unknown.

The great benefit of Internet mail is in being able to advertise a mail address and receive mail from anyone even if this person is previously unknown. Treating these previously unknown senders the same as spammers would restrict the utility of email as it tends to block new entrants; this points to the second option as perhaps the best, though we will use simulations and experience to best determine this.

B. Issuing inter-node trust advertisements

The functions specified at interface (2) above allow for either a "pull" or a "push" model for sharing trust information. Messaging is asynchronous for experience reports and recommendations.

"Pull" model. The trust manager receives a `getTrust`, `getTrustResponseRequested`, or `getTrustAll` request for trust information about another node, or all nodes, and subsequently replies with a `trustReport` or `bulkTrustTeport` message. The sender of the request will need to use its own timeout mechanism. In the case of a `getTrust` message, there is no obligation on the recipient to reply at all. With `getTrustResponseRequested` and `getTrustAll`, the recipient should issue a reply even if this contains no trust information.

"Push" model. The trust manager may be configured to spontaneously issue trust updates, either to other nodes or to its local mail server. This would typically be for reasons of performance and efficiency. It is wasteful for nodes to repeatedly poll each other unless there is useful new information. In the "push" case, each node decides when and to whom to issue trust advertisements?

When to issue trust advertisements?

Each individual node controls the frequency of issuance of trust advertisements, and may even decide to issue none. Issuing a trust advertisement uses processing, memory and network resources of both the sender and recipient. The sender needs to find a balance – the more collaboration the more effective the system, but sending too many updates may put a strain on resources. It is also desirable if the recipient can control the number or frequency of advertisements from the sender. A sender trust advertisement strategy might have them sent periodically or on significant change in trust value or confidence level in this value. For example, the sender may issue trust advertisements relating to a node when its trust in that node exceeds a certain threshold *and* when its confidence level in this trust is high. If the trust level exceeds the threshold but its confidence level in this value is low, it may choose not to send anything.

The TOPAS protocol allows this to be based on the recipient's previously specified trust reporting preferences (using `setTrustReportingPreferences`).

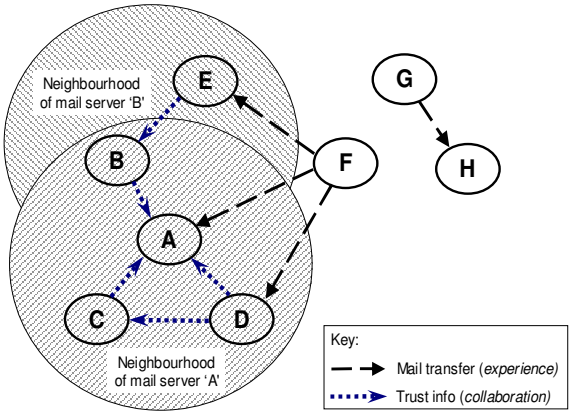

Figure 2. Distribution of trust information

To whom to issue trust advertisements?

Each individual node also controls the set of nodes that forms its "neighbourhood" – those nodes to which it sends trust advertisements. The set of neighbours could be defined, for example, as containing nodes that are nearby, most trusted, most collaborative, and/or have specifically requested trust reports (using `setTrustReportingPreferences`),

Fig. 2 illustrates the distribution of trust information. There are eight nodes (mail servers) in the example shown. Each node has its own neighbourhood, two of which are shown. In some short time duration, node 'F' sends mail to 'A', 'D' and 'E'. Node 'A' establishes a trust score for node 'F' based on (i) mail it receives directly from 'F' and (ii) reputation information from its neighbours, 'B', 'C' and 'D'. The reputation information provided by node 'B' relates to the experience of node 'E' that 'E' has shared with it. The reputation information provided by node 'C' relates to the experience of node 'D' that 'D' has shared with it. Note that trust transitivity may allow a trust score to propagate quite some distance – in this example, the score that 'E' records for 'F' is propagated to 'B' and onward to 'A'.

C. Handling experience reports

On receipt of a `singleMail` message from the mail host relating to node *j*, the trust manager at node *i* updates its trust value in node *j*, $T_{i,j}$, as follows:

$$T_{i,j} := f_e(T_{i,j}, S),$$

where f_e is a function defining how trust is updated and S is a binary value indicating node *i*'s spam filter's determination of whether the received message was spam (i.e. $S = 0$ for spam and 1 for non-spam).

One example is to update trust as follows:

$$T_{i,j} := \alpha S + (1-\alpha)T_{i,j}$$

In this example, $T_{i,j}$ can be viewed as the exponential average trust level, with parameter α that can be viewed as the *rate of adoption of trust*, $0 \le \alpha \le 1$. Note that having $\alpha = 0$ means that the trust value is unaffected by the experience. Having $\alpha = 1$ means that local trust is always defined by the latest experience and no memory is retained. In general, the higher the value of α, the greater the influence of recent mails from a node on the trust value maintained for that node. Lower values of α encourage stability of the system.

If a succession of experiences of receiving mail from node *j* return the same spam determination, *S*, then the trust value $T_{i,j}$ converges towards *S* (towards 0 for a sequence of spam messages or towards 1 for a sequence of normal messages).

D. Handling recommendations from other nodes

On receipt of a `trustReport` from node *j*, indicating a level of trust in node *k*, the trust manager at node *i* updates its trust value in node *k* as follows:

$$T_{i,k} := f_r(T_{i,k}, T_{i,j}, T_{j,k}),$$

where f_r is a function defining how trust is updated. In our simulations, we update trust as follows:

$$T_{i,k} := T_{i,k} - \beta T_{i,j}(T_{i,k} - T_{j,k}),$$

where β is a parameter indicating the level of influence that "recommender trust" has on local trust, $0 \le \beta \le 1$. Note that, the larger the value of $T_{i,j}$, (i.e. the more *i* trusts *j*), the greater the influence of *j*'s trust in *k* on the newly updated value of $T_{i,k}$. If $T_{i,j} = 0$, (i.e. *i* has no trust in *j*), this causes the local trust value in *k*, $T_{i,k}$, to be unchanged.

VII. Using Trust Scores to Filter Email

Assume that a spam filtering system applies some test to each incoming email. In each case, a decision is made whether to accept the mail. A negative result means that the mail is accepted and a positive result means that the mail is rejected (or at least marked as spam).

Most spam filters combine a variety of measures into a suspicion score and compare this with a *pre-defined* threshold. This threshold is a fixed value, which may be tuned manually. Mail resulting in a score above the threshold is marked as spam and the remainder (under the threshold) is accepted.

In our system, we attempt to improve spam filtering by allowing the threshold to vary. The threshold level depends on the trustworthiness of the node that sent the message. So, we use the sender's trust score (as perceived by the receiver) to define the threshold – i.e. the more trusted a node is, the higher we set the threshold for marking a new mail message as spam. Conversely, if a node is untrusted then the threshold is set to a lower value. There are several ways to cause this threshold to vary. In our initial experiments, the threshold for mail received from a server is simply a linear function of the trust score of that server. As the mean of the trust score range is 0.5 and the default threshold for *SpamAssassin* is 5, we set the threshold to be simply ten times the trust score for the purposes of our experiments.

In practice, the dynamics of trust applied to spam filtering allows an organisational mail server to process email in a way that depends on its trust in the sending node. In many cases, this trust level will be somewhere in the middle, between "trusted" and "untrusted". Table II illustrates possible effects of recording trust scores for a variety of mail servers, and desired implicit classification.

In the current (fixed threshold) situation, consider where the spam filter threshold is 5.0. All incoming mail is given a spam score. If a genuine message scores 5.1, it is diverted to the spam box. If a spam message scores 4.9, it is allowed into the user's inbox.

If we instead have a dynamically tuned threshold, however, a genuine message from a trusted source scoring 5.1 will be accepted as this source should have a higher associated threshold (e.g. the business partner category in Table II). Likewise, spam received from an untrusted server scoring 4.9 will be filtered out as the threshold should be lower (e.g. the "lazy" configuration category in Table II).

VIII. EXPERIMENTS

This section reports on an illustrative implementation of the TOPAS protocol with a simulated network of mail servers generating traffic, some of which is spam. We show how improvements in spam filtering (in terms of reduced false positive and false negative rates) can be achieved through closed loop control based on sharing trust scores.

A. Simulation set up

In our simulations, each node has a *neighbourhood* defined. This is a set of nodes that are somehow "close" to the node in question, with the expectation of above average frequency of communication with them.

Neighbourhoods are defined randomly for each mail server. This is done as follows: Initially, the neighbourhood of each node is the empty set. Choose two nodes at random. Add one to the neighbourhood of the other. Repeat until the total set of neighbour relations is equivalent to a connected graph. Having a connected graph means that trust can (eventually) propagate throughout the network.

Email traffic volumes also vary randomly in practice, with spammers tending to produce email in greater quantities than regular email users. For each mail sent, the recipient can be anywhere on the network, but is more likely to be a neighbour.

Each email contains a value S' that we use to model aggregated indicators of spam, in the style of *SpamAssassin*. This value is used by the receiving node to test for spam.

For our simulations, S' has a Gaussian (normal) probability density function with mean μ and standard deviation σ. Mail that is actually spam tends to have a high value of μ. Normal mail tends to have a lower value of μ. The standard deviation determines the tendency for the filtering system that analyses such mail to be prone to false positives and false negatives.

For the purposes of our experiments, whether or not an email is actually spam is indicated by a binary value that is communicated separately to the receiver. This is not used in spam detection, but is used afterwards in the evaluation of how well the spam filter worked.

B. Effects of varying rates of adoption of trust

How trust converges to stable values in our system depends on several factors, including the size of the network, how neighbourhoods are defined and the extent of spam and, perhaps most importantly, the algorithm used to update trust scores based on experience and third party recommendations.

Fig. 3 compares the option of using direct experience only to update trust with the combination of direct experience and frequent recommendations from neighbours. In this example, there are fifty "good" nodes in the network and there is no spam, meaning that all trust values should eventually converge to 100%. Setting parameter β to zero removes any effect of recommendations. Note that, in the direct experience only case ($\alpha = 0.1$, $\beta = 0$), trust level converges less smoothly – each jump occurs when a (non-spam) mail is received from the node in question. Recommendations allow trust values to be updated on receipt of mail by *another* node.

Fig. 4 examines the effect of the sizes of parameters α and β on convergence. Not surprisingly, the higher their values, the faster is convergence. It is important not to set these values too high though where spammers are at work to avoid the risk of trust values oscillating widely.

The default (initial) trust value is set to 0.2 for all nodes in these simulations.

C. Using Trust to Enhance Mail filtering

In our experiments, we examine two different approaches to mail filtering:

- Use of a pre-defined threshold. The value of the threshold is selected to be half way between the means of the probability density functions of S' for spam and non-spam respectively. Our experiments are designed so that this mean is 5.0 (a common SpamAssassin threshold).
- Use of an automatic threshold that is directly related to trust in the sending node.

The first approach takes no account of trust information. The second uses trust information to tune the spam filter.

We now illustrate how improvements in spam filtering

TABLE II. POSSIBLE EFFECTS OF RECORDING TRUST SCORES FOR A VARIETY OF MAIL SERVERS

Trust score range	Category of mail server	Spam filter threshold (TrustScore * 10)	Effect
1.0	Internal host	10	All mail accepted
0.8 – 1.0	Business partner	8 – 10	Most mail accepted
0.7 – 0.8	University/College	7 – 8	Mail checked for spam but is probably ok
0.5 – 0.7	Popular ISP	5 – 7	Mail checked for spam
0.3 – 0.5	System with little user verification; e.g. webmail	3 – 5	Mail checked thoroughly for spam
0.1 – 0.3	"Lazy" configuration	1 – 3	Spam quite likely
0 – 0.1	Open mail relays; known spam sources	< 1	Most mail flagged as spam

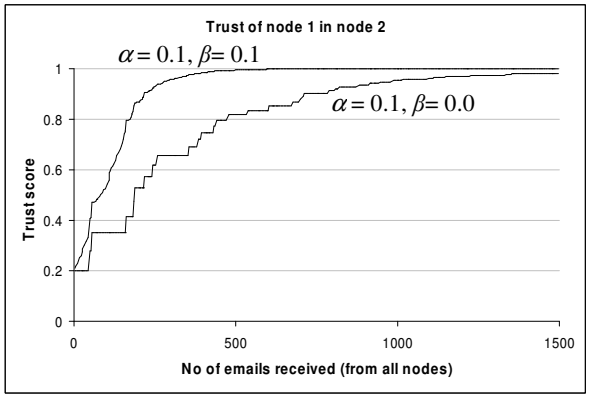

Figure 3. Effect on convergence of using recommendations to update trust score

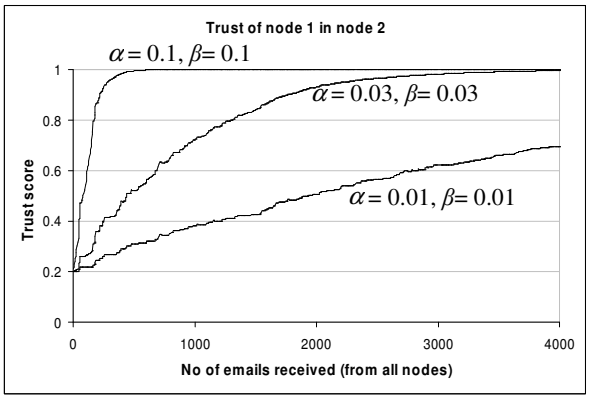

Figure 4. Effect on convergence of varying parameters α and β

(in terms of reduced false positive and false negative rates) can be achieved through closed loop control based on sharing trust scores. We simulate a network of fifty nodes, of which a single one is a spammer. The spammer is responsible for 50% of all email generated in the system. Trust convergence for "good" nodes is moderately fast, with parameters α and β both set to 0.03. The randomly generated neighbourhood of each node consists on average of one-seventh of all nodes.

For this experiment, we choose relatively flat (but distinct) probability density functions for spam indicators for both spam and non-spam email. Both have the Gaussian (normal) distributions shown in Table III. Note the overlap implied by the relatively large standard deviation values.

As already mentioned, most spam filters combine a variety of measures into a suspicion score and compare this score with a pre-defined threshold. For our experiments, a fixed threshold of 5.0 is chosen (*SpamAssassin* default) and used as a benchmark. As can be seen in Fig. 5 and Fig. 6, a significant reduction in both false positives and false negatives can be achieved with auto-tuning of the threshold based on trust values. Auto-tuning is of course most effective in a steady-state situation when trust values are quite stable. A range of other predefined threshold values were also tried, but with no significantly better results than the value of 5.0 shown. Choosing a higher predefined threshold causes an increase in false negatives and choosing a lower predefined threshold causes an increase in false positives.

IX. Related Work

Specific spam filtering techniques are not of direct concern to us in this paper and can effectively be plugged in as needed. Our emphasis is on using trust information to tune such filters, and our initial implementation focuses on filters that are based on thresholds. This section overviews other work that uses collaborative techniques to fight spam, and draws attention to similarities and differences in approach to ours.

There has been some other work on applying trust and reputation information to spam filtering.

Golbeck and Hendler [14] present a technique based on social networks for sharing reputation information among email users. This allows email users to sort received messages based on a rating value associated with the sender. This rating value is set by the user or is inferred from neighbours' ratings using a weighted average technique. Our approach differs from this in two main ways: firstly, we focus on mail servers rather than individual email addresses; secondly, our system provides closed loop feedback without direct user involvement.

Kong et al. [15] also focus on end user email addresses, but provide for the anonymous sharing of information with a wider community. When a user flags a mail as spam, this is made available to other users' spam filters, which is useful as the same spam messages are usually sent to a large number of users. Damiani et al [16] similarly present a system of sharing spam information, wherein a cryptographic digest of each spam mail encountered is shared.

Seigneur et al [9] use "hard" cryptographic

TABLE III.
PARAMETERS FOR GAUSSIAN (NORMAL) DISTRIBUTIONS USED IN EXPERIMENTS

	Mean	Standard Deviation
Spam	8.0	4.0
Non-spam	2.0	4.0

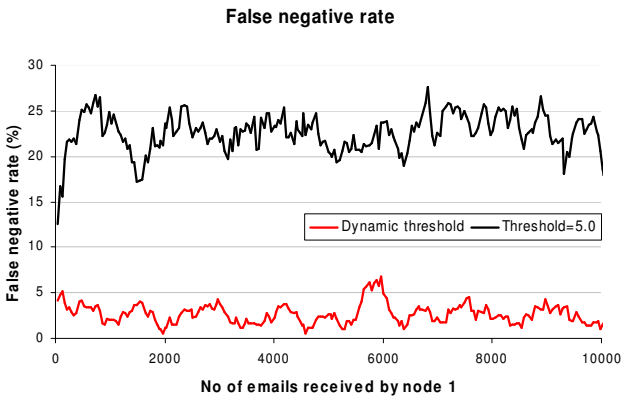

Figure 5. Comparison of dynamic vs fixed threshold: impact on rate of false negatives.

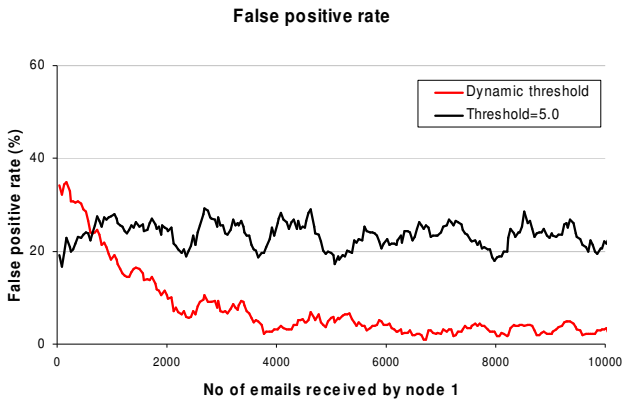

Figure 6. Comparison of dynamic vs fixed threshold: impact on rate of false positives.

authentication to support whitelists of known good guys in conjunction with "soft" trust management for new contacts. Trust derives from stored *evidence*, which includes recommendations, observations, certificates and reputations. Though our trust model is similar to this ("observations" are called "experience" in our model, and we consider certificates and reputation to be the same as recommendations), our work uses this information for filter tuning and investigating the dynamics of the resulting system. Also, we focus on mail servers rather than individual mail users, as mentioned above.

Foukia et al [17] are, like us, motivated to encourage mail servers to restrict output of spam. Their approach is agent-based – each participating mail server has an associated Federated Security Context Agent that contributes to, and draws on, an aggregated community view (the Federated Security Context). They also use quotas to control the volume of mail output by a server in an attempt to prevent temporary traffic bursts that are typical of spammer activity.

X. Conclusions and Further Work

A new approach to improving spam filtering, based on collaboration between mail servers to manage trust, has been described in this paper. A trust management overlay architecture and a new lightweight protocol have been presented. In this system, each mail server records trust measures relating to each other mail server of which it is aware. Trust by one mail server in another is influenced by direct experience as well as recommendations issued by collaborating mail servers. The TOPAS protocol specifies how these experiences and recommendations are communicated between each spam filter and its associated trust manager, and between trust managers of different mail servers. A technique for improving mail filtering performance and the TOPAS protocol using these trust measures has also been described. Experimental results have illustrated use of the protocol in a simulated network scenario, and indicate the potential of this approach to significantly improve rates of false positives and false negatives in anti-spam systems.

There is significant scope for further work. We have just presented simulation results based on a straightforward averaging strategy for updating trust. Work is required on refinement of strategies to be effective against resourceful spammers. In particular, any trust-based anti-spam system needs to be robust in the face of attackers who try to corrupt trust ratings, possibly in collusion with one another. Game theoretic approaches are worth considering in this regard.

Furthermore, our initial voting (averaging) strategy does not provide any incentive for nodes to vote – why would a node bother to share information with other nodes. Building in some kind of incentive (e.g. increased trust rating for active participation) into trust update strategies should assist with this.

It would also be useful to evaluate the integration of our system with various a real mail infrastructure, including a variety of mail server and spam filters.

Acknowledgment

This work was supported by the European Commission (*OPAALS* FP6 project) and by Science Foundation Ireland (*Foundations of Autonomics* project).

References

[1] J. McGibney and D. Botvich, "A trust overlay architecture and protocol for enhanced protection against spam", Proc. 2nd International Conference on Availability, Reliability and Security (ARES), Vienna, Austria, pp 749-756, April 2007.

[2] J. Klensin (ed.), *Simple mail transfer protocol*, RFC 2821, Internet Engineering Task Force, 2001.

[3] J.B. Postel, *Simple mail transfer protocol*, RFC 821, Internet Engineering Task Force, 1982.

[4] P.R. Zimmermann, *The official PGP user's guide*, MIT Press, 1995.

[5] D. Gambetta, "Can we trust trust?", In *D. Gambetta (Ed.), Trust: making and breaking cooperative relations*, pp 213-237, Blackwell, 1998.

[6] A. Schwartz, *SpamAssassin*, O'Reilly, 2004

[7] J. Goodman and R. Rounthwaite, "Stopping outgoing spam", *Proc. ACM Conference on E-Commerce*, New York, 2004.

[8] M. Abadi, A. Birrell, M. Burrows, F. Dabek, T. Wobber, "Bankable postage for network services", *Proc. 8th Asian Computing Science Conf.*, 2003.

[9] J.-M. Seigneur, N. Dimmock, C. Bryce, and C. Jensen, "Combating Spam with TEA (Trustworthy Email Addresses)", *Proc. 2nd Annual Conf. on Privacy, Security and Trust (PST)*, Fredericton, Canada, 2004.

[10] R. Roman, J. Zhou, J. Lopez, "Protection against Spam Using Pre-Challenges", *Proc. 20th IFIP International Information Security Conf.*, Chiba, Japan, 2005.

[11] M. Naor, "Verification of a Human in the Loop or Identification via the Turing Test", 1996, *Unpublished manuscript*: http://w.wisdom.weizmann.ac.il/~naor

[12] M. Dam and R. Stadler, "A generic protocol for network state aggregation", *Proc. Radio Science and Communication conference* (RVK), Linköping, Sweden, 2005.

[13] J. Douceur, "The Sybil attack", *Proc. 1st Int'l Workshop on Peer-to-Peer Systems*, 2002.

[14] J. Golbeck and J. Hendler, "Reputation network analysis for email filtering", *Proc. 1st Conf. on Email and Anti-Spam (CEAS)*, Mountain View, CA, USA, 2004.
[15] J. Kong, B. Rezaei, N. Sarshar, V. Roychowdhury, and P. Oscar Boykin, "Collaborative spam filtering using e-mail networks," *IEEE Computer*, vol. 39, no. 8, pp. 67-73, 2006.
[16] E. Damiani, S. de Capitani di Vimercati, S. Paraboschi and P. Samarati, "P2P-based collaborative spam detection and filtering," *Proc. 4th IEEE Int'l Conf. on Peer-to-Peer Computing*, 2004.
[17] N. Foukia, L. Zhou, C. Neuman, "Multilateral decisions for collaborative defense against unsolicited bulk e-mail", *K. Stolen et al (Eds.): iTrust 2006,* LNCS 3986, pp 77-92*,* 2006.

Jimmy McGibney obtained a BE from University College Dublin, Ireland, in 1992 and an MEng from Dublin City University in 1994. He currently is a lecturer in applied computing at Waterford Institute of Technology. His main research interests are in trust models and distributed intrusion detection and spam prevention. He also worked for several years as a software developer in the telecoms industry.

Dmitri Botvich received BSc and PhD degrees in mathematics from Moscow State University, Russia, in 1980 and 1984 respectively. He currently is a principal investigator with the TSSG at Waterford Institute of Technology. He has over twenty years of research experience, including periods in industry, and has over fifty refereed publications in telecommunications, queuing theory, computer geometry and mathematical physics. He also has five patents and patent applications. His main current research interests are in the principles, architecture and methods to support autonomic management in future networks.