The following paper was originally published
in the Proceedings of the
Tenth USENIX System Administration Conference (LISA X)
Chicago, IL, USA, Sept. 29 - Oct. 4, 1996

Automatic and Reliable Elimination of E-mail Loops Based on Statistical Analysis

E. Solana, V. Baggiolini, M. Ramluckun, and J. Harms
University of Geneva (Switzerland)

Automatic and Reliable Elimination of E-mail Loops Based on Statistical Analysis

E. Solana, V. Baggiolini, M. Ramluckun and J. Harms
– University of Geneva (Switzerland)

ABSTRACT

One of the major threats to the E-mail service are message loops, mail messages which endlessly circulate between a number of E-mail relays and severely deteriorate or even interrupt the messaging service. Although the standards ruling E-mail communications include mechanisms to prevent message loops, these mechanisms are often unimplemented and, if they are, can fail to work reliably under certain circumstances. This paper describes a novel approach to the automatic and reliable elimination of looping messages. Statistical properties of the message traffic are analyzed to detect messages that might be looping. Such messages are earmarked, and if they are observed to actually be looping, they are eliminated. This method has been implemented and successfully applied in the multi-protocol E-mail environment of the University of Geneva.

Problem description

A message is said to be looping if it is forwarded endlessly in a circular manner among a set of *mail relays* (*Message Transfer Agents* or *MTAs*). Normally, two or three mail relays take part in a message loop, but there can be more than that. The period of a loop (the time it takes to complete) ranges from a few seconds to a few minutes. Looping messages usually grow in size, since mail relays typically add header lines to messages they process.

A message that loops during a weekend can be forwarded tens of thousands of times, growing in size to megabytes and producing megabytes of logging data. The situation is even worse if *distribution lists* are involved (c.f. [1]): the number of messages explodes and soon the mail system crashes under the load. These scenarios illustrate how important it is to detect and eliminate looping messages *automatically*.

Loops are caused by wrong addressing configuration on one or more systems:

- *bad configuration* of mail relays, e.g., a relay that does not "recognize" messages addressed to it and, instead of keeping them for local delivery, forwards them to a central mail relay which sends them back again[1].
- *automatic forward mechanisms* such as aliases, distribution lists, or personal forward mechanisms (`.forward files`) pointing to invalid addresses.
- *automatic reply mechanisms*, such as vacation programs or non-delivery reports.

The mechanisms that are directly under control of a proficient administrator account for only a small part of the errors. The problem lies much more in the forwarding facilities that are under user control, such as `~/.forward` files and vacation programs or even mail relays installed without the administrator knowing. Furthermore, even if all the mail relays in a given responsibility domain are perfectly configured, a mail transaction with a misconfigured or non-compliant site may result in a message loop.

Standard Mechanisms For Loop Detection

There are two typical countermeasures that are integrated into the mail relays or taken by postmasters:

- *Counting the* `Received from` *lines* in the message header. RFC 821 [2] says every mail relay processing a message must add such a line to the message header and count the number of `Received from` lines already contained. When the header encloses more than a pre-defined number (e.g., 25) of such lines, the message is considered to be looping and automatically sent back to the expeditor together with a non-delivery report. Fixing such a limit is quite a delicate task (especially due to the growing complexity of the Internet).
- *Tracing* `Message IDs`. `Message IDs` allow to uniquely identify E-mail messages. Postmasters can inspect the log-files and check if the same `Message ID` recurs multiple times. If this is the case, the corresponding message is considered to be looping.

Both of these approaches work well if there are only mail relays of one protocol involved. However, they can fail in a multi-protocol environment (e.g., `SMTP` [2, 3] together with `X.400` [4][2]) or in case of

[1]This is the behavior of many pre-installed *sendmails*.

[2]Although the standard for protocol conversion (RFC-1327 [5]) describes how trace fields should be converted, a number of mail relay products are not compliant to these recommendations.

faulty configurations or non-compliance to the protocol recommendations. When a message is translated into a different E-mail protocol by a gateway, `Received from` lines can get lost; some gateways can even be explicitly configured to eliminate them. The same problem exists for `Message ID`s: some mail relays change the old `Message ID` and assign a new one. Furthermore, both methods fail if a new message is generated in the looping process (e.g., `vacation` or non-delivery reports).

Related work

Most existing solutions rely on the mechanisms described above and are normally included in mail relay implementations. A recent work being conducted by D. Bernstein [7] proposes a set of recommendations for so-called *automailers* (a generic term designating mail relays, list exploders, vacation programs, etc.). Due to the great number and diversity of existent software implementations, compliance to these recommendations will not be achieved in a foreseeable future (if ever...) and the need of external mechanisms, as the one proposed in this paper, will persist.

To our knowledge, there is no independent, commonly agreed-on solution for preventing loops between E-mail systems. The approach adopted by some administrators consists in periodically observing the message traffic for a given period of time in order to find potential loops.

Unfortunately, the most common practice consists in reacting “a posteriori”, alerted by the unpleasant effects of a loop in the E-mail system.

A more scientific approach to solve the problem of looping messages has been studied by J. F. Paccini in his Ph.D. thesis [8]. He proposes a method to identify a loop by tracing suspect messages amongst the set of involved mail relays, the tracing information being shared by each mail relay by means of a standardized distributed database (`X.500`, [9]). The two major drawbacks of this approach are the significant management overhead introduced by a distributed database and the security issues related to the sharing of sensitive information (message trace statistics) across administrative domain borders.

Our Approach

The proposed method is based on a *statistical analysis* of the messages recently exchanged by a single mail relay. As will be explained, message loops have very particular statistical properties that are visible at a *single observation point*. We propose to exploit these properties in order to detect suspect messages without having to consider management information from other mail relays.

Since message loops occur occasionally, we have chosen to trigger the detection mechanism upon specific conditions directly related to loop problems: *abnormal message rate*, *disk*, and/or *CPU usage*.

E-mail related information can be obtained by periodically querying the mail relay console or by means of an `SNMP` agent that provides monitoring data in the form of a `MIB` (c.f. [6]). Information concerning system resources can be retrieved by means of standard UNIX commands.

The process of loop detection and suppression is composed of two phases: the first focuses on the *detection of suspect messages* based on the logging information of recent message exchanges and the second aims at the *accurate identification and elimination of messages* that do actually loop.

Detection Of Suspect Messages

The aim of this process is to identify those messages that might be looping. To achieve this, we build a *statistical database* based on the historical information (log-files) generated by the mail relay where the observation point is located. Note that the necessary data is contained in the log-files of about any existing E-mail implementation:

- *The Sender and Receiver addresses*. Since `Message ID` information is not always reliable, the only invariant of a looping message observed from one specific point are the sender and receiver addresses of the envelope. Figure 3 shows that this assertion does not hold when the observation point changes (address rewritten by involved mail relays).
  This information, that we denote *SR pair*, constitutes the *search key* of our statistical database.
- *The number of messages exchanged* by this *SR pair*.
- *Absolute time of last message exchange*. This value is used to determine whether the loop is still active or not.
- *Time lapses between occurrences* of the same *SR pair*: A set of integer values containing the time intervals in seconds between messages with identical *SR pairs*.

Based on this, we pick the most often recurring *SR pairs* for further analysis. Generally, this is a threshold-based decision which depends on site characteristics and the extent of historical information being considered. If the loop has been running for some time, the number of messages exchanged might be sufficient to identify a potential loop[3].

However, if the loop has just begun and only a few iterations have occurred, a more elaborate analysis is required to more accurately distinguish looping messages from normal traffic. This is done by performing *statistical computations* over the time lapses set, namely:

- *Mean value* of time lapses: The arithmetic mean value calculated over the above set of

[3] 3000 messages are unlikely to be exchanged by two partners in a single day.

message lapses. This value represents the frequency of arrival of suspect messages. A low mean value implies a high frequency of arrival of messages with identical *SR pairs* and, thus, a high probability of a loop.

- *Variance* of time lapses: The variance of time lapses measures the deviation from the mean value. This value is very relevant since looping messages are normally processed at uniform time intervals and, as a consequence, display a remarkably low variance. On the contrary, normal E-mail traffic has a very irregular time distribution resulting in very high variance values.

Based on this statistical information, we have elaborated a *heuristic function* for computing the *loop probability* for each *SR pair* appearing repeatedly in the log-files. Figure 1 shows the inner workings of the heuristic function that has been used to detect loops in our site.

```
/*
Variables for each SR Pair:
    number_of_messages, time_of_last_message, lapses_variance,
    lapses_mean_value;
Thresholds (depending on site characteristics):
    minimum_amount_of_messages, timefactor,
    suspect_var_threshold, suspect_mean_value
*/
function loop_probability_fct {
    loop_probability = 0; /* initial value */
    /* if the number of messages is under a given threshold don't go
       any further. (statistical computations are not sufficiently
       accurate for a small amount of msgs.) */
    if (number_of_messages < minimum_amount_of_messages)
        return(loop_probability);
    if (((current_time - time_of_last_message) / mean_value)
        > timefactor)
    /* if the time interval between the last message and the current time is
       timefactor times higher than the mean value, we assume that the
       potential loop has been fixed */
        return(loop_probability);
    /* if the variance is below a given threshold, increase probability */
    if (lapses_variance < suspect_var_threshold)
        loop_probability += 1 - lapses_variance / suspect_var_threshold;
    /* ditto with the mean value */
    if (lapses_mean_value < suspect_mean_value)
        loop_probability += 1 - lapses_mean_value / suspect_mean_value;
    if (number_of_messages > suspect_number_of_messages)
    /* if the number of messages exchanged is abnormally high, increase
       probability */
        loop_probability += number_of_messages /
            total_number_of_exchanged_messages;
    if (loop_probability > 1)
        return (1);
    else
        return(loop_probability);
} /* loop_probability_fct */
```

Figure 1: Heuristic Function Computing Loop Probability

Firstly, the function tests if the number of messages exchanged is sufficiently high to ensure accurate statistical computations. The lapse between the last message and the current time is compared to the mean value of previous transactions lapses in order to detect if the potential loop is still active. If both tests yield positive results, the effective loop probability is computed. Low values for either the variance or the mean value increase loop probability. An abnormally high number of messages exchanged will also augment the result returned by this function. It should be noted that a suspect value in any of these parameters will have significant impact in the overall loop probability returned by the function.

The following example shows the results of the loop probability computations for a message loop (Figure 2a) and for a regular group of messages exchanged by an *SR pair* (Figure 2b). In the first case, note both the high frequency (approx. one message every 20 seconds) and the uniform distribution of time lapses between message arrival times.

This results in very low mean and variance values and, as a consequence, in a very high (close to 1) loop probability value. In the second example, where normal mail traffic is analyzed, the high variance and mean values yield a remarkably low loop probability.

Figure 3 shows a typical loop scenario where a message (addressed to `loop@mail[123]`) circulates endlessly between three mail relays (for instance, due to an incorrect forward address combination for the `loop` user). The statistics are retrieved at the *observation point* number 1.

Note that due to the high number of messages exchanged and the low mean and variance values, the loop probability (`loopprob`) for this message is very high compared to the remaining messages. As a consequence, the *SR pair* `[a@mail3 | loop@mail1]` is considered to be suspect and used as input for the next processing stage.

```
MESSAGES FROM solana@cui.unige.ch
TO = loop@bad.domain
NUMBER OF MESSAGES EXCHANGED = 75
TIMELAST = 05/04 10:29:55
LAPSES = [17, 17, 32, 18, 17, 18,
          31, 33, 18, 19, 17, 17,
          17, 34, 17, 23, 20, 22,
          17, 18, 17, 20, ...]
MEAN VALUE = 20
VARIANCE = 66
LOOP PROBABILITY = 0.97
```

Figure 2a: Statistical Properties of a Loop

```
MESSAGES FROM solana@cui.unige.ch
TO normal@cui.unige.ch
NUMBER OF MESSAGES EXCHANGED = 70
TIMELAST = 05/04 10:25:53
LAPSES = [205, 1800, 3357, 264, 45,
          188, 256, 313, 294, 141,
          107, 135, 161, 689, 1267,
          691, 5402, 24640, ...]
MEAN VALUE = 1125
VARIANCE = 85443
LOOP PROBABILITY = 0.01
```

Figure 2b: Statistical Properties of Normal Messages

The next phase will permit to accurately determine if the selected *SR pairs* actually correspond to a message loop. If yes, the necessary corrective actions are carried out.

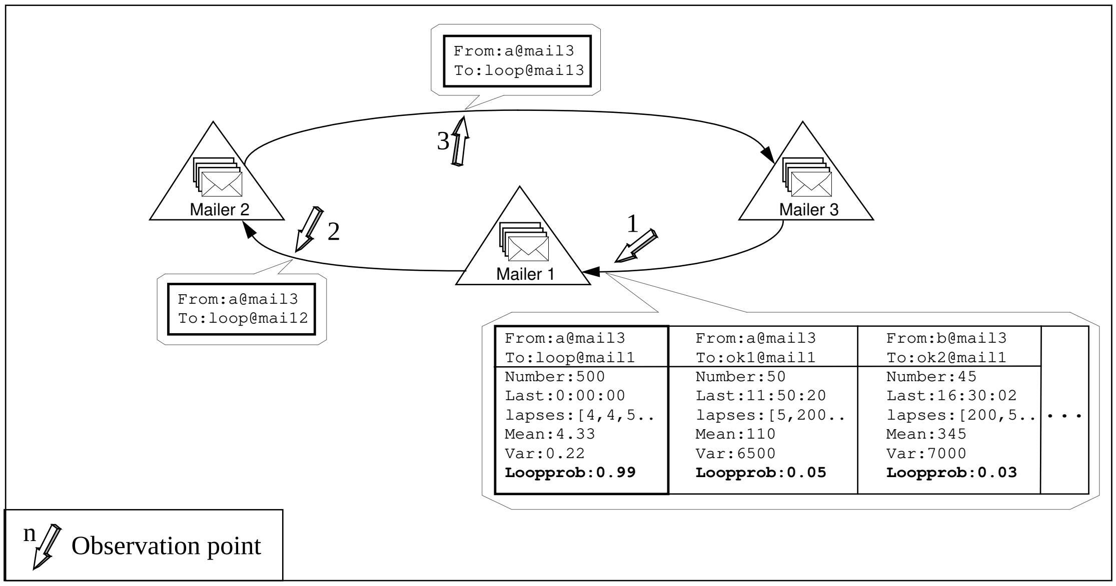

Figure 3: Detection of Suspect Messages

Identification and Elimination of a Message Loop

If suspicious *SR pairs* have been identified, the program will actively wait for the corresponding messages to arrive and carry out the following actions for each of them:

- *Freeze the suspect incoming message* in order to enable write operations on the message header.
- *Mark the message with an Unique Identification Tag* (`UIT`) using a reserved *header record*[4] and a well-defined string (containing, for instance, the mail relay identification plus an exact time value).
  The `UIT` of each suspicious *SR pair* will be recorded for further identification.
- *Flush the message*, i.e., force the message to leave the mail relay again.
- *Detect the tagged message*. Wait for the tagged message to arrive again. The arrival of a message containing both a suspicious *SR pair* and an `UIT` means that the message has been processed twice by the mail relay with identical envelope addresses. This fact is an unequivocal loop symptom. If the envelope addresses were different, the presence of an `UIT` alone would not constitute a sufficient condition for this assertion since regular messages might be processed twice by a mail relay as a result of a user forward to an external address[5].
- *Remove the message loop* . The immediate action to eliminate a message loop may consist in "bouncing" the message back to the sender with an error diagnostic or to discard the message and generate a report directed to the local E-mail administrator. Further actions must then be undertaken to diagnose and fix the actual reasons (configuration errors, protocol failures, etc.) of the loop.

Implementation Experience

The proposed loop detection and removal mechanism has been fully implemented in two central mail relays of the operational E-mail service in the University of Geneva: the main gateway between `X.400` and `SMTP` and an `SMTP` relay containing user mailboxes. Apart from a few message loops that were introduced for testing, both mail relays are subject to normal operating conditions.

During a short calibration period, parameters of the heuristic function were tuned to the particular traffic. After this period, the system could distinguish with remarkable accuracy looping messages from regular traffic. When combined with the *mark and remove* mechanism described above, the system automatically eliminate all possible message loops.

The loop detection script has been implemented in PERL and is currently used by postmasters of the Geneva University network and by SWITCH, the major Swiss Internet service provider for academia. The package is freely available at `http://cuiwww.unige.ch/~solana/loop-watch.tar`.

This work is part of a larger project[6] that focuses on distributed application management with special focus on E-mail management.

Conclusions

During several months of utilization, the loop finder script has automatically detected and removed several message loops that in the past would have remained undetected for several days and might have caused serious problems to the E-mail service. This shows that even the analysis of basic statistical parameters is sufficient to distinguish looping messages from ordinary traffic.

As described in the previous section, the loop detection process obtains the message informations from the log-files of mail relays implementations. A version that processes the information provided by an `SMTP sniffer` – a program capable of analyzing SMTP transactions obtained directly "on the wire" – is being finalized. This will notably enhance the portability (no need to consider proprietary log-file formats) and the reliability (no influence from mail relay software errors).

Although the main objective of our work consisted in detecting and removing message loops, the accurate statistic information obtained, has brought significant insight into the E-mail traffic conditions of our site. In particular, relying on this information, we have operated important changes in the topology of our campus E-mail architecture resulting in an improved load balancing among message relays. Furthermore, the nature of information provided by this script can be easily exploited for accounting and billing issues.

Experience shows that in widely distributed and heterogeneous environments such as electronic mail, the recommendations of standard protocols are often not correctly implemented. Mail loops are a good example of a problem that theoretically should not happen but practically does. Furthermore, the incredible growth of E-mail utilization will certainly aggravate this problem. Techniques as the one described in this paper can detect malfunctions before the quality of service provided to end-users degrades.

---

[4] The header record should be carefully chosen to allow multi-protocol inter-operability.

[5] As a consequence of the address rewriting inherent to the forward process, the message enters and leaves the mail relay with different recipient addresses.

[6] Project funded by the "*Fonds National suisse pour la Recherche Scientifique*" (FNRS).

Author Information

Eduardo Solana received his M.Sc. degree in Computer Science in 1991 from the University of Geneva. He has been a member of the IBM technical staff for three years. Currently he is a Ph.D. student in the Teleinformatics and Operating Systems (TIOS) group of the University of Geneva. His interests are in the fields of Computer Communications, Management of Distributed Applications, Network Security and Cryptography. Write to him via e-mail at <Eduardo.Solana@cui.unige.ch>.

Vito Baggiolini is a Ph.D. student at the University of Geneva where he has been working since 1993 on the management of distributed applications. He graduated as an electronic engineer from the Federal Institute of Technology in Zurich in 1989. Between 1990 and 1993 he worked at the European Space Agency in Frascati (Italy) in the field of computer networking.

Mira Ramluckun received her M.Sc. degree in Computer Science from the University of Compiègne (Paris). Between 1992 and 1994 she worked in the Linguistics Department of the University of Geneva in the field of Natural Language Processing. Since 1994 she is a Ph.D. student in the Teleinformatics and Operating Systems (TIOS) group of the University of Geneva. Her interests are in the fields of Computer Communications, Database Design and Natural Language Processing.

Jürgen Harms. Studies in physics at the Polytechnic School of Vienna (Austria) and in electronic engineering at the University of California (USA). Since 1972 professor of computer science at the University of Geneva where he heads the computer department. President of the SWITCH foundation (Swiss national research network). Fields of interest: Operating Systems, Teleinformatics.

Bibliography

[1] J. Udell. "E-mail adventures". *BYTE* magazine, Volume 21, Number 4. April, 1996.

[2] J. Postel. "Simple Mail Transfer Protocol". *Request for Comments 821*. August, 1982.

[3] D. Crocker. "Standard for the format of ARPA Internet text messages". *Request for Comments 822*. August, 1982.

[4] ITU and CIITT – *Data Communications Networks*. Message Handling Systems. Recommendations X.400-X.420, 1988.

[5] S. Hardcastle-Kille. "Mapping between X.400 (1988) / ISO 10021 and RFC 822". *Request for Comments 1327*. May, 1992.

[6] S. Kille, N. Freed: "Mail Monitoring MIB", *Request for Comments 1566*. January 1994.

[7] D. Bernstein. *Tools in the War on Mail Loops*. Internet-draft: draft-bernstein-mail-loops-war-00. txt. April, 1996.

[8] J. F. Paccini. *De la gestion de réseaux à la gestion d'applications: le rôle des données historiques*. Ph. D. Thesis, Université de Genève, 1994.

[9] ITU and CIITT – *Data Communications Networks. Directory.* Recommendations X.500-X.521, 1988.