International Refereed Journal of Engineering and Science (IRJES)
ISSN (Online) 2319-183X, (Print) 2319-1821
Volume 6, Issue 2 (February 2017), PP. 07- 16

Through-Mail Feature: An Enhancement to Contemporary Email Services

Y. Dowlut[1], S. Yong[2], B. Sonah[3]
[1 2 3] *Department Of Computer Science, University Of Mauritius, Mauritius*

ABSTRACT: - in many organisations where there exists several levels of hierarchy, there is often the need to route a request from one level to the next until it reaches the last level where a decision is ultimately taken. In contemporary email services, this will be usually achieved by composing a mail and either forwarding it from one intermediary to another, or carbon-copying to all intermediaries. Unfortunately, these options have several drawbacks one of which is that the content of the original request can be modified by any member in the route. In this paper, we add a through-mail feature by which a user may channel his request via a predetermined route of intermediaries entered via a purpose-built interface on the email client. The request will reside at transitbox of an intermediary for a user-specified transit time. Our transit server will have the task to monitor the transit time of a transit mail at an intermediary. Another task of the transit server is to relay a mail from one intermediary to another when former responds within the transit time or when the transit time expires. At his transitbox, the user may read the comments from past intermediaries as well as post his own. This process of email transiting is significantly more convenient, temper-proof and traceable, making it a very desired feature in an email service for many organisations.

Keywords:- email, internet message access protocol (imap), internet message format (imf), multipurpose internet mail extensions (mime), simple mail transfer protocol (smtp), transit, through-Mail

I. INTRODUCTION

Over the past few years, email has become one of the main communication tool used around the world to effectively transmit information both internally (within an institution) and externally (from one institution to another). In contrast to other forms of communication, email is a convenient, simple, fast and economical mean of delivering messages.

In this paper, we propose to add a through-mail service in the hope of making communication more efficient in an organization where there exists several levels of hierarchy. In such an organization, a request will usually originate from a member at the lowest level (originator) and is targeted to the member at the highest level (destinator) who makes the final decision. However, before making the decision, the latter has to obtain feedbacks or comments from the members at the intermediate levels (intermediaries). For example, in a company, a secretary may request for the purchase of an item from the budget director who can only approve provided the branch manager has given his consent. As another example, an academic may apply for a vacation leave to be approved by the vice-chancellor. The vice-chancellor can only concede to the request provided the head of the department and the dean of the faculty find no objection.

In a contemporary email system, the originator will either (1) send an email request to the destinator and carbon copy it to all intermediaries, or (2) send his email request to the member at the next level of hierarchy, who in turn relays it to the next member along the route. Both cases contain a number of limitations, namely, the original message and the comments from intermediaries may be altered by the current recipient before forwarding to the next level. Second, in option (1), the destinator will have to view the feedback from each intermediary in the form of individual mails received very likely in chronologically unorderly manner. Lastly, the destinator in option (2) will have the final mail in which the messages from each intermediary are concatenated as they are forwarded from one intermediary to the next (making it very hard to read, specially if the hierarchy levels are many).

To Counter The Above Limitations, Our Through-Mail Feature Will Have The Following Objectives:

1. Via a compose email screen, the system shall allow the originator to declare the email address of the destinator and the intermediaries in some desired order, plus the maximum time the mail should reside within the mailbox of each intermediary. This is will be called the transit time, within which an intermediary should post his comments before sending.
2. The system shall allow an intermediary to see the original mail plus the comments posted by previous intermediaries in the route.
3. At the server side, the system shall monitor the transit time allocated to each intermediary. If the member responds before the transit time is over, the server should channel the mail to the next intermediary and

should at the same time allocate extra transit time to that intermediary. For example, if the transit time at intermediary A is 24 hours and A has replied after just 10 hours, the server will allocate an extra of 14 hours to the next intermediary B.

4. If the intermediary fails to respond within the transit time, the server should send a notification to all previous intermediaries and to the originator, and still relay the mail to the next intermediary.

In this paper, we explore, in the literature section, the different technologies that are currently used in existing email systems. In the following section, we describe the different components of the through-mail feature which we propose to bring to the contemporary email service. Finally, we will assess how well the solution implemented works and propose some further works to be done.

II. LITERATURE REVIEW

In order to implement the through-mail feature, we need to first understand how the current email system work. An email message usually originates from a sender. The sender uses a mail user agent (MUA) to compose the message and also to specify the recipients of the message. The MUA, which is also known as an email client, then transforms the message into a special form known as the Internet Message Format (IMF) and delivers it to a mail transfer agent (MTA). The MTA is usually a Simple Mail Transfer Protocol (SMTP) server and is responsible for routing the IMF-formatted messages to the mailbox of the correct recipient(s). Once the message is in the correct mailbox, its recipient can access it using an email client. The email client would retrieve the email message from the recipient's mailbox using the Internet Message Access Protocol (IMAP) and present it to the user in a comprehensible form.

We shall describe this process in more detail in the following sections.

2.1. Internet Message Format (IMF)

The IMF [1] defines the syntax used for electronic mail messages. It was initially defined by David H. Crocker in a Request for Comments (RFC) document published by the Internet Engineering Task Force (IETF) and has since been revised several times.

The IMF specification states that messages should be divided into two sections: a header part and a body part [2]. The header section consists of several header fields. Each field appears on its own line and begins with the field's name followed by a colon and the field's body. The body section contains the message that was composed by the sender and appears after the header section. A blank line is used to separate the header section and the body section. Fig. 1 shows an example of an IMF message and Table 1 describes the example in parts.

```
From: <jarvis@bank.com>
Sender: <assistant@bank.com>
To: <mrmouse@customer.com>
Subject: Re: Loan application
Message-ID: <msg-1011@bank.com>
In-Reply-To: <msg-10@customer.com>
References: <msg-10@customer.com>
   <msg-1010@example.com>
   <msg-09@customer.com>
Date: Sun, 14 Dec 2014 19:01:52 +0400
X-Loan-Amount: 10000

Dear Mr. Mouse,

Thank you providing all the documents that we had requested. We have
decided to approve your load. The money will be debited to your account
within a week.

Kind regards,

Mr. Jarvis
Loan Coordinator
```

Fig. 1: An IMF email example

Table 1: IMF example description

Field	Description
From	Contains one or more email addresses belonging to the author(s) of the message.
Sender	Contains the email address of the person who is responsible for actually sending the message. The From and Sender fields differs in the sense that an email composed by one person may be sent by another person, for example, a secretary.
To	Contains a list of email addresses for the primary recipients of the message.
Reply-To	An optional field which can be used by the sender of the email to specify an alternate email address to which replies to the email should be provided. When this field is available, replies are sent to the specified address instead of the address specified in the From field.
Subject	Contains a short text identifying the topic of the message.
Message-ID	A compulsory field which contains a single unique message identifier that refers to a particular message. The identifier's format is similar to an email address, but does not refer to an actual email address. It is automatically generated by an email client and is not intended to be human-readable.
In-Reply-To	Used when replying to a message. This field contains the Message-ID of all messages that are being replied to.
References	May be used in addition to the In-Reply-To field when replying to a message. This field contains the Message-ID of all messages that belong to the same conversation. In the above example, the References field contains the Message-ID of three messages. This indicates that the email being composed together with these three messages form a single conversation. This information is very useful for email clients to properly group and display related emails together.
Date	Contains the date and time at which the email was composed and sent by the user.
X-Amount	This is a non-essential and non-standard header field. Its semantics are application-specific. In this case, it contains the amount of loan requested by the bank customer. The IMF specification allows non-standard header fields to be used provided that they're prefixed with "X-". This feature will be later exploited to allow the transit system to work.

Note: The IMF specification only supports plain text messages. In order to support rich contents and attachments, a very common extension to the IMF specification known as Multipurpose Internet Mail Extensions (MIME) [3] is used. However, this extension is not required for the transit system to work and will thus not be covered.

2.2. Simple Mail Transfer Protocol (SMTP)

Simple Mail Transfer Protocol (SMTP) [4] is an Internet standard initially published by the Internet Engineering Task Force (IETF) as a Request for Comments (RFC) document in 1982. It is an electronic mail transport and delivery protocol used by mail transfer agents (MTA) that was primarily designed to reliably and efficiently relay email messages across different networks to ultimately deliver it to its destination.

Before an SMTP client is able to use the protocol, it must first determine the address of the SMTP server to which an email should be delivered. This is achieved by querying a domain name system (DNS) server for a mail exchanger (MX) record for the destination domain of the email. For example, if sophia@example.com has been specified as the recipient of an email, the SMTP client would first query a DNS server for an MX record belonging to the example.com domain.

An SMTP server can be either the ultimate destination of the email, an intermediate relay or a gateway. If the server is the ultimate destination, it stores a copy of the message received from the SMTP client to the file system. The message can then be later retrieved by a MUA using a protocol such as the Internet Message Access Protocol (IMAP) or Post Office Protocol (POP). If the server is a relay, it does not store the message. Instead, it connects to another SMTP server to transfer the incoming message from the SMTP client to the second SMTP server. In other words, a relay is both an SMTP server to a MUA, and a SMTP client to another SMTP server. An SMTP gateway is similar to a relay. It accepts incoming messages through the SMTP protocol. However, it does not use SMTP for relaying the message to another server. It uses another protocol which is dependent on the network within which it resides.

In order to transfer a message to the SMTP server, the client has to perform a mail transaction. The client initiates the transaction by sending the MAIL command to the server. The command must include a value for the FROM parameter of for indicating a reverse path. The reverse path is the sender's email address and is used for

error reporting. Following the MAIL command, the client should issue one or more RCPT commands to specify the recipients of the message. The RCPT command takes a TO parameter which must contain the email address of a recipient. After specifying the recipients, the DATA command is used to specify the contents of the message. The content may span over multiple lines and should abide to the Internet Message Format defined in RFC5322 [2]. Once the contents has been specified, a line with a single period character should be sent to indicate the end of the data. At this point, the mail transaction is completed and the message is queued for processing.

2.3. Internet Message Access Protocol (IMAP)

The Internet Message Access Protocol (IMAP) [5] is a protocol designed by Mark Crispin for storing and retrieving email messages from a mail delivery agent (MDA). It has been published as a standard in a Request for Comments (RFC) document by the Internet Engineering Task Force (IETF). The specification has been updated multiple times, and the latest revision is known as IMAP4rev1 [6].

The protocol allows a mail user agent (MUA) to access and manipulate electronic mail message on a server. It does not provide a mean for sending email messages, which instead is handled by a mail transfer protocol such as Simple Mail Transfer Protocol (SMTP). The communication between the email server and the email client occurs as commands and responses. Each client command consists of the name of a function to perform followed by arguments which are optional, depending on the function requested. The client command should be compulsorily prefixed with an identifier, known as a tag, which must be generated by the email client.

The email server may supply one or more responses to a client's command. Responses provided are either tagged responses or untagged responses, and are also delimited by CRLF. Untagged responses begins with the token "*" and is followed by information relevant to the command sent by the client. An untagged response indicates that the execution of the command on the server is still in progress, however useful data concerning the latter is already available and is sent immediately to the email client. To indicate the completion of a command, the server sends a tagged response. The tagged response is prefixed with the tag that the client initially included in its command followed by the status the operation which can be either OK, NO or BAD. OK signifies a successful execution of the command, NO indicates a failure during the execution of the command and BAD indicates a protocol error, such as, sending an invalid command.

Email messages on the server are accessible by either specifying a Unique Identifier (UID) or a sequence number, both of which are characterised as attributes of the messages. Unique identifiers are 32-bit values which are assigned to each messages in the mailbox in an ascending fashion, starting with the value of 1. Unique identifiers can be used in conjunction with another value known as the unique identifier validity. The UID and UID validity value together forms a 64-bit value which can be used to uniquely identify a particular message in the mailbox at any given point in time. Therefore, the UID of a message is guaranteed to remain the same in between different sessions as long as the message exists in the mailbox.

Sequence numbers can also be used to identify a message on the server. However, its value for each message is dependent on the current state of the mailbox and is not persistent over time. The first message in the mailbox is always given a sequence number value of 1 and subsequent messages are assigned increasing sequence number values. Therefore, if there are N messages in the mailbox, the first message will always have a sequence number of 1 and last one will always have a sequence number of N. If the first message is deleted from the mailbox, all messages with sequence number greater than 1 will have their sequence number decremented by one. The message with sequence number of 2 will become 1, the one with sequence number of 3 will become 2, and so on. This is contrast with unique identifiers, where, if the first message is deleted, the UID of the remaining messages remains unchanged.

In addition to a unique identifier and a sequence number, each message have another attribute known as Flags. A flag gives an indication about the current state of a message. The IMAP specification defines several flags, amongst which are the \Seen, \Answered and \Deleted flag. The \Seen indicates that the message has already been previously read by the user and is automatically set when the user fetches an email message from the server. The \Answered is used to indicate that a reply has already been provided to the message. The \Deleted flag is set for messages that have been marked for deletion, but have not been removed from the mailbox yet. These messages are permanently removed from the mailbox when the client sends an EXPUNGE command to the server.

2.4. Timer Wheel Algorithm

In the through-mail feature, one important activity of the server is to monitor the transit time of a mail at an intermediary mailbox. Intermittent checking of the transit time is obviously a huge overhead on the server. We propose to use an existing algorithm known as the hierarchical timing wheel[7][8] to reduce this overhead significantly.

The timer wheel algorithm consists of four categories, each associated with a timeout range as shown in Fig. 2; 0 – 3, 4 – 15, and so on. Each category is further divided into four buckets.

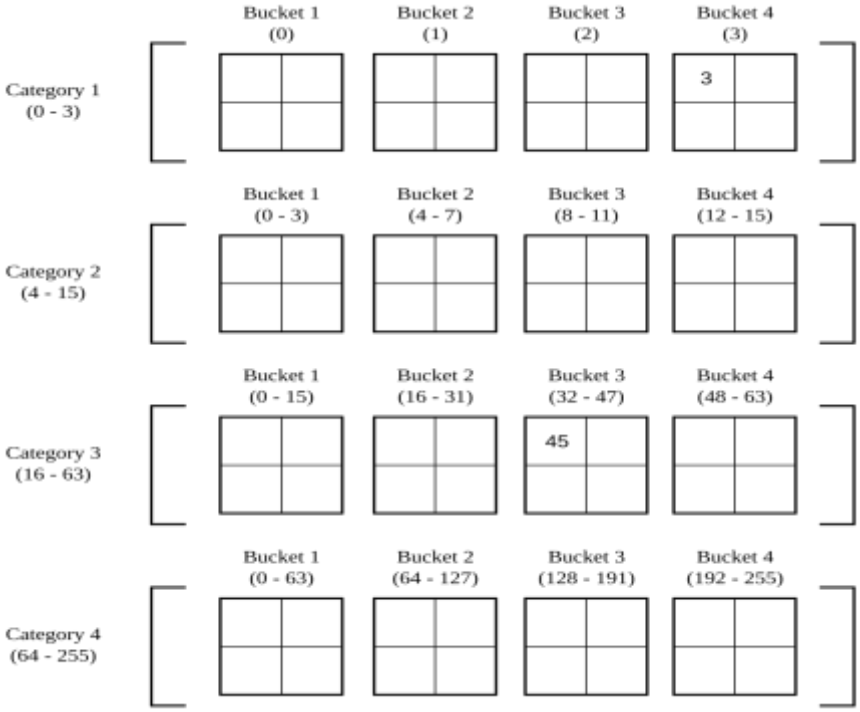

Fig. 2: Composing email user interface

In Fig. 2, category 1 refers to a timeout between 0 and 3. If a mail has a transit time of 3 minutes, then its timer is placed in bucket 4 of category 1. Likewise, if the transit time is 45 minutes, its timer is placed in bucket 3 of category 3.

Each category has a bucket pointer. Initially, all bucket pointers point to the first bucket of their respective category. At each clock tick (which is equivalent to 1 minute), the bucket pointer of the first category is advanced by one position to point to the next bucket. All timers located within the bucket are then expired.

This process continues until the bucket pointer of the first category reaches the last bucket. At this point, the bucket pointer wraps around and goes back to the first bucket. Then, all timers within the second category located within the bucket currently being pointed to by the bucket pointer of the second category are distributed amongst all buckets within the first category. The bucket pointer of the second category is then advanced forward by one position. This operation is known as cascading. It is recursive process and may propagate until the last category.

The strength of the timer wheel algorithm is that the server intermittently checks the timers of only the first bucket. If there is any timeout, the server provides a response. The timer wheel's insert operation is in the order of O(k), where k is the number of categories. Hence, insert operations takes a constant amount of time and is very fast. The timer wheel's remove operation, which involves removing a timer from the structure, is also constant and is in the order of O(1) since the timer can be directly removed from the structure using a pointer that points directly to it. On average, the timer wheel's expire operation is also in the order of O(1) since the bucket pointer of the first category simply needs to be incremented to expire further timers which represents only one operation. However, when the cascading is needed, the expire operation will take slightly more time. In the absolute worst case, the expire operation is in the order of O(N). The worse case represents an extremely rare situation. There is a very little chance of it occurring in practice. We can therefore conclude that, in general, the timer wheel is a very efficient and fast algorithm.

III. THROUGH-MAIL FEATURE

In this section, we describe a methodology for achieving the objectives of the **through-mail feature**. We design the GUI to be used in the email client and the processes running at the server, namely, receipt of a transit mail issued from the originator, handling of a response from an intermediary, and providing notification to intermediaries.

3.1. Email Client User Interface

The email client interface consist of two main views: (1) The Compose Mail View, and (2) The Transitbox View.

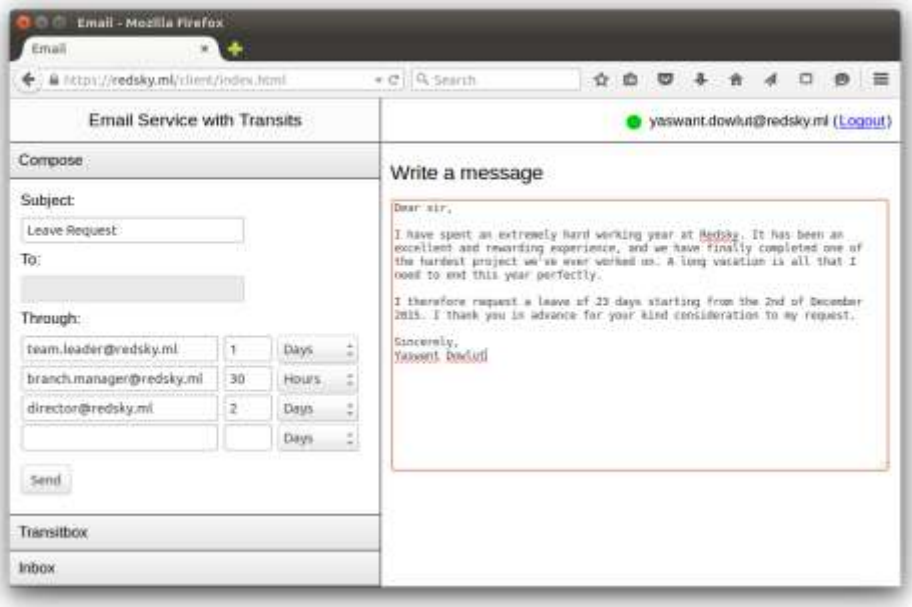

Fig. 3: Composing email user interface

The Compose Mail View (see Fig. 3) shall:

1. Disable the To field if the user inserts an entry in the Through field. The last entry in the Through field is the destinator.
2. Allow the user to specify the subject and the contents of the email.
3. Allow the user to specify the details for a list of intermediaries (any number), namely, the email address and the transit time. Each time the detail of an intermediary is inserted, an additional Through field is automatically generated for the next entry.

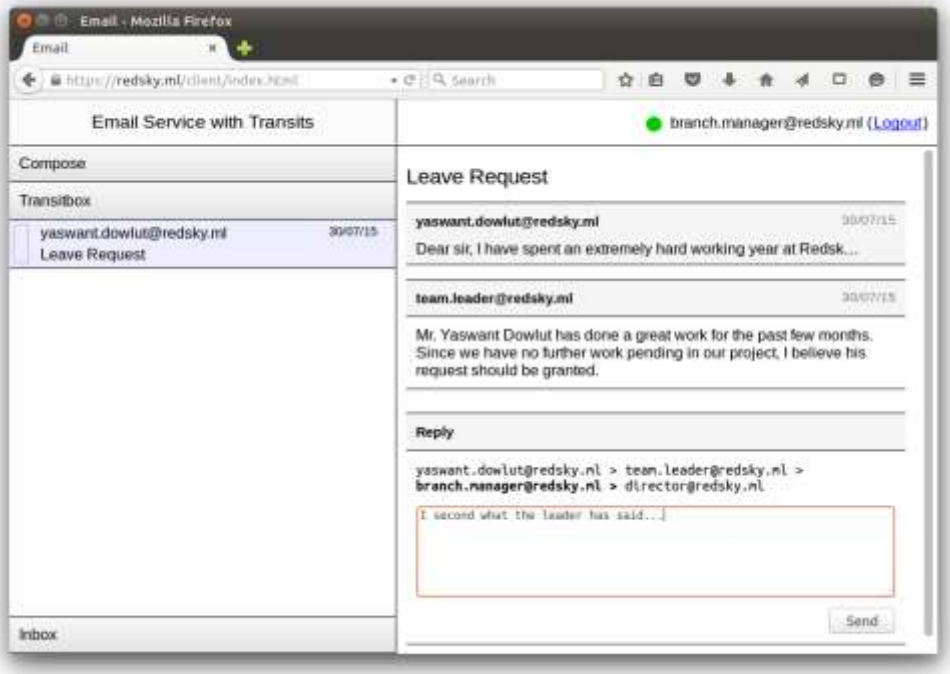

Fig. 4: Displaying transit conversation user interface

The Transitbox View (Fig. 4) of the email client:

1. Lists out, on the left, all transit mails received by the user.
2. Allows the user to select a transit mail and read its contents on the right. The content consists of address information and comments posted by each of the previous intermediaries in expandable and collapsable boxes. Note that the content posted by previous intermediaries are read-only.
3. Allows the user to place his own comments/feedback in an editable text area and click Send.

3.2. Handling a Transit Mail Composed by the Originator

When a user sends a new transit mail, the procedure which the system follows in order to deliver the message to the first intermediary is described in Fig. 5, which shows the components and their interactions, namely:

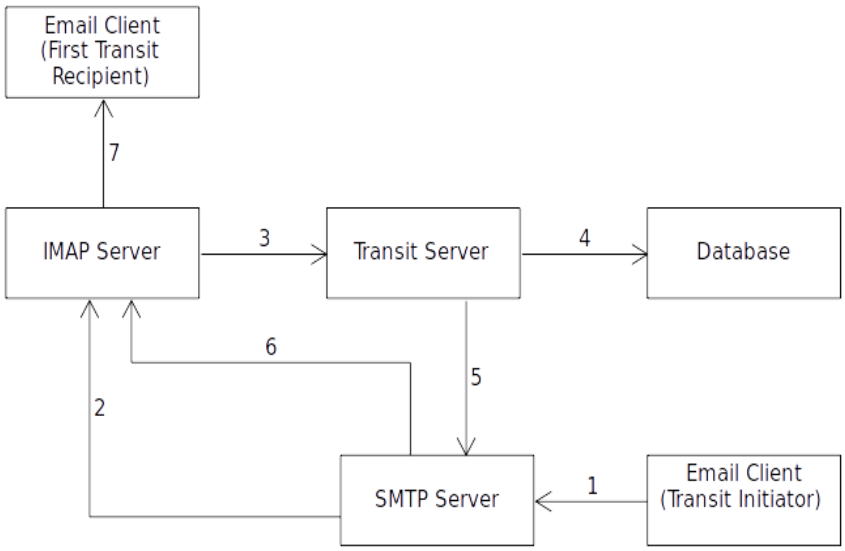

Fig. 5: Process of sending a new transit email

1. Before a transit email is sent, the user needs to have access to the web email client. After having successfully inserted his credentials into the system, the user logs into his mailbox through the email client. Then, the user have to access the compose interface to be able to write a message. When the user presses the send button, an email message is generated which contains the transit recipients specified by the user and their corresponding time limits within the X-Transit-Recipients and X-Transit-Time header fields respectively. The email would also contain the X-Transit-Type header field bearing "New" as its value to indicate the type of request it is. The email is then forwarded to the SMTP server for delivery to the transit server.

The following shows example values about the header fields. Note that the time limits are specified as minutes and the first time limit corresponds to the first recipient, the second one to the second recipient, and so on:
X-Transit-Type: New
X-Transit-Recipients: person.a@example.com; person.b@example.com; person.c@site.com
X-Transit-Time: 1440; 1440; 2880

2. When the SMTP server receives the email, it delivers it to the mailbox of the transit server which is located within the IMAP server.
3. The transit server then retrieves the transit message from its mailbox and extracts all the transit-related header fields from the latter. Through the X-Transit-Type header field, the transit server determines that the email corresponds to a request for creating a new transit message. It therefore extracts the list of email addresses of intermediaries and time limits from the X-Transit-Recipients and X-Transit-Time header fields respectively.
4. The transit server stores all the transit details in the database so that they can be accessed later. It also stores the message number that will be used in the message ID header field when forwarding the email. Furthermore, the transit server inserts a timer inside the timer wheel structure to monitor the transit time within which the first intermediary should reply. Particularities of the timer wheel structure has been described in the literature review section.

5. The transit server forwards the transit message to the first intermediary through the SMTP server. The transit server places a message ID value with a special format in the email that is forwarded. The message ID uses the following format:
   <Transit-ID>.<Message-Number>.transits@example.com
   Therefore, a message ID of "1.2.transits@example.com" would correspond to the transit with ID of 1 and message number of 2.
   The transit server uses this format to be able to track replies to the transit message. More specifically, if a user replies to the forwarded message, his email client would include the message ID of the forwarded message in the In-Reply-To header field. When the email then reaches the transit server, it will be able to determine to which transit record the response belongs to.
6. The SMTP server places the message into the mailbox of the first intermediary inside the IMAP server.
7. The transit message is then accessible to the first intermediary through his email client.

3.3. Handling a Response from an Intermediary

When the first intermediary has received a transit mail, he will be given a lapse of time to provide a response. Otherwise, the transit message is automatically forwarded to the next intermediary by the transit server. Fig. 6 depicts the steps the server takes to relay a mail from one mailbox to another:

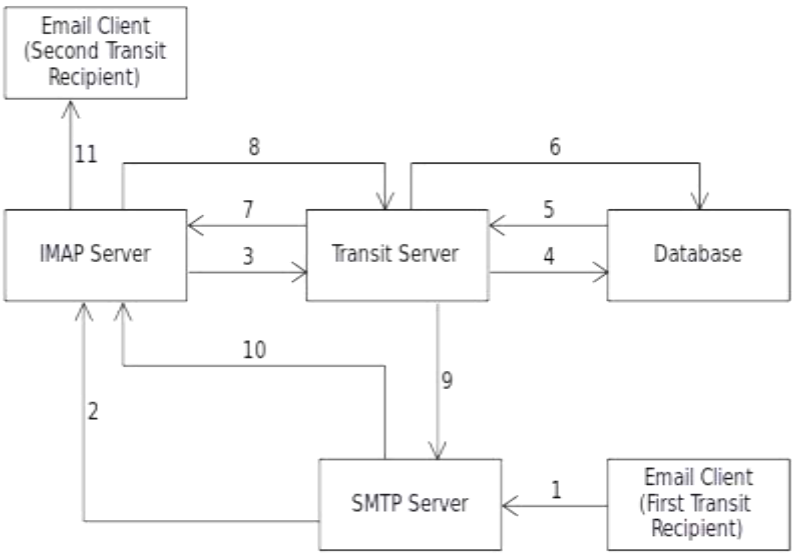

Fig. 6: Process of handling a transit response

1. To provide a response to a transit message, a user must open the latter in his email client and use the reply feature of his email client. Using the reply ensures that the message ID of the transit message gets included in the In-Reply-To header field when the email client sends the reply. The user input his reply and then the email client delivers it to the transit server through the SMTP server.
2. The SMTP server receives the email and places it into the transit server's mailbox within the IMAP server.
3. The transit server retrieves the email from the IMAP server and looks for transit-related fields to it. Since none were included by the email client, the transit server will not be able to find any. The transit server then falls back to analysing the In-Reply-To field of the email. If it finds a message ID in the special format defined in the previous section, it then assumes that the email is a response to a transit. It parses the message ID to determine to which transit the response belongs and begin the procedure for forwarding the response to the next intermediary of the transit which is described in the remaining steps.
4. The transit server stores the UID of the email containing the response in the database and fetches all information about the transit from the latter. It also removes the timer that was inserted in the timer wheel structure for monitoring the transit time of the previous intermediary.
5. The database delivers all information concerning the transit to the transit server. The information includes details such as the list of intermediaries, list of time limits and list of UID. The list of UID contains the UID of all emails which were sent as responses for the transit, including the UID of the email which initiated the transit. By forwarding these emails to the next intermediary, the next recipient would be able to see what everyone before him provided as response to the transit.
6. The transit server generates and stores message numbers which it will use within the message ID of emails

that it will dispatch afterwards.

7. The transit server fetches all emails in the list of UID from its mailbox on IMAP server.

8. The IMAP server provides the header and body of all emails requested in step 7. The transit modifies the message ID found in the header of each email to include one conforming to the special format defined earlier.

9. The transit server then forwards all the messages to the next intermediary through the SMTP server. The transit server also inserts a timer inside the timer wheel structure to monitor the time limit of the next intermediary.

10. The SMTP server then places the messages in the mailbox of the second recipient within the IMAP server.

11. The second recipient will then be able to see the transit message along with all responses of past recipients through his email client.

3.4. Notifications to Originator and Intermediaries

After a response to a transit email has been issued and the latter has been forwarded to the next intermediary, a notification is delivered to all previous intermediaries. Fig. 7 illustrates the steps involved in that process, namely:

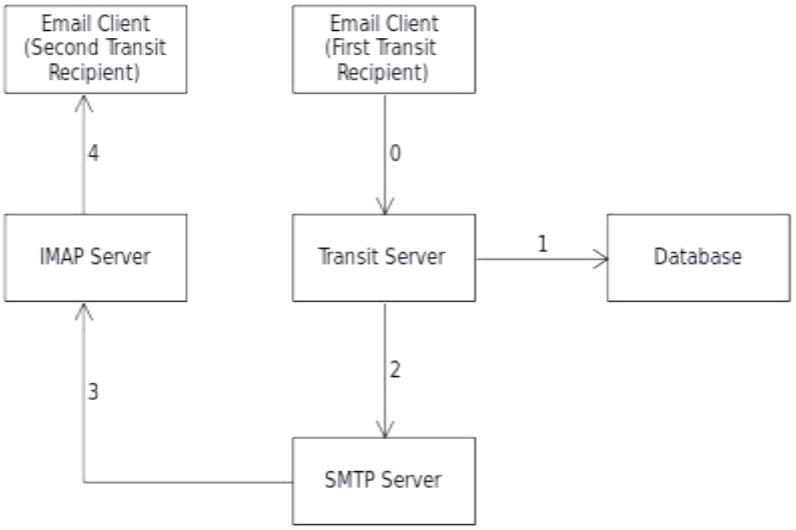

Fig. 7: Process of notifying recipients of response

0. The first intermediary of a transit message provides a response to the transit server. This step involves the whole process that was described in the previous section. This step has been included here to indicate that the process of sending notifications to recipients is a continuity of the process described in the previous section.

1. The transit server generates message numbers that will be included in the message ID of notification emails that will be dispatched. The message numbers are then stored in the database.

2. The transit server then send the notifications emails to all the previous intermediaries through the SMTP server.

3. The SMTP server places the notification email in the mailbox of the previous intermediaries within the IMAP server.

4. Then, using any email client, the previous intermediaries may view the notifications

IV. EVALUATION

The existing method of achieving email transiting is to use the traditional forward feature provided by many email clients. One drawback of this method is that any intermediary may change the content of the original message. Moreover, the intermediary recipients are free to forward it to whoever they want and thus may forward it to the wrong person or may not forward it at all. Lastly, the process can be cumbersome as the response of each intermediary recipient has to be concatenated in a single long email before forwarding.

In comparison to using contemporary email services, our proposed through-mail feature has the following strong points and weak points. The process of email transiting has been made significantly more convenient, temper-proof and traceable. It is more convenient as the user only need to specify the list of intermediaries and their transit times, and the rest is handled by the server. Morever, on a single screen, the user

is able to view the comments/feedback of any intermediary by click-opening his respective expandable box. Furthermore, in order to respond to a transit mail in his Transitbox, the intermediary only need to type in his comments and click Send. It is temper-proof as a receiver of a transit mail cannot alter the responses provided by earlier recipients. Finally, it is traceable since notifications are sent to the originator of the transit as well as to intermediaries through which the mail has gone through so far, thus allowing them to know exactly where the transit mail has reached.

On the other hand, our proposed email service has a few limitations. While not allowing intermediaries to change the route of a transit mail is a strong point, it can be also be a weak point since the originator may himself erroneously input the list of intermediaries. However, we believe it is on the onus of the user to insert the right intermediaries. Another limitation is that the through-mail feature requires a modified email client. However, the amount of modifications is minimal and can be easily implemented in existing email clients.

V. CONCLUSION

In this paper, we have implemented a through-mail feature by which a user may channel his request via a route of intermediaries in some order. The mail will reside in the transitbox of an intermediary for a transit time monitored by the transit server. Whether an intermediary responds within the transit time or not, the server channels the mail to the next intermediary and notify the previous intermediaries accordingly. Each intermediary may read the comments from past intermediaries as well as post his own. These points make the through-mail feature a desired feature in an email system for many organisations where there exist multiple levels of hierarchy.

To make the through-mail system complete, we may provide in the future a mechanism for allowing intermediaries to add attachments to their comments/feedback (documents and media files). The user interface of the email client should be updated accordingly to be able to demarcate the attachments of each intermediary from each other. Furthermore, the through-mail feature may be implemented in existing open source email clients so that access to it is readily available.

REFERENCES

[1]. D. Crocker, "Arpa Internet Text Messages", Internet Engineering Task Force (Ietf), 1982. [Online]. Available: Https://Tools.Ietf.Org/Html/Rfc822. [Accessed: 02- Jun- 2015].
[2]. P. Resnick, "Internet Message Format", Internet Engineering Task Force (Ietf), 2008. [Online]. Available: Https://Tools.Ietf.Org/Html/Rfc5322. [Accessed: 02- Jun- 2015].
[3]. N. Freed And N. Borenstein, "Multipurpose Internet Mail Extensions (Mime) Part One: Format Of Internet Message Bodies", Internet Engineering Task Force (Ietf), 1996. [Online]. Available: Https://Tools.Ietf.Org/Html/Rfc2045. [Accessed: 02- Jun- 2015].
[4]. J. Klensin, "Simple Mail Transfer Protocol", Internet Engineering Task Force (Ietf), 2001. [Online]. Available: Https://Tools.Ietf.Org/Html/Rfc2821. [Accessed: 04- Jun- 2015].
[5]. M. Crispin, "Interactive Mail Access Protocol - Version 2", Internet Engineering Task Force (Ietf), 1988. [Online]. Available: Https://Tools.Ietf.Org/Html/Rfc1064. [Accessed: 05- Jun- 2015].
[6]. M. Crispin, "Internet Message Access Protocol - Version 4rev1", Internet Engineering Task Force (Ietf), 2003. [Online]. Available: Https://Tools.Ietf.Org/Html/Rfc3501. [Accessed: 05- Jun- 2015].
[7]. Molnar, "Kernel/Timer.C Design", Lkml, 2005. [Online]. Available: Https://Lkml.Org/Lkml/2005/10/19/46. [Accessed: 06- Jun- 2015].
[8]. G. Varghese And T. Lauck, "Hashed And Hierarchical Timing Wheels: Data Structures For The Efficient Implementation Of A Timer Facility", Acm, 1987.