MESSAGE RECALL IN ELECTRONIC MAIL SYSTEMS

Extended Abstract

Hussein M. Abdel-Wahab
Old Dominion University

Mohamed G. Gouda
The University of Texas at Austin

Abstract

In electronic mail systems, most sent messages are replied to with replies that contain the original messages. This is wasteful because in many situations the original message is still in the sender's storage when the reply, containing a second copy of the message, arrives at the sender. In this paper, we propose a new class of electronic mail systems where every sent message is stored in the sender's storage for some time. If a message is replied to within a reasonable amount of time, then the reply does not contain the original message. Instead, the original message is appended to the reply after the reply arrives at the sender and before it is delivered to the user.

1. Introduction

A common scenario in electronic mail systems is as follows. One party sends a data message to a second party, then the second party replies back with a reply that contains the original message. This scenario is inefficient because the contents of the original message make a complete round trip: from the first party to the second and back to the first party.

Electronic mail systems can be made more efficient by employing a message recall protocol that avoids inefficient scenarios, similar to the one discussed above. A message recall protocol consists of three main steps.

i. When a party sends a message to another party, the first party maintains a copy of the message in its storage.

ii. When the second party receives the message and sends a reply to it, this party does not append the message to the sent reply.

iii. When the first party receives the reply, it retrieves the original message from its storage and appends it to the reply before delivering the reply to its user.

This message recall protocol may seem simple at first. However, a closer examination of the protocol reveals that it is in fact very complex. In this paper, we give a formal definition of this protocol.

The use of message recall protocols in broadcast systems is discussed earlier in [1] and [2]. However, this is the first time where message recall protocols are proposed as means to increase the efficiency of traditional electronic mail systems.

2. Overview of the Message Recall Protocol

We consider a system that consists of n user processes. Associated with each user process u[i] is a mail process m[i], where i is

in the range 0..n-1. Each mail process m[i] acts an agent for sending, receiving, and buffering data messages on behalf of its user process u[i].

For a user process u[i] to send a data message to another user process u[j], u[i] supplies its mail process m[i] with the message text and destination j. Process m[i] then constructs the data message and sends it to the mail process m[j]. Process m[j] keeps the message so that its user u[j] can read it, and possibly reply to it. The message reply, if any, is sent from m[j] to m[i] which keeps it so that its user u[i] can read it, and possibly reply to it. The new reply, if any, is sent from m[i] to m[j], and the cycle repeats.

Each mail process m[i] has two buffers. The first is an infinite buffer for storing all data messages received by m[i]. The second is a circular buffer for storing up to q data messages sent by m[i]. Received messages are stored in m[i] so that user u[i] can read and reply to any of them later. Sent messages are stored in m[i] so that when m[i] receives a reply to any previously sent message, m[i] can append the message to the received reply before storing the reply in the infinite buffer.

Data messages sent by m[i] are stored in the circular buffer of m[i] for at least S seconds. Thus, a reply to any of these messages should arrive at m[i] within S seconds from the instant the message is sent in order to ensure that the message is still in the circular buffer when the reply arrives. If another mail process m[j] receives a message from m[i] and sends a reply to it within R seconds from the instant the message is received, then S and R should satisfy the following relation.

$$S \geq R + d.(i, j)$$

where d.(i, j) is an upper bound on the time period, in seconds, needed for a data message to travel from m[i] to m[j], then for a reply message to travel from m[j] to m[i].

Note that period S is measured using a real-time clock in the mail process that sends the message, and period R is measured using a real-time clock in the mail process that sends the message reply. Therefore, the correctness of the above relation is based on the assumption that these two clocks have ths same rate.

Each data message sent in this system has five fields as follows.

data(tm, src, txt, ptm, rcl)

tm is the time instant at which the message is sent. It is called the inception time of the message. Because a process can send at most one message at a time, the inception time of a message can be used to uniquely identify the message in the message sender.

src is the index of the mail process which sent the message.

txt is the message text.

ptm is an integer. If this message is a reply to some previously received message g, then ptm is the inception time of g. Otherwise, ptm is 0, different from the inception time of any message.

rcl is a boolean value. If this message was stored in the sending mail process when it was sent, then rcl is true. Otherwise, rcl is false. Note that each mail process can store up to q sent messages at a time.

Therefore, some data messages may be sent without being stored.

3. Data Structures of the Protocol

Each mail process m[i] has the following six variables.

```
var
    tm  : integer,
    src : 0..n-1,
    txt : text,
    ptm : integer,
    rcl : boolean,
    dst : 0..n-1
```

The first five variables are used to store the fields of any message after it is received by m[i]. They are also used to store the fields of any message before it is sent by m[i]. The last variable is used to store the index of the destination process of a message before the message is sent by m[i].

Each m[i] also has an infinite buffer to store the fields of every message received by m[i]. This buffer consists of five arrays rtm, rsrc, rtxt, rrcl, and rrcv, along with a single variable r that acts as an index for the five arrays.

```
var
    rtm  : array [integer] of integer,
    rsrc : array [integer] of 0..n-1,
    rtxt : array [integer] of text,
    rrcl : array [integer] of boolean,
    rrcv : array [integer] of integer,
    r    : integer
```

After a data(tm, src, text, ptm, rcl) message is received by m[i], the four fields tm, src, text, and rcl of the message are assigned to the current locations of the first four arrays, namely rtm[r], rsrc[r], rtxt[r], and rcl[r], respectively. Then, rrcv[r] is assigned the current time at which the data message is received. Finally, index r is incremented by one.

Note that field ptm of a received message is not stored in the infinite buffer. This field is used to decide whether to append the text of a previously sent message to the text of the received message if the received message happens to be a reply to the sent message.

Each mail process m[i] also has a circular buffer to store the fields of up to q messages sent by m[i]. This buffer consists of two arrays stm and stxt along with a single variable s that acts as an index for the two arrays.

```
var
    stm  : array [0..q-1] of integer,
    stxt : array [0..q-1] of text,
    s    : 0..q-1
```

Before a data(tm, src, text, ptm, rcl) message is sent by m[i], process m[i] compares field tm of the message with the current value of stm[s]. If (tm - stm[s]) is bigger than S, then process m[i] recognizes that the message currently stored in location s is old and can be discarded. In this case, m[i] stores the two fields tm and txt of the new message in stm[s] and stxt[s], respectively, and assigns field rcl in the new message the value true. Otherwise, m[i] assigns field rcl in the new message the value false.

4. Actions of the Protocol

Each mail process m[i] has three actions. One action for receiving a data message, a second action for sending a data message that is not a reply to any previously received message, and a third action for sending a data message that is a reply to a previously received message. Next, we discuss each of these actions in detail.

The first action in process m[i] is defined as follows.

```
rcv data(tm, src, txt, ptm, rcl) -->
    rtm[r] := tm;
```

```
rsrc[r] := src;
rtxt[r] := txt;
if ptm = 0 --> skip
[] ptm ≠ 0 -->
    find k such that ptm = stm[k]
    rtxt[r] := (stxt[k] ; rtxt[r])
fi;
rrcl[r] := rcl;
rrcv[r] := <current time of m[i]>;
r := r + 1
```

This action becomes enabled for execution when there is an incoming data message waiting to be received by m[i]. When this action becomes enabled, process m[i] executes the action as follows. First, m[i] receives the incoming data message. Second, it stores the message fields in the corresponding r-th locations in the infinite buffer. Third, process m[i] assigns rrcv[r] its current time, the time at which the data message is received by m[i]. Finally, m[i] increments r by one.

The second action in process m[i] is defined as follows.

```
true -->
    {user u[i] sends a new message}
    tm := <current time of m[i]>;
    src := i;
    txt := <from user[i]>;
    ptm := 0;
    if tm - stm[s] > S -->
        stm[s] := tm;
        stxt[s] := txt;
        s := s +r 1
        rcl := true
    [] tm - stm[s] ≤ S -->
        rcl := false
    fi;
    dst := <from user[i]>;
    send data(tm,src,txt,ptm,rcl) to
    m[dst]
```

The guard of this action is true. This implies that this action can be executed anytime when user u[i] wants to send a data message that is not a reply to a previously received message.

To execute this action, process m[i] first computes the first four fields of the message. (Note that field ptm is assigned 0 because the message is not a reply to a previously received message.) Then, m[i] decides whether its circular buffer has room to store this message and computes field rcl of the message accordingly. Finally, m[i] computes the message destination dst, and sends the message to process m[dst].

The third (and last) action in process m[i] is defined as follows.

```
true -->
    {user u[i] replies to the k rcvd msg}
    k := <from user[i]>;
    tm := <current time of m[i]>;
    src := i;
    txt := <from user[i]>;
    if tm - rrcv[k] ≤ R ^ rrcl[k] -->
        ptm := rtm[k]
    [] tm - rrcv[k] > R ∨ ~rrcl[k] -->
        txt := (rtxt[k] ; txt);
        ptm := 0
    fi;
    if tm - stm[s] > S -->
        stm[s] := tm;
        stxt[s] := txt;
        s := s +r 1;
        rcl := true
    [] tm - stm[s] ≤ S -->
        rcl := false
    fi;
    dst := rsrc[k];
    send data(tm,src,txt,ptm,rcl) to m[dst]
```

References

[1] H. Abdel-Wahab and M. G. Gouda, "Recall Broadcast", Proc. of the International Conf. on Global Data Networks, Dec. 1993.

[2] H. Abdel-Wahab and M. G. Gouda, "Systems of Recall Broadcast", Submitted for Journal Publication, Oct. 1993.