Extending Real Time Mobile Collaboration Algorithms to Handle Membership Events in an Ad-Hoc Mobile Network

Sandy Citro, Jim McGovern, Caspar Ryan
School of Computer Science and Information Technology
RMIT University
Melbourne, VIC, Australia
scitro@cs.rmit.edu.au, jim.mcgovern@rmit.edu.au, caspar@cs.rmit.edu.au

Abstract—Real time mobile collaboration systems allow two or more geographically separated users to work on a shared document at the same time. There has been a large amount of work on consistency management for real-time collaboration in replicated architectures, however, the algorithms depend on fixed membership, and are not suited to ad-hoc mobile networks. This paper builds on the previous work of the authors by enhancing the algorithm to handle membership events. The proposed algorithm is built on point to point membership management, and uses a state map to handle multiple sites joining the session at the same time. It is able to manage all membership events while maintaining document consistency.

The algorithm has been implemented and evaluated across a number of mobile collaboration scenarios. The results show that the algorithm is able to handle membership events correctly with a relatively low impact on resource consumption.

I. INTRODUCTION

Real time collaboration systems allow two or more users, at different geographical locations, to work on a shared document at the same time. Users can participate in the collaboration as long as there are network paths that allow them to communicate with each other.

Real time collaboration systems can adopt a centralised or a replicated architecture. In a replicated architecture, each device holds a document copy and each device is responsible for managing the collaboration. This architecture is particularly attractive in mobile environments since, unlike the centralised architecture, it does not introduce a single point of failure and devices can continue participating in the collaboration when other devices are disconnected.

Several algorithms have been proposed for real time collaboration systems to ensure consistency among document replicas, for example dOPT [1], adOPTed [2], GOT [3], GOTO [4], SOCT2 [5], SOCT3 [6], SOCT4 [6], LICRA [7], and ORESTE [8]. However, some of those algorithms fail to maintain consistency of the document in some specific scenarios [9], while others rely on a dedicated server [6]. Citro et al. [10] proposed a consistency management algorithm that works in mobile environments. It does not rely on a dedicated server and minimizes the consumption of the device’s resources. However, like existing algorithms that work in wired networks, it assumes that the number of participants is fixed, participants join the collaboration in the beginning of the collaboration session, and participants leave the collaboration voluntarily by quitting the session.

Unfortunately, these membership behaviours do not often hold in wireless network environments, especially in mobile ad-hoc network environments. Mobile networks are characterised by fluctuating bandwidth, rapid and random movement of users, frequent disconnection, and ad-hoc network creation. Frequent disconnection means that the number of active participants fluctuates and messages are not always guaranteed to arrive at their intended destination (the problem of missing operations). Furthermore, a site (participant) can join the session any time during the session whenever it is within the transmission range of a current session participant (the problem of late-joining). Sites are arbitrarily and involuntarily disconnected due to the hostile characteristics of mobile environments, and some sites may join the collaboration after a session has been running for some time. These characteristics create challenges for real time collaboration in a mobile environment.

As an effort to handle the above challenges, SOCT2 [11] extends its consistency management algorithm by introducing techniques to support disconnection and reconnection of mobile sites. However, the solution has some limitations: it requires the site to broadcast sufficient notification to other sites before it disconnects and it has to contact a previously designated site when it is about to reconnect. Therefore it does not work well in the event of frequent involuntary disconnection. Consequently, unlike SOCT2, the proposed algorithm aims to handle both involuntary and voluntary disconnection as well as sites joining a currently running session. The proposed algorithm extends the consistency management algorithm proposed by Citro et al. [10] so as to ensure the replicas are consistent in the midst of the membership events.

This work is part of the research program of the Australian Telecommunications Cooperative Research Centre (ATCRC)
In the event of *missing operations*, the proposed algorithm allows a site to detect missed operations, and once detected, the site will send a request to the originator site to resend the missing operations. In the event of *late joining*, the proposed algorithm populates the new site with the latest state of one of the current participants and the algorithm allows other sites to detect this new site and adjust their states accordingly. A new technique called a *state map* is introduced to allow the algorithm to handle concurrent late joining.

The rest of the paper is structured as follows: Section II provides a brief review of the existing work on membership management. Section III provides a background on real-time mobile collaboration systems. Then the proposed algorithm is presented in Section IV. The proposed algorithm has been implemented and evaluated, and the simulation results and analysis are presented in Section V. Finally, Section VI concludes the paper and outlines some possible future work.

## II. Related Work

Some algorithms have been devised to ensure consistency in real-time collaboration applications that are based on replicated architecture [1]–[6]. Those algorithms implement immediate operation broadcast to promote concurrency and operational transformations to ensure consistency. However, most do not discuss the implication of missing operation and late joining as they assume the participants are fixed and the network is reliable. SOCT2 [11], however, introduces an algorithm to support disconnection and reconnection of mobile sites. When a site is about to disconnect, it executes the procedure *Disconnect* which broadcasts a message to all sites in order to announce the disconnection, disables broadcast and reception of operations and defines a pointer in the history to mark the beginning of offline collaboration. Unfortunately, in a mobile environment, a mobile site might not be aware if it is currently disconnected from other sites, therefore the *Disconnect* procedure might not have been executed. It also defines a *Reconnect* procedure to allow the reconnecting site to get the most updated document state. However, the algorithm requires the disconnected site to designate another site to be a server site, and this server site will have to send the current copy of the document with its current history. If mobile sites get disconnected frequently (which is a common case in mobile networks), the server site has to send the document copy from time to time which obviously will consume unnecessary bandwidth. Furthermore, this also introduces a single point of failure.

Some work has also been done to manage group membership outside the real-time collaboration context. Two major approaches to managing group membership in distributed system environments are application level membership services [12]–[16] which are collectively called Group Membership Services (GMS) [17], and network layer membership services, such as IP Multicast [18]. Both approaches attempt to organize the processes/sites into a group such that all active sites are those that belong to the group and all messages are exchanged among members of the group. IP Multicast manages the group membership in the network layer where each site needs to send messages to only one multicast address, whereas GMS works in the application layer where each site is aware of who the other members are and sends messages to a targeted group.

GMS provides multipoint-to-multipoint group communication by organizing processes/sites into groups, such that every site that needs to communicate with other group members, sends a message targeted to the group and GMS delivers the message to the group members [13]. A process becomes a group member by requesting to join the group and it can cease being a member by requesting to leave the group or by failing. A group membership service monitors any changes in the group membership and notifies all group members of the new membership configuration by delivering the updated *group view* of the current group membership (a list of the current group members, identified by their unique process identifiers). The notion of *view-synchronous* (or *view-atomic*) communication, a formulation of virtually synchronous communication developed in the ISIS system [19], [20], aims to guarantee the delivery of view notifications in the same order as multicast messages [17].

Unfortunately, GMS has some characteristics that are inapplicable to wireless networks, especially ad hoc networks. First, it requires sites to periodically broadcast a beacon to signify its presence to other sites. This requires adequate available bandwidth and good connectivity, requirements that are not necessarily satisfied in wireless networks. Second, it requires a membership server [13] or a site to act as a group leader [12]. This will only work if the presence of a dedicated server is guaranteed and if the connectivity between the server and the members is stable, which is not the common case in wireless networks, especially ad hoc wireless networks. Third, it is difficult to maintain a consistent view of group membership in asynchronous systems, especially in the presence of unannounced disconnections [12]. In fact, it is impossible to create a GMS that is always able to inform members of the correct state of the membership and to always agree on the views delivered to different members [21]. Fourth, the view agreement is only reached when the network becomes stable. When the network is unstable, the membership changes constantly, thus making it difficult to reflect the correct membership at any given time. In mobile network environments, the network might never become stable. Finally, GMS requires a notification service to monitor and detect site failures. However, it is impossible to distinguish a crashed process from one that is very slow or currently disconnected and Fischer et al. [22] show that any problem requiring "all correct processes" to take some action cannot be solved deterministically.

IP multicast, as a weak case of a group membership service [16], does not provide group members with the information about who the other members are. Each site needs to know one multicast address to send messages to. The main aim of IP multicast is to provide communication channel transparency to the members. IP multicast requires routers that are willing to forward multicast messages and are running
an IP multicast algorithm that ensures all desired destinations receive messages. Furthermore, these routers have to always be connected so that the packet can be reliably forwarded through to final destinations. Unfortunately, the presence of such routers is not necessarily guaranteed in mobile network environments. In ad hoc wireless network environments, each device would have to build a multicast tree that represents the current network topology so that each packet that is sent knows what path it has to take to reach all group members. If there are changes in router connectivity (changes in topology), all devices have to rebuild their multicast trees to represent the new topology. Due to mobility and frequent disconnection, the wireless network topology changes rapidly. This requires the devices to constantly monitor the changes in topology and to rebuild its tree every time the topology changes. If the topology changes very frequently, the mobile devices will expend resources and time building and rebuilding the tree.

Furthermore, while both GMS and IP Multicast are designed to ensure messages generated by one group member are communicated to all other group members, they are not designed to ensure all messages eventually arrive at all group members. When a group member is disconnected indefinitely, it will be excluded from the group and may miss some messages. When this group member rejoins the group, there is no mechanism to ensure that all the missing operations are sent to this particular member so as to maintain document consistency with other sites.

Therefore, based on the above discussion, an algorithm to handle dynamic membership issues in real-time mobile collaboration systems should satisfy the following minimum requirements.

- It must not require a dedicated server or a group leader or introduce any other single point of failure.
- Devices do not need to detect changes in network topology, thus dedicating resources to participating in the session.
- The algorithm must be able to handle both voluntary and involuntary disconnection.
- Devices need not constantly monitor and maintain the current view of who the other members are.
- All operations eventually arrive at all collaboration session participants, while maintaining document consistency across all sites.

III. REAL-TIME MOBILE COLLABORATION SYSTEMS

The membership algorithm described in this paper is integrated with the consistency management algorithm proposed by Citro et al. [10] so that the combined algorithm can handle membership problems while maintaining the document consistency. In a replicated architecture for real-time collaboration, participants modify their document replicas concurrently and the updates are sent to all other participants so as to ensure all participants see the updates in the shortest time possible. In order to ensure timeliness and document consistency, each site goes through five different phases in a collaboration session.

1) Local Operation Generation: Each site makes changes to the document replica by generating local operations. Local operations are applied immediately to the local replica to ensure responsiveness before being stored in the site’s operation history. Each operation is timestamped with a state vector used to determine the precedence and the concurrency of two operations.

2) Operation Broadcast: Each generated local operation is then broadcast to all other sites as a remote operation so that they can apply the update to their replicas.

3) Remote Operation Reception: Upon receiving a remote operation, the site has to ensure that the remote operation is causally ready before it is executed. An operation is causally ready if all operations that precede it have been executed, and this is determined by using the state vector technique [1].

4) Remote Operation Execution: When a remote operation arrives at a particular site, the site might have already executed some operations that are concurrent to it. To preserve the intention of the originating site, the remote operation has to be transformed into its variant to make sure the effect of the operation is the same as its original effect to the document replica of the originating site [1] and any conflict that occurs should be resolved accordingly. The remote operation is then executed and stored in the operation history.

5) History Trimming: All executed operations are stored in the history such that any incoming remote operation can be transformed appropriately with the concurrent operations in the history. As the collaboration session goes on, the history size increases. Therefore, all operations that are not needed anymore are discarded or garbage collected [23] from the history.

IV. MEMBERSHIP MANAGEMENT

This section outlines the two distinct membership events and explains the algorithm proposed to handle them in real-time collaboration systems.

A. Late Joining

In wireless network environments, especially in ad-hoc networks, two or more sites can start a session as soon as they are within the transmission range of each other and they do not need an established infrastructure to start communicating. Furthermore, a device can arbitrarily join a currently running session whenever it is within the transmission range of any of the current participants. This is a case of "late joining". It is assumed that a new site $S_i$ decides to join the session after contacting a current collaboration participant $S_j$. A correct membership algorithm has to ensure that $S_i$ is consistent with the other members such that it is able to correctly participate in the collaboration session and that all other participants know that there is a new site joining and therefore make necessary adjustments to accommodate the new site correctly. In other words, when a new site joins a currently running session, the algorithm has to ensure that the current participants and the
new participant can collaborate in the session correctly as if they have all been in the session from the start.

Due to concurrent operations and various membership events, each site may be in different states at the time $S_i$ decides to join the session. Each site maintains its own current state of the document replica and its own site state (i.e., state vector and logical clock), which are not necessarily the same as the other sites. This means that in order for a new site $S_i$ to participate in the session, $S_i$ has to: (1) acquire an updated document from the contacted site ($S_j$), (2) acquire the complete operations history of $S_j$ so that this new site is able to process any future incoming operation (i.e., correctly transform remote operations such that the user intentions are preserved to ensure document consistency), and (3) acquire the latest state of the contacted site $S_j$ so that the new site $S_i$ can start participating in the collaboration in the same state as $S_j$ after the joining process. If $S_i$ gets the document from $S_j$, $S_i$ has to acquire the operation history and other information from $S_j$. If $S_i$ gets the operation history from another site than $S_j$, site $S_i$ might not be able to correctly transform future remote operations, thus the replica held by $S_i$ will not be consistent with other replicas.

Therefore, whenever a new site $S_i$ joins a collaboration session via $S_j$, the following procedure has to be executed.

1) $S_j$ constructs a state message ($\sigma_{S_j}$) that consists of:
- the latest version of the replica held by $S_j$ ($D_{S_j}$),
- the operations history of $S_j$ ($H_{S_j}$),
- the list of current participants ($P_{S_j}$), and
- the current site state of $S_j$ that consists of the logical clock ($LC_{S_j}$) and the state vector of $S_j$ ($V_{S_j}$).

2) $S_j$ sends the state message to $S_i$.

3) Upon receiving the state message, $S_i$ replicates $S_j$ by populating itself with the information provided by the state message.

4) An element in the state vector associated with $S_i$ is added into the state vector of $S_i$ and $S_j$.

Consider the scenario in Figure 2 where site 8 and site 9 want to join the session concurrently through different participants. Following the above procedure, site 7 replicates itself to sites 8 and 3 to site 9. Before the join process, site 7 and site 3 have their state vector as $V_7 = \{V_7[0], V_7[1], ..., V_7[6]\}$ and $V_3 = \{V_3[0], V_3[1], ..., V_3[6]\}$ respectively. After the join process, site 7 will add an element in its state vector such that $V_7[7]$ represents site 8, and concurrently, site 3 will add an element in its state vector such that $V_3[7]$ represents site 9. This causes a conflict and the precedence and concurrency will not be able to be determined properly, and may cause document inconsistency. The core of the problem is that the site naively adds an element to the end of the state vector to represent a new site, where the other site might be doing the same thing concurrently for another new site.

Therefore, we propose the use of a data structure called a state map to ensure that the contacted site can add an element to its state map without causing conflict with other sites. A state map, similar to a state vector, is a data structure to store the number of executed operations. Each site is assigned a place in the map. A site is allocated a unique key, such as its IP address or its hostname, to represent it in the map. Similar to the state vector technique, each site maintains a state map ($M_i$), and every time a local operation is generated, it carries a state map ($M_{op}$) with the same values as the current state map of the generator site.

In Figure 2, when site 7 grants the 'join' request of site 8, site 7 will add 8 into its list of participants, and site 7 will also add an entry to its state map with the key of site 8 and zero as its initial value. At the same time, site 9 joins via site 3, thus site 3 will add an entry in its state map with the key of site 9. When the information of the two newly-joined sites is disseminated to all sites, there will be no conflict in the state map since the new sites are uniquely identified by
Their keys.

As illustrated in Figure 3, the state map of site 7 before the join process is
\[ M_7 = \{ \text{key}(S_1) \Rightarrow \text{val}_1, \text{key}(S_2) \Rightarrow \text{val}_2, ... \text{key}(S_i) \Rightarrow \text{val}_i \} \]
where key(S_i) is the unique key of site i and val_i is the number of operations generated by site i that have been executed by site 7. After the join process, its state map becomes
\[ M_7 = \{ \text{key}(S_1) \Rightarrow \text{val}_1, \text{key}(S_2) \Rightarrow \text{val}_2, ... \text{key}(S_i) \Rightarrow \text{val}_i, \text{key}(S_{i+1}) \Rightarrow \text{val}_{i+1} \} \]
Concurrently, after site 9 joins through site 3, the state map becomes
\[ M_3 = \{ \text{key}(S_1) \Rightarrow \text{val}_1, ... \text{key}(S_i) \Rightarrow \text{val}_i, \text{key}(S_{i+1}) \Rightarrow \text{val}_{i+1} \} \]
The new element of each state map is appended at the end of the state map, but they have different keys such that they do not need to be placed in a particular location in the state map. Eventually, both sites will have
\[ M = \{ \text{key}(S_1) \Rightarrow \text{val}_1, ... \text{key}(S_i) \Rightarrow \text{val}_i, \text{key}(S_{i+1}) \Rightarrow \text{val}_{i+1} \} \]
Therefore, the state map technique is able to handle multiple sites joining a session concurrently.

After a site joins a session, the other sites might not be aware that there is a new site in the session until they receive an operation that originates from either the new site or from another site that already knows about the new sites. For example, site 7 generates and broadcasts an operation op_i after site 8 joins the session. Operation op_i will bear a state map with an extra element associated with site 8 (M_{op_i}[key(S_8)]). When op_i arrives at site 4, site 4 must be able to determine that there is an element in the state map of op_i, M_{op_i}[key(S_8)], that is new to site 4. Site 4 will know that there is a new site, site 8 with key key(S_8), and add the new element into its state map (M_{op}[key(S_8)] = 0) as well as add site 8 into its participants list. From this point onwards, all operations that originate from site 4 will bear the new element associated with site 8 in their state maps.

Figure 4 outlines the procedures to handle late joining problem and accommodate concurrent late joins: receiving a join request from a new site (recv_join_req), receiving the state message from another site (recv_state_msg) and checking for a new site upon receiving a remote operation (recv_remote_op).

Every site maintains a state map instead of a state vector and each operation will carry a state map which is equal to the state map of the originator site when the operation was generated. The procedure that uses a state map to determine precedence and concurrency is modified as in Figure 5.

\begin{figure}[h]
\centering
\includegraphics[width=\textwidth]{fig3.png}
\caption{State map for late join.}
\end{figure}

\begin{figure}[h]
\centering
\includegraphics[width=\textwidth]{fig4.png}
\caption{Late join procedure}
\end{figure}

**B. Missing Operations**

Events such as user mobility, inadequate available bandwidth or device failure may cause disconnection in wireless networks. It is often difficult, if not impossible, to determine the cause of the disconnection. There is no way to distinguish whether the disconnection is caused by a device being outside the transmission range, insufficient bandwidth, or the device having crashed.

When a site S_0 is disconnected, it stops receiving messages from other devices. Site S_0 is not able to broadcast messages to other devices either. In a collaboration session, this would mean that the disconnected site S_0 will miss some operations that are being broadcast while it is disconnected. This is a case of "missing operations". Not only will site S_0 miss operations while it is disconnected, the other devices also miss operations that site S_0 generates during its disconnection period. Therefore, the algorithm must ensure that an operation generated by one site will eventually arrive at all participants.
When $S_i$ receives an operation $op$ from $S_j$ after being disconnected, $S_i$ might not be able to deliver or execute $op$ immediately because there are some operations that precede $op$ that were sent while $S_i$ was disconnected. Therefore, $S_i$ needs to identify the operations that it has missed during its disconnection period and it has to request those operations to be re-sent to it.

A remote operation $op$ that arrives at $S_i$ carries a state map that describes the state of the originating site when $op$ was generated. By comparing the state map of $S_i$ ($M_{S_i}$) and the state map of the received operation ($M_{op}$), $S_i$ will be able to detect whether there are missing operations that it needs to request. $M_{S_i}[key(S_j)]$ signifies the number of operations originated from $S_j$ that have already been executed by $S_i$, and $M_{op}[key(S_j)]$ signifies the number of operations originated from $S_j$ that have already been executed by $S_j$ just before $op$ was generated. By comparing $M_{S_i}[key(S_j)]$ and $M_{op}[key(S_j)]$, site $S_i$ is able to deduce that it has missed some operations from $S_j$. This situation also applies when site $S_j$ receives an operation op from site $S_i$. By comparing $M_{S_i}[key(S_j)]$ and $M_{op}[key(S_j)]$, site $S_j$ knows what operations that it has not received from $S_i$.

Consider the scenario illustrated in Figure 6 where $S_i$ is disconnected right after receiving $op_1$ from $S_j$. During disconnection, $S_j$ generates $op_2$ and $op_3$ that are not received by $S_i$. When $S_i$ is reconnected, $op_4$ arrives at $S_i$, with $M_{op_4} = \{s_1 \Rightarrow 0, s_2 \Rightarrow 3\}$. From the state map, $S_i$ determines that $S_j$ has generated three operations before $op_4$ ($M_{op_4}[S_j] = 3$). Since $S_i$ has already executed one operation from site 2 ($M_{S_i}[S_j] = 1$), $S_i$ requests $S_j$ to send two operations generated before $op_4$, by sending a tuple <S1, 1, 2>. When $S_j$ receives this request, $S_j$ will send operations that satisfy $\forall op_i : (S_{op_i} = S_j) \land (1 \leq M_{op_i}[S_j] \leq 2)$.

In order to detect missing operations, a site needs to compare its state map with the state map of the incoming operation whenever a remote operation is received. The remote_op_reception is modified as in Figure 7.

When $S_j$ sends an operation request to $S_i$, $S_j$ has to process the request and eventually sends all the operations that site $S_i$ has requested. Site $S_j$ traverses through its operations history and sends the matching operations. However, operations in the history of $S_j$ might have been transformed, and may not be in their original form. To maintain the document consistency, the operations that are going to be sent have to be in their original form as if they are being sent when they were freshly generated by the originator. Therefore, the operation history needs to store operations in their original form in addition to their transformed variants. In the case of a simple text editor, where the operations are simply insert and delete, storing the original form of the operation means simply storing the original position of the inserted or deleted character. For other types of applications, the history might need to store the whole original form of the operation. Figure 8 outlines the procedure receive_op_request executed by every site $S_i$ to process an operation request message. When the requested operations arrive at site $S_i$, $S_i$ will treat them as ordinary remote operations by executing the remote_op_reception procedure to resume the collaboration as usual.

C. Summary

The impact of each membership event has been discussed and solutions to each event have been proposed such that the collaboration session can continue regardless of the changes in the membership. The disconnected site is able to resume collaboration when reconnected and a late join site is able to participate in the session as if it has been there since the beginning.

In the late joining case, a site must invoke a join request procedure to join the session via one of the current participants. The contacted site will send its latest state to the new site. After the new site receives the state and applies it, it can participate in the session (Figure 4). In the disconnection case, a site must ensure that it gets all the operations that it missed during the disconnection and all other sites must ensure they get all the operations that they missed from the disconnected
void rev_remote_op(<op, LCop, Mop, Sop>)
{
    /* check for any new site */
    for (all k in keysOf(Mop))
    {
        /* check if this site has element k */
        if (Mop[k] == 0) {
            /* add a new element with */
            /* key k and value 0 */
            Mop = Mop + {k => 0};
        }
        /* add the new site to participants list */
        PS = PS + {Sk};
    }
}
/* compare state maps */
if (Mop[Sop] < Mop[Sop])
{
    /* request op from Sop */
    send_op_req(Sop, <S, Mop[Sop], Mop[Sop] - 1>);
}
/* causal reception */
wait until Mop[k] > Mop[k], (k ∈ keysOf(Mop));
exec_remote_op(<op, LCop, Mop, Sop>);

Fig. 7. Remote operation reception procedure.

void op_req(<Si, x, y>)
{
    /* find ops in Hs that satisfy */
    /* ∀ op : (Sopi = S) ∧ (x ≤ Mopi[S] ≤ y) */
    for i = 1 to n do
    {
        <op, LCopi, Mopi, Sopi> = Hs[i];
        if (Sopi = S ∧ x ≤ Mopi[S] ≤ y) then
            /* send the operations to Si */
            send(Si, <op, LCopi, Mopi, Sopi>);
        end;
    endfor;
}

Fig. 8. Operation request reception procedure.

However, the simulation does show the resource usage of the additional functions relative to the existing consistency management algorithm without the membership functions and the simulation gives the trend of the expected results on various disconnection rates.

Firstly, the number of lost operations, the number of operation requests, and the total number of messages required to completely send all operations to all destinations are counted under various disconnection rates in order to determine the ratio of each of them against the total number of operations that have to be broadcast to all destinations. The number of lost operations is the number of operations that are sent by a site and lost during transmission, whereas the number of operation requests is the number of request sent by a site because it misses some operations.

Figure 9 shows the ratio of lost operations over the total number of operations sent. As expected, the disconnection rate increases, the number of lost operations increases exponentially. This is because some operation requests and resent operations might get lost during transmission causing the receiver site to send additional operation requests and the number of lost operations increases exponentially.

Figure 10 shows that, again as expected, the number of operation requests increases as the disconnection rate increases. For example, when two sites are participating and the disconnection rate is 30%, the number of operation requests is up to 50% of the number of sent operations. Figure 11 shows the correlation between the number of sent messages and the disconnection rate. The number of sent messages is the average number of messages being transmitted by a site in order to completely receive all operations. This number includes the number of operations sent in the first place, the number of operations being re-sent (due to disconnections), and the number of operation requests. When the disconnection rate is 30%, the number of total messages transmitted reaches up to 200% of the number of operations needed to be transmitted.

Secondly, in order to add membership functionality, the proposed algorithm requires additional overhead in terms of processing time. Before actually executing the remote operation, a site needs extra processing time that comprises: (1) time

V. PERFORMANCE ANALYSIS

The algorithm has been implemented and tested, by simulation, on various numbers of sites, numbers of generated operations, network delays and disconnection rates. The simulation aims to evaluate the resource usage of the introduced functions (handling missing operation and late joining) by measuring the portion of processing time used for handling membership events. The simulation is done using Java and a PC which simulates a number of sites, generating and receiving operations with various simulated network delays and disconnection rate. A simulation in a real mobile network environment is not necessary since the simulation does not aim to represent the actual time taken to run those function and the actual disconnection rate since they vary depending on the platforms and network configurations, the different ways of implementations, users behaviour, and users mobility.

site (Figure 7).

Authorized licensed use limited to: RMIT University. Downloaded on January 5, 2010 at 22:12 from IEEE Xplore. Restrictions apply.
to check whether or not there is any new site, (2) time to check whether or not the incoming operation has been received or executed, (3) time to check if there are any missing operations, (4) time to construct an operation request for any missing operations if there are any missing operations, and (5) time to put the operation request to the outgoing message queue ready to be sent. Figure 12 shows the proportion of total time taken to handle the membership management processing overhead.

When there are two sites collaborating, around 25 - 35% of the time is used for handling membership while the remaining 65-75% is used to actually execute operations. When there are more than 8 sites collaborating, the site requires less than 5% of the time to handle membership events. As the number of sites increases, the portion of time used for performing membership functions declines exponentially. This is caused by the fact that the more sites the more time is taken to execute the operation (executing an operation involves tracing the history to find concurrent operations and to do the necessary operational transformations).

The experiment shows that the time used by the proposed algorithm to perform the membership functions is constant regardless of the number of participating sites or the number of generated operations. In the simulation using a PC with 1.8GHz of CPU clock and 512MB of RAM, the time to execute a remote operation ranges from 1.42ms for two sites to around 45ms for 15 sites, whereas the time used for these membership functions is relatively constant around 0.23ms to 0.5ms.

VI. CONCLUSION

This paper has discussed the need to handle membership events in a fully distributed real time mobile collaboration system. Frequent disconnections and late joinings cause fluctuations in collaboration session membership. If this is not handled properly, it may result in inconsistent document replicas among the collaborating sites.

An algorithm that handles these two major membership events in mobile networks has been proposed and it is integrated with a consistency management algorithm proposed by Citro et al. [10] so as to ensure the consistency among document replicas. The proposed algorithm detects if there are any missing operations and requests the originator site to resend them, and allows a new joining site to blend in the session smoothly and be able to participate in the session. The algorithm incorporates a novel technique, called a state map, as a replacement of the state vector technique to allow concurrent late joinings to be handled correctly by the algorithm. The algorithm does not require new participants to explicitly inform other participants that they have joined the session. The experiments show that the proposed algorithm provides each collaborating site with the capability of handling various membership events while maintaining document consistency with a relatively small overhead (5%-25% of the total processing time).

Future work may include an investigation on how to optimize the algorithm in terms of minimizing the number of sent messages in the event of disconnections. Some possible strategies may include sending the missing operations in one message and dividing the document into partitions so as to reduce the overall number of messages needed to be transferred. Group membership events might also be analysed to identify patterns of behaviour that can be incorporated into the algorithm to help the algorithm anticipate membership events and to improve the overall performance.
REFERENCES


