An Efficient Algorithm for Maintaining Consistent Group Membership in Ad Hoc Networks

Pushkar Pradhan and Abdelsalam (Sumi) Helal

Computer and Information Science and Engineering Department
University of Florida, Gainesville, FL 32611, USA
helal@cise.ufl.edu

Keywords: *Ad hoc networks, Group Membership, View Formation, Algorithms, Distributed Computing, Peer to Peer Computing.*

1. Introduction

Most existing group membership protocols such as Abbadi et al's View Formation (VF) protocol assume that the network infrastructure is static and reliable. This, however, is not the case in ad hoc networks. VF makes an excessive use of aborts to deal with simultaneous attempts to change the group configuration with the objective of guaranteeing consistency, forcing simultaneous coordinating hosts to restart the protocol to propagate the change in the group view. This leads to an excessive generation of messages, draining the battery power of mobile hosts in the ad hoc case.

We have identified an opportunity to improve on VF and have modified it to reduce the unnecessary aborts, thus reducing the number of messages generated and hence saving battery power. We achieve this savings without sacrificing the consistency of the group views. We present our algorithm and briefly summarize related work. We also give a proof of correctness for our algorithm and present some experimental results.

2. Problem Definition

The group membership maintenance problem is defined as the requirement for each host in the group to have knowledge of what other hosts are members of its group (i.e. the list of currently active and connected members of the group), and for such knowledge to be consistent across the entire group at all times [6].

The problem we are dealing with here is to model a reliable group membership service that is aware of the peculiarities of ad hoc networks, i.e. absence of infrastructure, node host failures, limited power, and the inevitable and continuous event of network partitions.

3. Related Work

Abbadi, Skeen and Cristian in their paper [2] have explained the Virtual Partitioning algorithm that includes a view formation protocol for maintaining consistent group membership in distributed systems. Their algorithm makes use of a voting scheme to serialize joins and departs. It can be summarized as follows:

Whenever a host A detects a change in the group configuration, it initiates a two-phase protocol. In the first phase, A sends out a JOIN-VIEW message containing a new View ID (VID) to each site in its current view and waits for votes to accept its invitation. A site receiving the JOIN-VIEW will vote YES if its present VID is less than the VID in the JOIN-VIEW and if it has not already received an invitation to join a view with a higher numbered VID. The site will accept the invitation by sending a positive acknowledgment or reject it by sending a negative acknowledgement or nothing at all. In the latter case, coordinator A will recognize negative acknowledgments by detecting timeouts. In the second phase, A can proceed in one of two possible ways. If any negative acknowledgements have been received, it may abort the creation of the new view, either by sending explicit ABORT messages or by not sending any messages at all. If this occurs, A may attempt to restart the protocol using a greater VID hoping to convince more sites to accept its invitation. Alternatively, A may ignore any rejected invitations and form a new view consisting only of the set of sites v' $(v' \subseteq v)$ that accepted the invitation. It does this by sending a VIEW-FORMED message to each site in v' and attaching the set v' to that message. A site accepts the VIEW-FORMED message and assigns itself the view v only if it has not, in the meantime, accepted an invitation to join a higher numbered view. Any site that adopts v' as its view, adopts the new VID as its present VID. It need not acknowledge the VIEW-FORMED message.

Kenneth P. Birman and his colleagues at Cornell University introduced several new concepts in group communication for distributed systems. They have developed two distributed systems: ISIS [9], and its successor Horus [12]. Group management in ISIS is similar to a coordinating processor executing a Three-Phase commit [3] to atomically install new group views. This scheme has been presented by Ricciardi and Birman [1]. In normal circumstances, a centralized entity known as the coordinator repeatedly detects failures, computes new group views and installs them. When the coordinator goes down, a new coordinator needs to be elected to install new views. Since the installation of the previous view may have progressed halfway, the new coordinator must poll processors to get their opinions about what view must be installed.

Roman, Huang and Hazemi from the Department of Computer Science at Washington University have done considerable work in the area of consistent group membership in ad hoc networks [6]. Their protocol works in the following way.

The group is assumed to have a leader whose main function is to serialize all configuration changes. When an isolated host *u* discovers a neighboring host *v* already in the group G, it asks *v* about the identity of the group leader. Once *u* determines the identity of leader *l*, it can communicate directly with it. Once *l* has decided to admit *u* into the group, it informs *u* and all the members of the group about the configuration change by sending them a *join message*. When a host receives a *join message*, it stops transmitting regular messages and sends a *flush message* to all other hosts within the group indicating that it knows about the configuration change. Once s host receives a *join message* from the leader and *flush messages* from all the members of the group, it adds the new host *u* to the list of group members i.e., commits to the new group configuration, which includes host *u* as a member, and resumes sending regular messages. The leader will also wait to receive *flush messages* from all the members of the group before updating its group membership list. FIFO communication channels have been assumed, therefore receiving the *flush message* on a particular link guarantees that there are no more messages in transit on that link that may have been sent in a previous group configuration. Mobile host *u* has now successfully joined the group G and all the members of the group have the same view of the group.

3. Our Approach

It is our observation that the limitation of the Virtual Partitioning Algorithm's View Formation protocol is that when two or more different hosts initiate the view formation protocol at the same time, the ones with lower VIDs will have to abort.

This abort is superfluous since JOIN-VIEW messages can be buffered. In our approach, we have used the concept of a *sorted list* to buffer JOIN-VIEW messages**.**

When a mobile host A detects a change in the group configuration, it initiates a two-phase protocol. It sends a JOIN-VIEW message containing a new VID to all sites in its view. When a site B receives the JOIN-VIEW, it accepts A's invitation to join the new view by sending a YES vote if its present VID is less than the new one, and adds the new VID to its sorted list. Otherwise, it rejects the invitation by not responding at all.

After receiving YES votes, A sends out a VIEW-COMMIT message to all sites in its view. These sites buffer the VIEW-COMMIT message until its corresponding VID rises to the top of its sorted list. Then the VIEW-COMMIT is accepted and the VID deleted from the list. If

VIEW-COMMITS are not received within a particular timeout period, the corresponding JOIN-VIEW expires and is deleted from the list.

The scheme for VID is the same as the one suggested by Abbadi et al. [2]. To guarantee network-wide uniqueness, VIDs are pairs (c, s) where s is the site identifier and c is a counter stored at s and incremented each time s tries to create a new view. Thus VIDs created by different sites differ in the second component, while VIDS created by the same site differ in the first component. Pairs are compared in the following way: (c, s) < (c', s') if c<c', or c = c' and s < s' lexicographically.

If two different hosts become coordinators at the same time, both will send their JOIN-VIEW messages at roughly the same time. The order in which hosts receive these messages could be different. However, since we are using a sorted list and since all JOIN-VIEWs are guaranteed to have globally unique VIDs, they will be stored in the same order in the hosts' sorted lists. When the corresponding view commits arrive, they will be processed in exactly the same order. Hence we have serialized each configuration change.

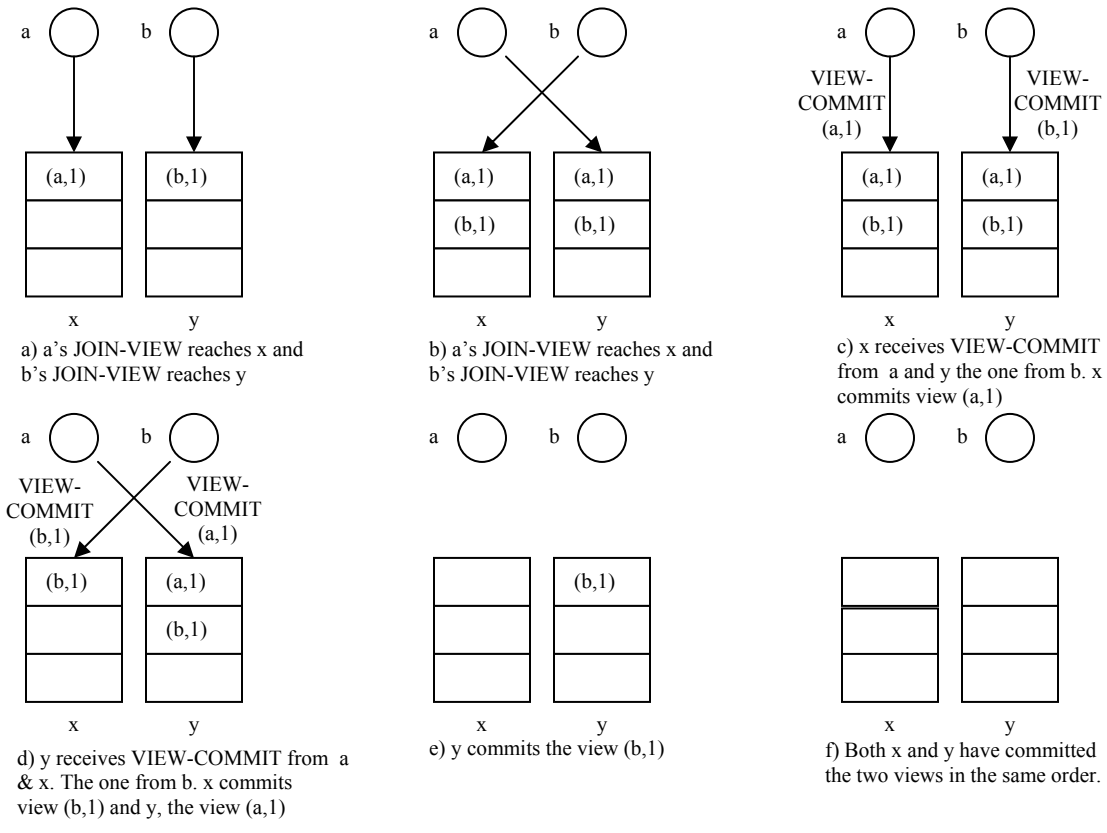

Fig 1. How simultaneous configuration change attempts succeed.

The decision of when the protocol must be run is left up to the application. A site may run the protocol to detect configuration changes if it suspects that a change in configuration has occurred. In the spirit of the peer-to-peer model, the level of group synchrony desired must be left to the application e.g. an advertising application will not fail if 10% of the hosts in its group do not receive their messages. However, in the scenarios such as players playing a game over the network, it is absolutely crucial for the application to detect when players join and leave the game, otherwise it will fail.

Fig .1 shows how unnecessary aborts are avoided. Let host *a* generate a VID (a,1) and *b* generate a VID (b,1) Consider two other hosts *x* and *y* where *x* receives *a*'s JOINVIEW and *y* receives *b*'s JOIN-VIEW first (Fig 1 a).Thus *x* has (a,1) at the top of its sorted list and *y* has (b,1) at the top of its sorted list. Node *x* sends its YES vote to *a* and *y* the corresponding YES vote to *b*. In the meanwhile, *a*'s JOIN-VIEW reaches *y* and *b*'s JOIN-VIEW reaches *x* (Fig 1b) Thus now *x* and *y* both have (a,1) at the top of their respective sorted lists. Node *x* sends its YES vote to *b* and *y* sends its YES vote to *a*. Hence both *a* and *b* have received YES votes from all other hosts including *x* and *y*. hence *a* and *b* both send their respective VIEW-COMMIT messages to all other hosts. Assume that *x* gets *a*'s VIEW-COMMIT first and *y* gets *b*'s VIEW-COMMIT first (Fig 1c). Node *x* checks the top of its sorted list, finds that (a,1) is the entry at the top and COMMITS the view with the ID (a,1). Node *y* examines the topmost element of its sorted list, finds (a,1) and thus buffers *b*'s VIEW-COMMIT. Node *x* deletes (a,1) from the top, hence allowing VID (b,1) emerge to the top. In the meantime, *a*'s VIEW-COMMIT reaches *y* and *b*'s VIEW-COMMIT reaches *x* (Fig 1d). Node *x* commits *b*'s view and *y* finds (a,1) at the top of its sorted list, hence it commits view (a,1), and allows (b,1) to rise to the top. Node *x* deletes (b,1) from the top of its list (Fig 1e). Node *y* now finds (b,1) at the top of its list, retrieves *b*'s buffered VIEW-COMMIT and commits the view (b,1). It then deletes VID (b,1) from the top of its list. Thus both *x* and *y* have committed *a*'s and *b*'s VIEWs in the same order (*a,b*) (fig 1f). Hence the view formation has been serialized.

We have shown how our protocol serializes configuration changes in the group by using the sorted list. Also, since we can buffer VIEW-COMMIT messages, concurrent attempts to propagate configuration changes need not result in coordinators with lower VIDs aborting. Since these types of aborts do not occur, the coordinator does not need to restart its protocol to propagate its configuration change to the rest of the group. Hence there is a considerable saving in the number of messages exchanged

4. Analysis

We now analyze our modified View Formation protocol in terms of the number of messages generated. The initial JOIN-VIEW message is sent to all hosts, thus accounting for n messages. Each host then replies by sending its YES votes, leading to n additional messages. After receiving the YES votes, the coordinator either sends a VIEW-COMMIT or an ABORT message to all hosts, giving n more messages. Thus for each run of the protocol, we have $n + n + n = 3n$ messages or $O(n)$ messages. Also since aborts during concurrent configuration change attempts do not occur, we have saved $3n$ messages for every extra run of the protocol for every simultaneous configuration change attempt.

5. Proof that the Protocol Avoids Unnecessary Aborts

Consider 2 hosts x_a and x_b becoming coordinators at the same time. Let $x_1,\ldots x_n$ represent all the hosts in the system (including x_a and x_b), where $name(x_a) < name(x_b)$

Assume the view ids $IDx_a < IDx_b$ (IDx_a is the ID of the JOIN-VIEW sent by x_a). Without loss of generality, assume JOIN-VIEWx_a (the JOIN-VIEW message sent by x_a) reaches $x_1,\ldots x_i$ first and JOIN-VIEWx_b reaches $x_{i+1},\ldots x_n$ first. $x_1,\ldots x_i$ receive JOIN-VIEWx_a and send their YES votes $Yax_1,..Yax_i$ (Yax_1 is the YES vote sent to x_a by x_1)

$x_{i+1},\ldots x_n$ receive JOIN-VIEWx_b and send their YES votes $Ybx_{i+1},..Ybx_n$. $x_1,\ldots x_i$ then receive JOIN-VIEWx_b and put IDx_b in their sorted list, sending YES votes Ybx_1 to Ybx_i. $x_{i+1},\ldots x_n$ receive JOIN-VIEWx_a and send YES votes $Yax_{i+1},\ldots Yax_n$. Thus x_a receives $Yax_{1,\ldots}Yax_n$ i.e. hears from all hosts within the group and then sends the VIEW-COMMITx_a to hosts $x_1,\ldots x_n$ (VIEW-COMMITx_a is the VIEW-COMMIT message sent by x_a). Therefore IDx_a gets deleted from the top of the lists of $x_{1,\ldots}x_n$, and IDx_b emerges to the top. x_b sends VIEW-COMMITx_b to $x_1,\ldots x_n$. Hence $x_1,\ldots x_n$ adopt IDx_b as their new view ID. Therefore x_a and x_b have both managed to commit their respective views.

Let $x_p,\ldots x_q$ be k arbitrary hosts selected from $x_1,\ldots x_n$. Let us assume that when $x_p,\ldots x_q$ become coordinators at the same time, all of them commit their views. Now, consider another host x_r where all $[name(x_p),\ldots name(x_q)] < name(x_r)$. Assume that $x_p,\ldots x_q$ get their JOIN-VIEWs to $x_1,\ldots x_i$ and x_r gets its JOIN-VIEWx_r to $x_{i+1,\ldots}x_n$ first.

$x_1,\ldots x_i$ will vote YES to $x_p,\ldots x_q$, and $IDx_p,\ldots IDx_q$ will be higher in the list than IDx_r. $x_p,\ldots x_q$ will then commit since we have assumed that they will. Then IDxr gets voted for by

sending $Yrx_1,\ldots Yrx_{i\ .\ } x_{i+1,\ldots}x_n$ will receive JOIN-VIEWx_r which they vote for by sending Yrx_{i+1} ,…Yrx_n. They then receive JOIN-VIEWs from $x_p,\ldots x_q$. They will then send their votes after which $x_{i+1,\ldots}x_n$ will cause $x_p,\ldots x_q$ to commit. Thus x_r receives $Yx_1,\ldots Yx_n$, thus it can send VIEW-COMMITx_r thus allowing x_r to commit. Thus assuming that k hosts can commit on becoming coordinators, we have proved that k+1 hosts can commit. Therefore, by induction we have proved that when hosts simultaneously become coordinators, all of them commit.

5. Performance

Three simulators were written in JAVA, one each for Roman-Huang-Hazemi's, the original View Formation protocol, and the modified View Formation protocol. Events such as joins or departs were read in from a configuration file. Hosts were simulated by individual Java threads. The spawning of each node was simulated by starting a new thread each time a START was read in from the configuration file. To stop a node, a flag was set so that it stopped responding to messages.

Joins were simulated as follows. A server, written in Java was used to keep track of all current active hosts, and a random number generator was used to allocate a different host at random as a coordinator every time a new host requested it to admit it to the group.

The time interval between two consecutive joins was set using a random number generator, which generated a number between 0 and x seconds. 3 sets of reading were taken, one each for the values 10, 20 and 30 seconds respectively as the values for x. The number of messages was counted by incrementing a global variable each time a datagram packet was sent by a host. The number of messages was noted for each run of the protocols. For each time interval set, a graph of the number of messages versus the number of nodes joining was plotted. The graphs that follow show the results obtained.

It has been observed each time, Roman Huang and Hazemi's protocol gave the maximum number of messages, and this number has been constant for each of the time intervals. The Virtual Partitioning Algorithm's View formation protocol gave comparatively less messages, but this number was still pretty large. This was due to the number of aborts that were generated when joins were attempted at roughly the same time. As the time interval between joins was increased, the number of messages decreased in most cases, due to fewer aborts. The modified View Formation protocol gave the least number of messages, and the number of messages also stayed constant no matter what interval was used.

Hence we can conclude that the modified version of the View Formation protocol saves us a considerable number of transmitted messages, thereby saving power.

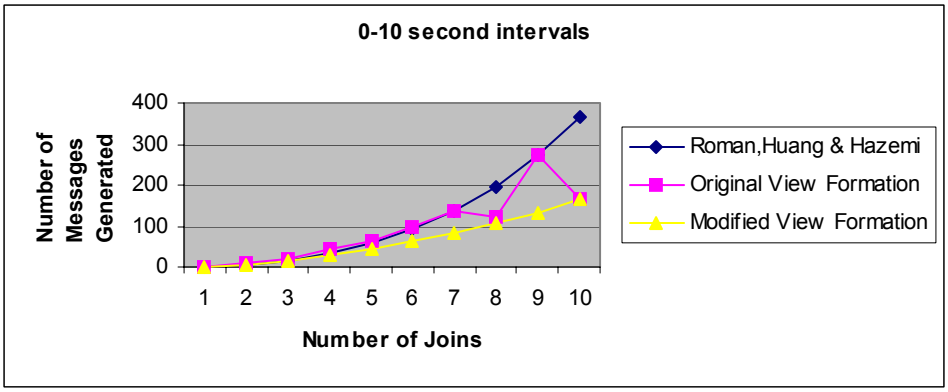

Fig. 2 Graph of Messages versus total Number of Join events for 0-10 second random intervals.

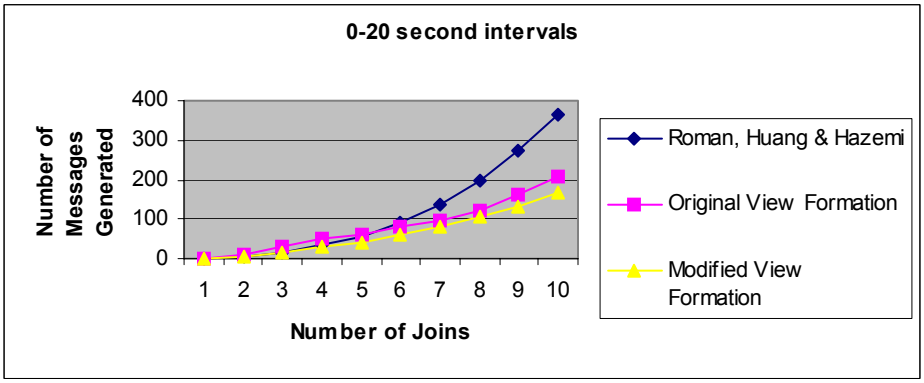

Fig. 3 Graph of Messages versus total Number of Join events for 0-20 second random intervals.

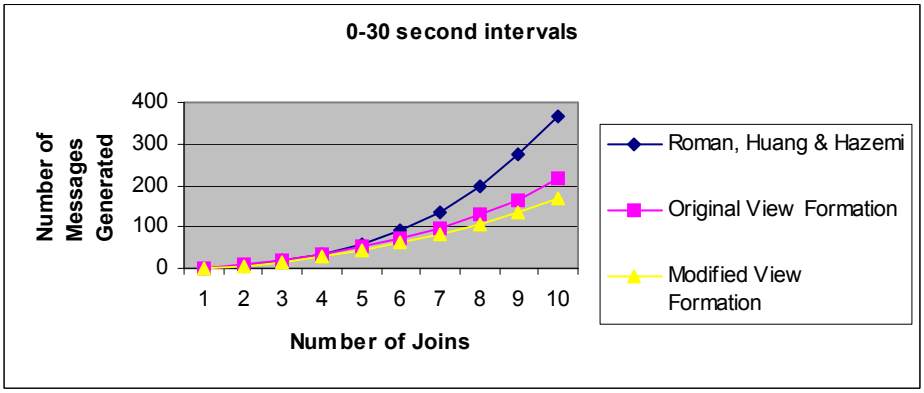

Fig. 4 Graph of Messages versus total Number of Join events for 0-30 second random intervals.

6. References

[1] A Ricciardi and Kenneth P. Birman “Process Membership in Asynchronous Environments” Technical Report TR 93-1328, Department of Computer Science, Cornell University, 1993.

[2] Amr El Abbadi, Dale Skeen, Flaviu Cristian: An Efficient, Fault-Tolerant Protocol for Replicated Data Management. PODS 1985: 215-229

[3] D. Skeen. “Crash Recovery in a Distributed Database System” Technical Report UCB/ERL M82/45, Electronics Research Laboratory, University of California at Berkeley 1982.

[4] D.D.E Long, J.-F. Paris, “Voting without version numbers” Performance, Computing, and Communications Conference, 1997. IPCCC 1997 , IEEE International , 1997 Page(s): 139 –145

[5] Gregory V. Chockler, Idit Keidar, and Roman Vitenberg "Group Communication Specifications: A Comprehensive Study. "ACM Computing Surveys 33(4), pages 1-43, December 2001.

[6] Gruia-Catalin Roman, Qingfeng Huang, Ali Hazemi:, "On maintaining Group Membership Data in Ad Hoc Networks," Washington University, St Louis, Technical Report wucs-00-26, April 16, 2000.

[7] Gruia-Catalin Roman, Qingfeng Huang, Ali Hazemi: “Consistent Group Membership in Ad Hoc Networks”. ICSE 2001: 381-388

[8] Jehan-François. Paris, D.D.E Long, “Efficient dynamic voting algorithms” Data Engineering, 1988. Proceedings. Fourth International Conference on , 1988 Page(s): 268 -275.

[9] K. Cho and K. Birman. “A Group Communication Approach for Mobile Computing MobileChannel: an ISIS Tool for Mobile Services.” May 1994. Cornell University's Computer Science Department Technical Report.

[10] Katherine Guo and Luis Rodrigues, “Dynamic Light-Weight Groups” Cornell University Technical Report, TR96-1612, October, 1996

[11] Kenneth Birman “The Process Group Approach to Reliable Distributed Computing”. Communications of the ACM, December, 1993, 37-53.

[12] Robbert van Renesse, Kenneth P. Birman and Silvano Maffeis, “Horus, a flexible Group Communication System”, Communications of the ACM, April 1996

[13] T.A. Joseph and Kenneth P. Birman “Low cost management of Replicated Data in Fault-Tolerant Distributed Systems” Proc. ACM SIGOPS Symp. On Operating System Principles, 4(1):54-70, 1986