EFFICIENT GROUP MEMBERSHIP ALGORITHM FOR AD HOC NETWORKS

By

PUSHKAR P. PRADHAN

A THESIS PRESENTED TO THE GRADUATE SCHOOL
OF THE UNIVERSITY OF FLORIDA IN PARTIAL FULFILLMENT
OF THE REQUIREMENTS FOR THE DEGREE OF
MASTER OF SCIENCE

UNIVERSITY OF FLORIDA

2002

I DEDICATE THIS THESIS TO MY PARENTS

ACKNOWLEDGMENTS

I take this opportunity to thank my advisor, Dr. Abdelsalam Helal, for his invaluable guidance and motivation, and for giving me the opportunity to work under him. I would also like to thank Dr Joachim Hammer and Dr. Michael Frank for serving on my committee. I would like to thank my parents, Pramod and Nayana Pradhan, for their encouragement and support. I would also like to thank my friend Nitin Desai for his advice, and my friends Ameya Deshmukh and Nishant Muley for helping with the proofreading of this thesis.

TABLE OF CONTENTS

page

LIST OF FIGURES

Abstract of Thesis Presented to the Graduate School
of the University of Florida in Partial Fulfillment of the
Requirements for the Degree of Master of Science

EFFICIENT GROUP MEMBERSHIP ALGORITHM FOR AD HOC NETWORKS
By

Pushkar P. Pradhan

December 2002

Chair: Dr. Abdelsalam Helal
Major Department: Computer and Information Science and Engineering

Designing a reliable group membership algorithm for mobile ad hoc networks is a non-trivial task since we must deal with limited device power, possibilities of frequent disconnections and network partitions. The function of the group membership service is to maintain a consistent group view across all members of the group. Only then can we say that the group membership service is reliable. Reliable group membership protocols in distributed systems (e.g., the virtual partitioning (VP) algorithm) make free use of aborts as a tactic to ensure the reliability of group formation. In this thesis, we show that such protocols are inappropriate in the ad hoc environment because they use aborts excessively. We present a group membership algorithm based on VP that minimizes message exchanges and thus reduces the power spent in transmission by detecting and eliminating superfluous aborts.

CHAPTER 1
INTRODUCTION

Mobile ad hoc networks are self-organizing network architectures in which a collection of mobile nodes with wireless network interfaces may form a temporary network without the aid of any established infrastructure or centralized administration. They define a new computing environment in which hosts travel through physical space and communicate in an opportunistic manner via wireless connectivity. In some cases, the hosts share a single broadcast medium, while in others some hosts act as routers for other hosts, therefore extending the range of communication beyond the transmission range of a single node.

One class of applications uses the concept of group communication. A group can be defined as a set of collaborating processes communicating over a network. Group communication is a means of providing multipoint-to-multipoint communication by organizing processes into groups.

Group Communication

An example of a group-communication-based application for ad hoc networks would be a set of emergency personnel and vehicles collaborating in a disaster-stricken area. Another application would be soldiers collaborating in a war zone. Yet another example would be a multimedia conference with the delegates collaborating using laptops and handheld computers.

The traditional application of group communication has been in distributed systems and to implement reliable multicast. The currently emerging domain of group

communication includes peer-to-peer based applications for sharing resources, one-on-one collaboration, and searching.

Motivation

Many problems are associated with directly adapting existing group membership protocols to ad hoc networks. Most existing group membership algorithms make the assumption that the network infrastructure is static and reliable, and that the network hosts are immovable. However, this is not the case in ad hoc networks.

Very few group membership protocols exist for ad hoc networks. One was developed by Roman, Huang, and Hazemi (2000). However this protocol generates a large number of messages for each run, hence causing unnecessary wastage of battery power.

We observed that most existing group membership algorithms use aborts extensively (Abbadi, Skeen and Cristian 1985) to deal with concurrent attempts to change the group membership list, with the objective of maintaining view consistency. These aborts mean that coordinating hosts must restart the group membership protocol to reflect a change in group membership in all the nodes in the current group. This often leads to unnecessary repetition of messages, thus increasing the total message count. In ad hoc networks, this situation is compounded by the high rate of hosts entering and leaving the group, often due to partitioning in the network or subnetworks merging to form a larger network. The resulting multiple invocations of the protocol lead to a high number of aborts and hence a high number of protocol restarts, thus incurring a high overhead (number of messages exchanged). Because considerable power is required to transmit and receive messages, unnecessary wastage of power occurs if the tactic of aborts is used to maintain group membership in ad hoc networks.

We identified an opportunity to adapt an existing group membership algorithm so to suit ad hoc networks. We began with the view formation protocol of the virtual partitioning algorithm (Abbadi, Skeen and Cristian 1985). We modified the algorithm to reduce the number of unnecessary aborts generated, thus reducing the number of superfluous restarts and hence the number of messages generated. Devices in ad hoc networks have limited resources in terms of battery power, and by reducing the number of messages exchanged we incur a considerable saving in battery power. In ad hoc networks, where partitions are a way of life, we propose that our small contribution to the view formation protocol will go a long way toward decreasing the message overhead that occurs when group configuration changes are attempted.

The rest of this thesis is structured as follows. Chapter 2 deals with ad hoc networks. It defines mobile ad hoc networks, describes their characteristics and discusses factors affecting their overall performance. Chapter 3 discusses group communication in general. It formally defines a group, discusses group views and the group membership maintenance problem, describes the virtual partition algorithm (Abbadi, Skeen and Cristian 1985), discusses Ken Birman's contribution (Birman and Glade 1995) to group communication and finally introduces the peer-to-peer paradigm with a brief discussion on JXTA. Chapter 3 also defines the problem of group membership in ad hoc networks, and describes Roman, Huang, and Hazemi's protocol for reliable group membership maintenance. Chapter 4 deals with group communication in ad hoc networks. It discusses our approach to solving the same problem. Chapter 5 presents a comparative study of Roman, Huang, and Hazemi's protocol (2000), the virtual partitioning algorithm's view formation protocol (Abbadi, Skeen and Cristian 1985), and the improved view formation

protocol. It also presents the results derived from the study, an analysis of the results, and suggests various improvements to our protocol.

CHAPTER 2
BACKGROUND ON AD-HOC NETWORKING

Wireless networks can be classified into two types: infrastructure networks and ad hoc networks. An infrastructure network consists of fixed and wired gateways. A mobile host communicates with a bridge in the network (called base station) within its communication radius. The mobile host can move geographically while it is communicating. When it goes out of range of one base station, it connects with a new base station and starts communicating through it. This is called handoff. In this approach the base stations are fixed (Misra 1999).

A "mobile ad-hoc network" (MANET) is an autonomous system of mobile nodes, each of which is required to act as a router (connected by wireless links). The union of these nodes forms an arbitrary graph (Macker 2002). The routers are free to move randomly and organize themselves arbitrarily; thus, the network's wireless topology may change rapidly and unpredictably. Such a network may operate in a stand alone fashion, or may be connected to the larger Internet. The size of ad hoc networks may vary from a few connected nodes to hundreds or thousands of nodes

In contrast to infrastructure-based networks, in ad hoc networks all nodes are mobile and can be connected dynamically in an arbitrary manner. All nodes of these networks behave as routers and take part in the discovery and maintenance of routes to other nodes in the network. Ad hoc networks are very useful in emergency search-and-rescue operations, meetings or conventions in which persons wish to quickly share

information, and data acquisition operations on inhospitable terrain (Royer and Toh 1999).

Military tactical operations are still the main application of ad-hoc networks today. For example, military units (soldiers, tanks, or planes) equipped with wireless communication devices could form an ad-hoc network when they roam in a battlefield. Ad-hoc networks can also be used for emergency, law enforcement, and rescue missions. Since an ad-hoc network can be deployed rapidly with relatively low cost, it becomes an attractive option for commercial uses such as sensor networks or virtual classrooms (Zhou and Haas 1999).

The goal of mobile ad hoc networking is to provide a rapidly deployable means of communication (and computing) independent of a pre-existing infrastructure (Perkins, Hughes and Owen 2002).

Characteristics of Mobile Ad Hoc Networks

Mobile ad hoc networks have several inherent characteristics that make them different from wired networks or infrastructure based wireless networks. A few of these properties are discussed below.

They Do Not Rely on Infrastructure and May Require Multi-Hop Routing

In infrastructure-based wireless networks, the wireless mobile node is never more than one hop away from a base station that can route data across the communication infrastructure. In mobile ad hoc networks, there are no base stations. Because of limited transmission range, multiple hops may be required for nodes to communicate across the ad hoc network. Routing functionality is incorporated into each host.

Dynamic Topology

In mobile ad hoc networks the hosts act as routing nodes. These routers are free to move arbitrarily within the network. This means that the network has a constantly changing topology, and the changes in topology may occur at a fairly rapid rate. Node mobility also implies that the nodes will often move out of each other's transmission range, causing link failures.

Frequent Disconnections

The nodes participating in a mobile ad hoc network may disconnect from the network without warning. This could happen for several reasons. A person may shut down his laptop when he is finished working or he may move out of the transmission range of all other nodes in the network. Also, in war or rescue operations, the mobile node may run out of battery power, shut down to conserve battery power, or may be destroyed.

High Likelihood of Network Partitions

Sometimes, a number of nodes may collectively move out of transmission range of all other nodes in the network and still remain within transmission range of each other. This event is known as a network partition. When network partitions occur, each partition will believe that it represents the whole ad hoc network and the nodes that are no longer within transmission range have failed. These partitions may later rejoin to yield the original network.

Limited Power and Resources

The nodes that participate in a mobile ad hoc network usually operate on batteries and have limited resources. These nodes include devices like notebook computers, handheld computers like PDAs, and other battery-powered specialized computers. Thus

any application designed for ad hoc networking should be power aware and limited-resource aware. One way to achieve this would be to limit the number of messages sent by one particular node, since transmitting a message requires considerable power.

Factors Affecting Overall Network Performance

Several factors will affect the overall performance of any protocol operating in an ad hoc network. Node mobility causes link failures that negatively impact routing and quality of service support. Network size, control overhead, and traffic intensity greatly affect network scalability. These factors and the inherent characteristics of ad hoc networks may cause unpredictable variations in overall network performance. Perkins et al. (2002) determined the impact of five factors: node speed, node pause time, network size, number of traffic sources, and routing protocol (Royer and Toh 1999).

CHAPTER 3
BACKGROUND ON GROUP COMMUNICATION

Group communication is a means of providing multi-point to multi-point communication, by organizing processes into groups (Chockler, Keidar and Vitenberg 2001). A group may be defined as a set of collaborating processes. Members are aware of each other's existence, and use this knowledge to send messages to each other. For example, a group can consist of users playing an online game with each other. Another group can consist of participants in a multimedia conference. Processes communicate either by addressing a message using the logical name of the member, or by sending the message to all members of the group by addressing it using the logical name of the group.

Formal Definition of a Group

The group has been formally defined in the following way by Roman, Huang, and Hazemi (2000). They have modeled an ad hoc network as a graph $C_0 = G (V, E_0)$, where V is the set of mobile hosts and E_0 is the set of bi-directional communication links among the hosts. The presence of an edge (u, v) indicates that u is within transmission range of v and vice versa.

Since it is infeasible to maintain an accurate picture of the physical connectivity graph C_0 in the presence of unexpected disconnections, the notion of a logical connectivity graph C = G (V, E) has been introduced. The two share the same set of vertices (hosts) but the latter is missing some links. The edges to include in the logical graph are determined by some group management policy designed to overcome the difficulties caused by unannounced disconnections taking place in the physical system.

A group G is simply a maximum sized sub graph of the logical connectivity graph C. Since each host process u is always the member of some group, G(u) is used to denote the group that includes u and the notation may be extended to V(u) and E(u) to refer to the vertices and edges of the group. The group management policy is assumed to add a new edge to the logical graph after it appears at the physical level and to remove it before it is likely to disappear from the physical graph.

Group View and the Group Membership Maintenance Problem

The list of all hosts that are currently part of the group as perceived by any member of the group is known as the group view. The group view is considered to be consistent among all processes in the group. 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.

The function of the group membership service is to maintain a consistent group view across all members of the group. Only then can we say that the group membership service is reliable. Events that could change the group view are host processes entering the group, host processes leaving the group, processes disconnecting voluntarily or involuntarily, inactive processes becoming active again, hosts moving in and out of transmission range, etc. These events can be classified into two major categories: joins and leaves. We can model joins and leaves as transactions that must be committed by all current members of the group. Consequently these transactions, like regular transactions must conform to the ACID properties, i.e., atomicity, consistency, isolation and durability.

- Atomicity: This property ensures that the event is indivisible. Either the entire join or leave operation should go through successfully, or it should abort leaving no other side effects.

- Consistency: This property ensures that the join or leave event should either execute on all current members of the group, or none of them. This property also implies that the order of joins and leaves should remain consistent over all members of the group. It is not necessary that the order of joins and leaves as perceived by a particular member must be the same as the order in which they actually occur, but it is necessary that all members of the group perceive the same order of joins and leaves. Thus the order of joins and leaves must be serialized.

- Isolation: This property ensures that all join and leave events are executed as though they were happening completely independent of each other, even if they are executing simultaneously in practice.

- Durability: This property ensures that the effects of the join or leave events is permanent; e.g., the effect of a particular join event is that the name of that member is added to every member's membership list, and it stays in this list till a corresponding leave for the same member is executed.

Applications of Group Membership Algorithms

The two most important properties of group membership algorithms are consistency--all group members must agree upon the same group view--and serializability--all views must be installed by all members in the same order. Making the assumption that the underlying group membership algorithm provides consistency and that group configuration changes are serialized simplifies the design of many distributed systems.

Two examples of such applications are replicated fault tolerant services and service availability management algorithms. In one of their papers, F. Cristian and S. Mishra (1994) describe a service availability management scheme, in which they state the assumption that all the members of the group have the same history of group configuration changes. Serializing group configuration changes is necessary in such systems in order to preserve consistency while restoring the system state in the wake of

server crashes. Also, when multiple servers are used to provide a highly available service, servers need to agree on the order of group configuration changes to ensure that their states are consistent.

Consider, for example a case in which a host attempts to join a group, then quickly exits the network. These actions may be detected by two different group members in the span of a very short time. If these changes are not serialized, they may arrive at some of the hosts out of order. The hosts that receive these messages will correctly reflect a group view not containing the removed host. The hosts that receive these messages out of order however, will first attempt to remove a non-existent member from the group, which may or may not cause an error to be generated, and then proceed to add the member to their view. Thus some of the hosts will have the new member in their views, and the system will continue to exist in an inconsistent state until the next run of the configuration change detection protocol. This is unacceptable for many distributed systems.

View Formation Protocol

Abbadi, Skeen and Cristian have presented in their paper (1985) an algorithm called the *virtual partition* (VP) algorithm to manage replicated data. The algorithm has been designed so that a transaction (in a distributed database system) never has to access more than one copy to read a data item. Part of this algorithm is the *view formation protocol* that allows consistent updating of views for sites storing replicated data.

The basic idea of VP is for each site A to maintain a *view*, consisting of the sites with which A believes it can communicate. We may denote this set by v(A). Maintaining the views in a consistent manner is a non-trivial matter because site and communication failures occur spontaneously. The virtual partition algorithm surmounts this difficulty by

using a special *view update transaction*. When a site detects a difference between its current view and the set of sites it can actually communicate with (as the result of a join or a depart), it initiates a View Update transaction whose purpose is to adjust the views of all the sites in the new view. In adjusting the views of the sites in a new view, view updates must be coordinated to avoid problems like the one exemplified by the following scenario. Consider a network with four sites A,B,C,D such that at some point v(B) ={B}, v(D) = {D} and v(A) = v(C) = {A,C}. Later, B discovers that it can communicate with A and C but not D and, at the same time, D discovers that it can communicate with A and C but not with B. [1] Thus B will initiate a view update to create view {A, B, C} and D a similar one to create view {A, C, D}. Unless these two activities are properly coordinated, it is possible that A joins the first view and C the second, creating the situation where v(A) = v(B) = {A, B, C} and v(C) = v(D) = {A, C, D}.

To avoid such inconsistencies, the virtual partition algorithm uses a view formation protocol. Associated with each view is a unique view identifier (VID).

When a site A wishes to form a new view, say v, it generates a new VID greater than its present VID. A then generates a two-phase protocol to establish the new view.

- A sends a JOIN-VIEW message to each site in v and waits for acknowledgements. This message contains the new VID.
- When a site B receives the JOIN-VIEW message, it regards this message as an invitation to join the new view being created by site A. B accepts the invitation provided its present VID is less than the new VID contained in the JOIN-VIEW, and if it has not already received another invitation to join a view with a higher numbered VID. If either of these conditions is not satisfied, B rejects the invitation. Accepting the invitation means sending a positive acknowledgement; rejecting it means sending either a negative acknowledgement or nothing at all. In the latter case, negative acknowledgements are recognized by timeouts. After

[1] Such anomalous situations can arise due to timeout failures

accepting or initiating an invitation and before committing itself to any view, a site is not assigned a new view.

- After receiving acknowledgements, A can proceed in one of the following two ways. It may abort the creation of the new view, either by sending explicit ABORT messages or by not sending any messages at all. In this event, A may attempt to restart the protocol using a greater new VID, hoping it will convince more sites to accept the invitation. Alternatively, A may ignore any rejected invitations and proceed to form a new view consisting only of the set of sites *v' (v'⊆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 receipt of the VIEW-FORMED message.

Returning to the mentioned example, if this protocol is followed, A and C will make a consistent choice. Both will join either the view generated by B {A, B, C} or the view initiated by D {A, C, D} depending on which of the two had a greater VID. The third possibility, that neither A nor C joins either new view, would be obtained if the present view ID of A and C is greater than the VIDs of the views that D and D are attempting to create.

To ensure 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.

For example, consider two hosts *a* and *b* trying to simultaneously install views (a,k) and (b,k) respectively. Assume a host *x* that receives *a*'s JOIN-VIEW first and *y* that receives *b*'s JOIN-VIEW first (Figure 3.1A). Both *a* and *b* accept the VIDs (a,k) and (b,k) and send a and b their respective acknowledgements(Figure 3.1B). Then *x* receives *b*'s JOIN-VIEW and since *b* > *a*, accepts the invitation (b,k) and sends the

acknowledgement (Figure 3.1C). *y* receives *a*'s JOIN-VIEW and since (a,k) <(b,k), rejects the JOINVIEW by not sending the acknowledgement (Figure 3.1D). Hence a does not get the acknowledgement from *y* and is forces to abort the transaction. Hence it sends an ABORT message to all hosts including *x* and *y* (Figure 3.1F). *b* has received acknowledgements from all nodes including both *x* and *y*. Hence *b* commits its view by sending a VIEW-FORMED message to all other hosts (Figure 3.1E).

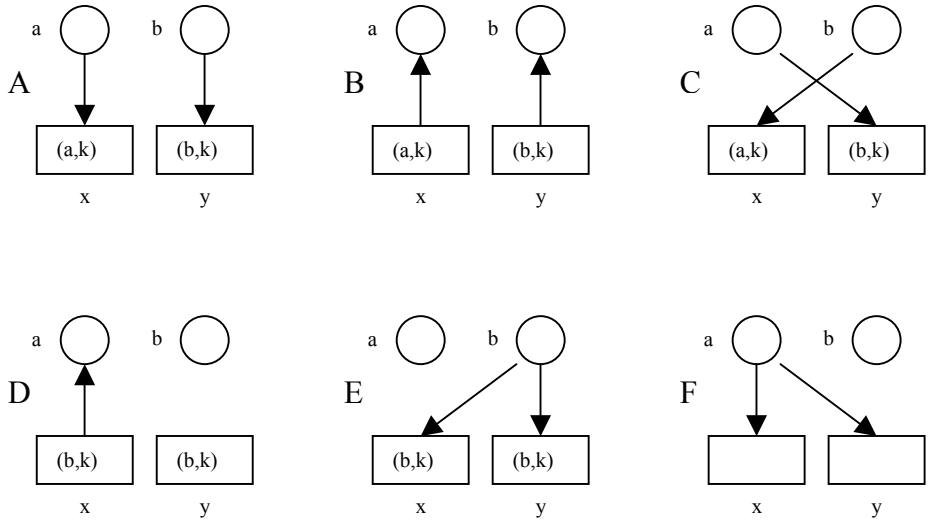

Figure 3-1 Dynamics of view formation protocol

A) Here a and b send JOIN-VIEWs. Node x receives a's and node y receives b's first.
B) Here x and y send acknowledgements to and b respectively.
C) Here x receives b's and y receives a's JOINVIEW.
D) Here x accepts b's JOINVIEW since (a,k) < b,k) and sends an acknowledgement. y rejects a's JOINVIEW.
E) Here b sends VIEW-FORMED message to x and y. Node a has to abort since it has not received all acknowledgements, including y's.
F) a sends an ABORT message to all nodes.

Thus we have seen that the protocol of Abbadi et al. (1985) serializes group configuration changes by aborting the transaction that has the lower VID, and restarting it with a higher VID. Thus an extra set of messages is generated for each ABORT that occurs. If we were to somehow eliminate the ABORTs, i.e., implement a scheme to

serialize group configuration changes without resorting to aborts, we will have a considerable saving in the number of superfluous messages generated, thus saving transmission power.

Kenneth P. Birman's Work

Considerable work on groups has been done by Ken Birman and his colleagues at Cornell University. In his papers, Ken Birman has introduced a number of important concepts. Traditionally, the idea of groups has been used to implement distributed systems and implementing reliable multicast.

Virtual Synchrony

A process group is a natural way to organize hosts to perform reliable multicast. Multicast Communication patterns are usually not arbitrary but repeatedly confined to the same set of hosts. Most activities such as maintaining a replicated server, coordinating a distributed program etc. require that all members of a group hear of an update to the global state, e.g., a change in the group configuration. In addition, much of the work of performing a reliable multicast can be reduced to maintaining a consistent group view (Malki, Birman, Schiper and Ricciardi 1994).

The virtually synchronous group communication paradigm (Birman and Joseph 1987) allows processes to be organized in groups within which messages are exchanged to achieve a common goal (Rodrigues, Guo, Sargento, Van Renesse, Glade, Verissimo, and Birman 1996). Virtual synchrony ensures that all processes in a group receive consistent information about the group membership in the form of views. The membership of a group may change with time because new processes may join the group and old processes may fail or voluntarily leave the group. Virtual synchrony also orders messages with view changes, and guarantees that all processes that install two

consecutive views deliver the same set of messages between these views. An algorithm is virtually synchronous if

- There is a unique group view in any consistent state on which all the members of the group agree.
- If a message *m* is multicast in the group view *v* before view change *c*, either no process in *v* that executes *c* ever receives *m*, or every process that executes *c* receives m.

Virtual synchrony gives a precise meaning to the notion of a delivery list. Between any two consecutive view changes, a set G of messages is multicast, and the senders are restricted to the processors in the view. If a processor *p* is removed from the view, the processors remaining in the view can assume that *p* has failed. Virtual synchrony guarantees that no messages from *p* will be delivered in the future. In addition, virtual synchrony makes reliable message delivery easier.

Intuitively, all processors in a group view v that do not fail receive all messages in G. A processor p that fails might not receive all of G. However, since message delivery is synchronized with view changes, we know what messages *p* must have received and therefore what actions to take to let *p* recover when it is repaired.

There is a subtlety in the definition of Virtual Synchrony. It is not necessarily the case that if a message *m* is delivered to a processor *p* in view *v*, then *m* is delivered to all processors in *v*. This is because the sender of the message can also fail.

D-Uniformity

Dynamic Uniformity or D-Uniformity has been described by Birman et al. in their paper (Malki, Birman, Schiper, and Ricciardi 1994). In D-Uniformity, there are certain actions that, if taken by any process in the system, must be eventually taken by every other active process. Only failure and forcible removal can excuse the process from

taking the action. This includes additions or deletions from the group view.

The motivation for D-Uniformity is that in a large distributed system, processes will often act on behalf of the system as a whole. While actions such as joins or leaves may occur locally, they must be eventually known to the entire membership. D-Uniformity captures the required propagation of such action, as well as the obligation of other processes to take these actions, while avoiding notions such as 'correct/incorrect' processes.

D-Uniformity is trivial to solve if the communication channels are reliable and the membership of the system is known to all processes. When channels are lossy, a process initiating an action must know that its cohorts are aware of the action it is initiating.

Consistency Types

Birman and Glade have defined various properties of a failure reporting mechanism in a distributed system (Birman and Glade 1995). A mechanism is live if it reports real failures within a finite time. A failure reporting mechanism is safe if at a given time, all operational hosts agree on the state of all other processes, and failed hosts are prevented from communicating with operational ones.

A system is said to be consistent if it guarantees safety at times determined by the evaluation rule. Under the evaluation rule, safety holds at the times specified by the rule for evaluating the safety property. Birman and Glade (1995) have presented 3 types of consistencies.

Stabilization Consistency

The requirement for stabilization consistency is that if the system stays quiet for sufficiently long, it will enter a state where consistency holds. The system must converge to a state in which consistency holds.

Piecewise Consistency

A piecewise consistency evaluation rule slices the execution into segments and says that, for any execution, within any segment, the safety property holds for all simultaneous states, i.e., states processes could have entered at the same time because of uncertainty in scheduling, execution speed or message delays.

In a piecewise consistent system, within each execution segment, all system processes have the same membership information.

This models the behavior of a system in which transitions from one membership list to another occur instantaneously. However, no real system can make transitions instantaneously so the definition does not require instantaneous behavior, it requires that the execution be one in which all processes concurrently and indivisibly see membership changes. In between transitions, processes must have identical views of which processes are operational and faulty.

Uniform Consistency

Uniform Consistency is motivated by the desire to limit the inconsistency that could be observed when a faulty host reconnects to the system and compares its state to other hosts that remained operational during its faulty period. Protocols for achieving this kind of consistency include the 2 phase commit in distributed databases. Thus we can account for hosts that fail during the execution of the JOIN protocol. The advantage of uniform consistency is that operational components know more about the potential final state of failed components that can simplify recovery algorithms. The disadvantage is that it is costly to support.

Birman and Glade have shown various strategies to implement consistent failure reporting in distributed systems. The one that is of the most interest in this discussion is

the dynamic consensus/group-membership protocol. Here the group membership protocol or GMP is the sole agent for recovery and failure decisions (we consider a failure to be the event of a host leaving the group and recovery the event of the a host rejoining the group).

ISIS and Horus

Ken Birman and his colleagues at Cornell University have developed two distributed systems: ISIS (Cho and Birman 1994), and its successor Horus (Van Renesse, Birman, and Maffeis 1996). The basic approach of these two projects is to develop a toolkit for distributed programming. Tools have been included for managing data, synchronizing distributed computations, automatic recovery and dynamic reconfiguration of the system to adapt to fluctuating workloads. ISIS is a system developed out of a study of fault tolerance in distributed systems. The system implements a collection of techniques for building software for distributed systems that performs well, is robust despite both hardware and software crashes, and exploits parallelism.

Group management in ISIS is similar to a coordinating processor executing a Three-Phase commit (Skeen 1982) to atomically install new group views. 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.

Horus is an extension to ISIS. In its related literature, Horus has been described as a flexible and extensible process-group communication system, in which the interfaces seen by the application can be varied to conceal the system behind more conventional

interfaces, and in which the actual properties of the groups used (membership, communication, events that affect the group) can be matched to the specific needs of the application.

Peer-to-Peer Paradigm

The peer-to-peer (P2P) paradigm is the way in which different computers (hence processes) interact with each other without the intervention of a centralized entity like a server. The advantage of using peer-to-peer is that we have avoided relying upon a central entity that is a single point of failure. This is exactly how we propose to maintain

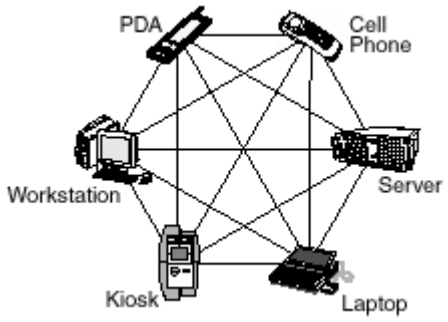

Figure 3-2 Peer-to-peer computing enables direct communication among peers and a new interaction style (Source: Gong 2001)

groups and this is the environment for which we need to solve the reliable group membership problem. Thus we can model each member of a group as a peer.

Thus, while traditionally group communication has been used for multicast messages and for modeling distributed systems, we can now use group views to maintain a list of active peers that a given peer can interact with.

Peer-to-peer computing offers an intuitive model for the most fundamental Internet activities: searching and sharing. Although today's applications are primarily for finding, retrieving and using media files, they hint at what complete access to the web can deliver in the future.

- Resources, including information and processing power, can be shared directly from those who have them to those who need them. Documents can be managed and encrypted so that they are anonymous, likely to exist somewhere in the network at all times, and more secure.

- Peer-to-peer searches are distributed, parallel, and asynchronous, enabling deep searches of Internet content that quickly yields up to the minute results. Current look-ups conducted by search engines are limited by the months it takes their crawlers to traverse the sites that they index. In contrast, peer-to-peer search results are more relevant, and can be more focused, e.g., by qualifying the peer group, and qualifying the search, more precise results can be obtained over a much larger search space.

- Instant messaging systems can locate users quickly with distributed searches, enabling direct user-to-user communication or collaboration independent of any service provider.

- Buyers and sellers can be matched directly through P2P Auction and transaction services, and new generations of distributed resource-sharing tools like the Search for Extra Terrestrial Intelligence (SETI@home) will begin to appear.

- Peer shells will enable users and developers to experiment and prototype new applications using simple scripting languages. Command line administrative tools can be used to manage applications and their peer groups.

Peer-to-peer computing enables applications that are collaborative and communication focused; more probabilistic than deterministic. Applications that are well suited to this model are those that can tolerate the coming and going of individual peers. In this model, if the desired result is not obtained, users can try again later. The information exchanged in P2P environments is timely and accurate, yet results may vary from time to time depending on which peer group members are available.

High availability comes through the existence of multiple peers in a group making it likely that at anytime there is a peer in a group able to satisfy a user request. This stands sharply contrasted to traditional computing models where high availability comes through complex load balancing and application fail-over schemes. Peer-to-peer computing leverages available computing performance, storage, and bandwidth found on systems all

over the world, and works because people realize that there is value in sharing their power with others in order to reap the benefits when they need it themselves.

Project JXTA

Seeing the potential of peer-to-peer computing to enable access to a broader, deeper web and the need to free today's model from its current limitations Sun's Chief Scientist Bill Joy started a small research effort called JXTA (Gong 2001). In many ways, the world of peer-to-peer computing is juxtaposed to the hierarchical client server model, hence the name *Project JXTA*. JXTA technology is a set of open protocols that allow any connected device on the network ranging from cell phones and wireless PDAs to PCs and servers to communicate and collaborate in a P2P manner. JXTA peers create a virtual network where any peer can interact with other peers and resources directly even when some of the peers and resources are behind firewalls and NATs or are on different network transports.

Project JXTA is building core network computing technology to provide a set of simple, small and flexible mechanisms that can support peer-to-peer on any platform, anywhere, and at any time. The project is first generalizing peer-to-peer functionality and then building core technology that addresses today's limitations on peer-to-peer computing. The focus is on creating basic mechanisms and leaving policy choices to application developers.

JXTA technology leverages open standards like XML, Java technology, and key concepts that make the UNIX operating system powerful and flexible, including the ability for shells to connect commands together using pipes to accomplish complex tasks. By using existing, proven technology and concepts, the result will be a peer-to-peer system that is familiar to developers and easy to build upon.

Project JXTA is creating a peer-to-peer system by identifying a small set of basic functions necessary to support peer-to-peer applications and providing them as building blocks for higher level functions (Figure 3-2). At the core, capabilities must exist to create and delete peer groups, to advertise them to potential members, to enable others to find them, and to join or leave them. At the next layer, the core capabilities can be used to create a set of peer services, including indexing, searching and file sharing. In addition, peer commands and a peer shell have been created to create a window into the JXTA technology based network.

JXTA applications are built using peer services as well as the core layer. The project's philosophy is to support the fundamental levels broadly, and rely on the peer-to-peer development community to provide additional peer services and applications. Peer applications enabled by both the core and peer service layers include peer-to-peer auctions that link buyers and sellers directly, with buyers able to program their bidding strategies using a simple scripting language. Resource sharing applications like SETI@home can be built more quickly and easily, with heterogeneous, worldwide peer groups supported from the first initiative. Instant messaging, mail and calendaring services can facilitate communication and collaboration within peer groups that are secure and independent of service provider hosted facilities.

JXTA technology provides fundamental mechanisms for solving many of today's distributed computing applications, enabling a new generation of ubiquitous, secure, interoperable, heterogeneous applications. It currently supports Java technology based platforms and systems without Java technology. In the future JXTA technology will support small mobile devices with limited memory footprints. JXTA software's

combination of Java technologies with XML data representations provides flexibility, power and the ability for vertical applications to interoperate in ways that overcome the limitations of peer-to-peer software today.

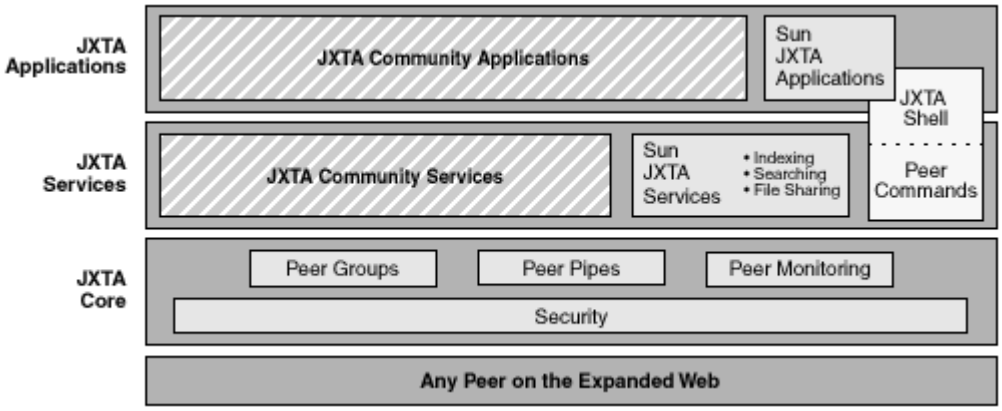

Figure 3-3 JXTA software is layered with core functionality developed through an open, collaborative effort and higher level services developed by P2P community developers (Source: Gong 2001)

Group Communication in Ad Hoc Networks

After discussing group communication in general, we now look at group communication in ad hoc networks. We have the same group membership maintenance problem as described in the previous chapter, but this time the idiosyncrasies of ad hoc networks need to be dealt with.

Problem Statement

The problem in this case is to implement a reliable group membership service in the context of ad hoc networks. This service should make group configuration changes appear to be atomic, while at the same time deal with frequent disconnections, low bandwidth, low transmitting power and network partitions. We need to design a protocol so that group configurations are serialized.

Designing a protocol as described above is non trivial since nodes can fail without warning, and communication links are not reliable. Also in the case where we deal with network partitions rejoining, the need to solve a problem of which group should merge into which other group, since each partition will be under the assumption that it itself represents the real group.

Roman, Huang, and Hazemi's Protocol

Roman, Huang, and Hazemi (2000) presented a protocol that solves the group membership problem in ad hoc networks. They considered the special case of a single host *u* joining or leaving the group G in an ad hoc mobile network. The group is assumed to have a leader whose main function is to serialize all configuration changes. The leader also decides when a host is to be admitted to or eliminated from the group, and make each member know the current composition of the group at all times. The leader's decisions are based on the relative locations of all hosts within communication range of with the group. They have also assumed that host failures do not occur and that communication channels between hosts are reliable, bi-directional and FIFO.

When an isolated host *u* discovers a host *v* that is already part of the group G, it asks *v* about the identity of the group leader. Here the authors have assumed that hosts are aware of other hosts within their transmission range by having each host transmit a beacon at regular intervals and by listening to signals from other hosts. Under the assumptions made, they have guaranteed that the host *v* will receive this inquiry. Once *u* determines the identity of the group leader *l*, direct message communication with the group leader becomes possible and host *u* will start reporting its position to the group leader *l* at regular intervals. The assumption is that all members of the group do the same. Once *l* has decided to admit *u* into the group, it informs *u* and all the current members of

the group about the configuration change by sending them a message called the *join* message. The group leader makes this decision based on the relative locations of host u and of hosts within *u*'s communication range.

When a host receives a join message from the leader, it stops transmitting regular messages and sends a flush message to all other members of the group indicating that it now knows about the configuration change. Once the host receives the *join* message and *flush* messages from all the members in its group view, it adds host *u* to its list of group members and forms a new view, i.e., commits to the new group configuration that includes *u* as its member, and resumes sending regular messages. Since FIFO communication channels have been assumed, receiving *flush* messages on a particular link guarantees that there are no more messages in transit on that link that may have been sent in a prior group configuration. Mobile host *u* thus successfully joins the group G and all members of the group have the same view of the group, i.e., the same group membership list.

When the leader decides that a host *w* cannot be a member of the group, it sends a depart message to all members of the group. All members process the depart message similar to the way they process a join message and, once they receive all the flush messages from other group members, they remove w from their membership list.

Why Flush Messages are Necessary

The *flush* messages are needed to make sure the group configuration changes appear to be atomic. If hosts wait for all *flush* messages, all communication changes take place either before or after a host joins or leaves the group. This message is guaranteed to be the last message delivered on a communication link before a configuration change.

Dealing with Network Partitions

When a mobile node comes to life, it declares itself the leader of a single group containing itself. Here it begins to run a neighbor discovery protocol to find any groups in its vicinity. When a new group is found, the two groups may negotiate a merger. The conditions for merger may consider issues such as the guarantee that no physical partitioning is likely to take place within a certain amount of time. The merging protocol uses a leader election algorithm that ensures that the new group will have only one leader, i.e., all other hosts become regular members.

To prevent unannounced disconnection, the leader seeks to anticipate possible partitions by re examining the overall configuration anytime a member changes location. When the leader expects a physical partition to take place, it forces an announced disconnection that guarantees that a logical disconnection occurs before any physical disconnection becomes possible. Predicting partitions is made possible by the fact that members of the group constantly update their leader with regard to their current location.

Analysis

We now analyze Roman, Huang, and Hazemi's protocol with respect to the number of messages. We initially have the *join* message sent to all members of the current group which requires n messages. Then we have each host sending flush messages to every host in the group, accounting for n^2 messages. Therefore, we have a total of $n(n+1)$ messages or $O(n^2)$ messages for each run of the protocol. Clearly this is prohibitively large, especially when power is a scarce resource (since transmitting messages drains battery power). An O(n) algorithm to accomplish reliable group membership would be more desirable.

CHAPTER 4
EFFICIENT GROUP MEMBERSHIP PROTOCOL FOR AD HOC NETWORKS

Our approach is based on the view formation protocol of the virtual partition algorithm (Abbadi, Skeen and Cristian 1985). Our approach seeks to alleviate the shortcoming listed above. We have used the concept of a sorted list to buffer join-view messages, thus avoiding unnecessary aborts, while at the same time guaranteeing that join and depart events are serialized.

Problem with Existing Approaches

Roman, Huang, and Hazemi's protocol (2000) has a number of shortcomings. Firstly, it requires the presence of a centralized entity called the leader that is responsible for coordinating joins and departs. If this entity fails, then this event needs to be detected by the other members of the group and a new leader needs to be elected. Secondly, members need to be aware of their own physical location (possibly using a GPS system, for example), and update the leader regarding the same. Thirdly, since flush messages need to be sent by every node to every other node in the group, the number of messages will be of the order of n^2. These messages imply an increased number of transmissions, leading to increased power consumption

Proposed Approach

Our approach, based on the view formation protocol (Abbadi, Skeen and Cristian 1985), lets any member of the group assume the role of the coordinator. A new concept, that of a sorted list, has been introduced. In our approach, members of the sorted list are sorted in increasing order of their View IDs (VID).

Assumptions

We make a number of assumptions. First, we assume that each host has a globally unique identifier (the "name" of the host). Second, we assume that all communication links are reliable, bi-directional and FIFO.

Protocol Description

When a Site A detects a change in the configuration of the group, it initiates a two-phase protocol.

Site A sends a JOIN-VIEW message to each site in *v* and waits for acknowledgements. This message contains the new VID.

Every site maintains its own sorted list to store JOIN-VIEWs. When Site B receives the JOIN-VIEW message, it regards this message as an invitation to join the new view being created by Site A. Site B accepts the invitation provided its present VID is less than the new VID contained in the JOIN-VIEW, and adds the JOIN-VIEW to its sorted list. If its present VID is greater than that of the JOIN-VIEW, it rejects the invitation. Accepting the invitation means sending a positive acknowledgement; rejecting it means sending either a negative acknowledgement or nothing at all. In the latter case, negative acknowledgements are recognized by timeouts. After accepting or initiating an invitation and before committing itself to any view, a site is not assigned a new view.

After receiving acknowledgements, Site A sends out a VIEW-COMMIT message to all other sites in its group view. The other sites buffer these VIEW-COMMIT messages until the corresponding JOIN-VIEWs rise to the top of their respective *sorted lists*. When the corresponding JOIN-VIEWs are the top-most in their respective lists, the VIEW-COMMIT message is accepted and the JOIN-VIEW deleted from the list. If all hosts have not replied within a certain time period, Site A sends out an ABORT message

so that hosts can purge the concerned VID from their respective sorted lists. If VIEW-COMMITs are not received within a particular timeout period, the corresponding JOIN-VIEW expires and is deleted from its list. This is done in order to deal with host that have sent the JOINVIEW messages but have crashed before sending the corresponding VIEW-COMMIT or ABORT messages.

The scheme for VID is the same as the one suggested by Abbadi et al. (1985). To ensure 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. These messages could appear at different hosts at different times, therefore 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.

The decision of when the protocol must be run is left up to the application. A site may run a 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 view do not receive their messages. However, in scenarios such as four 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.

Consider two hosts *u* and *v* that want to join the group at roughly the same time. *u* asks *a* to add it to the group while *v* asks *b* to add it to the group. Let *a* and *b* generate JOIN-VIEWS such that VID generated by *a* is less than that generated by *b* (VIDs generated are guaranteed to be unique). Both *a* and *b* will send JOIN-VIEWS to all the current members of the group, who will add these JOIN-VIEWS to their respective sorted list after voting YES to both JOIN-VIEWS. Regardless of the order in which they are received, the JOINVIEW generated by *a* will precede that generated by *b* in all sorted lists. After collecting all the votes, both *a* and *b* will generate VIEW-COMMIT messages.

Figure 4-1 shows how unnecessary aborts are avoided. Let *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 (Figure 4-1A). Thus *x* has (a,1) at the top of its sorted list and *y* has (b,1) at the top of its sorted list. Now *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* (Figure 4-1B) Thus now *x* and *y* both have (a,1) at the top of their respective sorted lists. Then *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 (Figure 4-1C). *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). Now *y* examines the topmost element of its sorted list, finds (a,1) and thus buffers *b*'s VIEW-COMMIT. Then *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* (Figure 4-1D). 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. *x* deletes (b,1) from the top of its list (Figure 4-1E). Now *y* 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* and *b*'s VIEWs in the same order (*a,b*). 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.

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 additional n 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 excluding the attempt that was successful. This makes the modified view formation

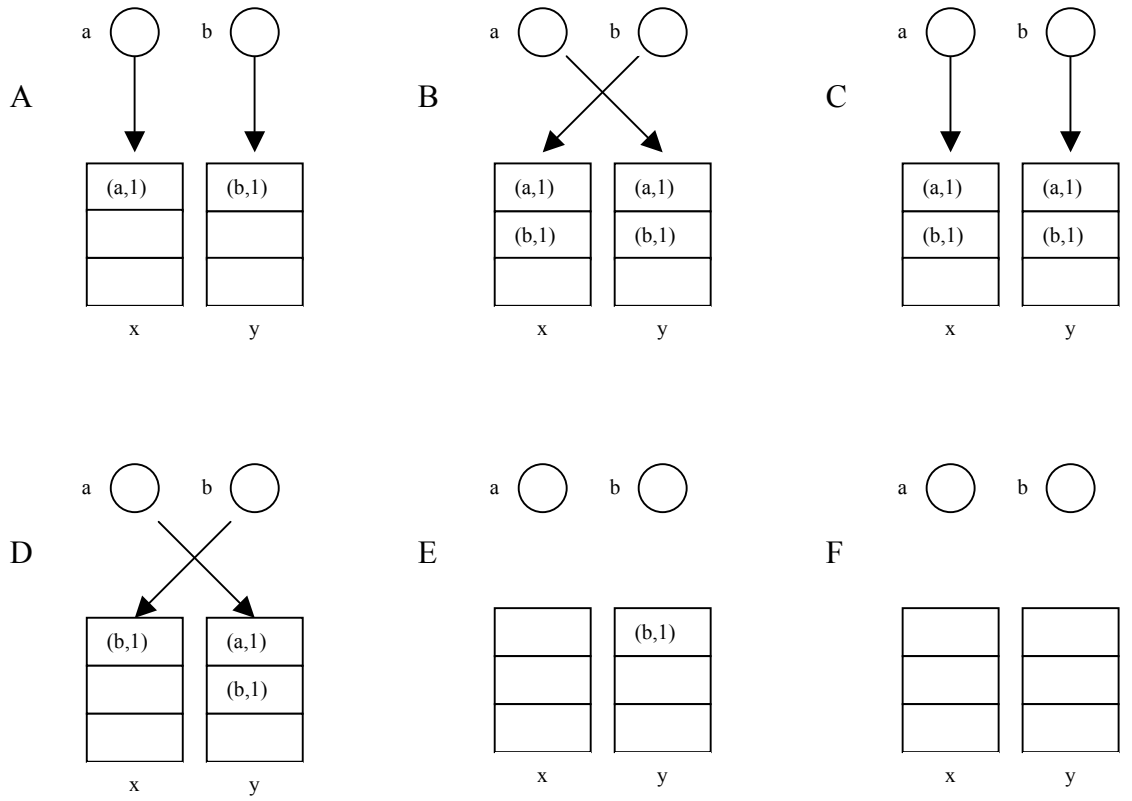

Figure 4-1 How two simultaneous JOINVIEW transactions can commit

A) Here a's JOIN-VIEW reaches x and b's JOIN-VIEW reaches y.
B) Here a's JOIN-VIEW reaches x and b's JOIN-VIEW reaches y.
C) Here x receives VIEW-COMMIT from a and y the one from b. x commits view (a,1).
D) Here y receives VIEW-COMMIT from a and x the one from b. x commits view (b,1) and y, the view (a,1)
E) Here y commits the view (b,1)
F) Here both x and y have committed the two views in the same order.

protocol superior to Roman, Huang, and Hazemi's protocol (2000). It also improves on Abbadi, Skeen, and Cristian's virtual partitioning algorithm (1985) in the context of ad hoc networks.

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 that the view ids $IDx_a < IDx_b$ (IDx_a is the ID of the JOINVIEW sent by x_a).

Without loss of generality, assume JOINVIEWx_a (the JOINVIEW message sent by x_a) reaches $x_1,\ldots x_i$ first and JOINVIEWx_b reaches $x_{i+1},\ldots x_n$ first.

$x_1,\ldots x_i$ receive JOINVIEWx_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 JOINVIEWx_b and send their YES votes $Ybx_{i+1},..Ybx_n$

$x_1,\ldots x_i$ then receive JOINVIEWx_b and put IDx_b in their sorted list, sending YES votes Ybx_1 to Ybx_i

$x_{i+1},\ldots x_n$ receive JOINVIEWx_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 VIEWCOMMITx_a to hosts $x_1,\ldots x_n$. 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 VIEWCOMMITx_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.

Consider another host x_r where all [name(x_p),…name(x_q)] < name(x_r)

Assume that $x_p,\ldots x_q$ get their JOINVIEWs to $x_1,\ldots x_i$ and x_r gets its JOINVIEWx_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,…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 JOINVIEWx_r which they vote for by sending Yrx_{i+1} ,…Yrx_n

They then receive JOINVIEWs from x_p,…x_q. They will then send their votes after which $x_{i+1,\ldots}x_n$ will cause x_p,…x_q to commit.

Thus x_r receives Yx_1,…Yx_n, thus it can send VIEWCOMMITx_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.

CHAPTER 5
EVALUATION, COMPARISON AND FUTURE WORK

This chapter presents a comparative study of the three different protocols for group membership in ad hoc networks. The first was developed by Roman, Huang, and Hazemi (2000), The second, the original view formation protocol of the virtual partition algorithm, was developed by Abbadi et al. (2000). We developed the third protocol (a modified version of the view formation protocol). The study of all three protocols was conducted in the following way.

A simulator was written for each of the three protocols. Each simulator was written as a Java program. Events such as joins or departures 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 particular host, a message was sent to it to set a flag 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. Ten readings were taken for each protocol, each time with a different seed for the random number generator. For each time interval set, a graph of the average number of messages versus the number of nodes joining was plotted. The graphs below show the results obtained.

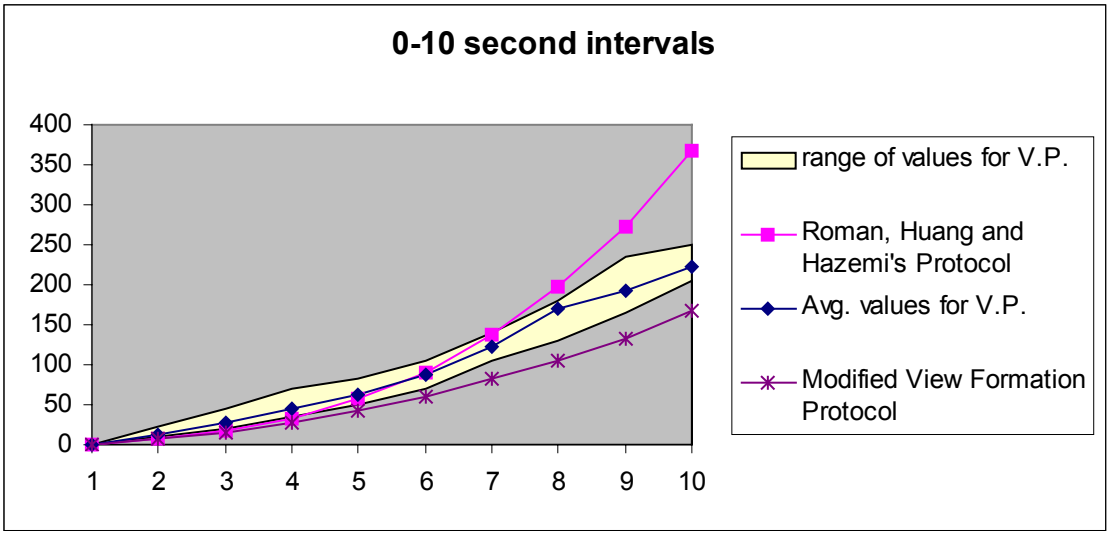

Figure 5-1 Messages vs. Joins for 0 to10 second intervals between joins

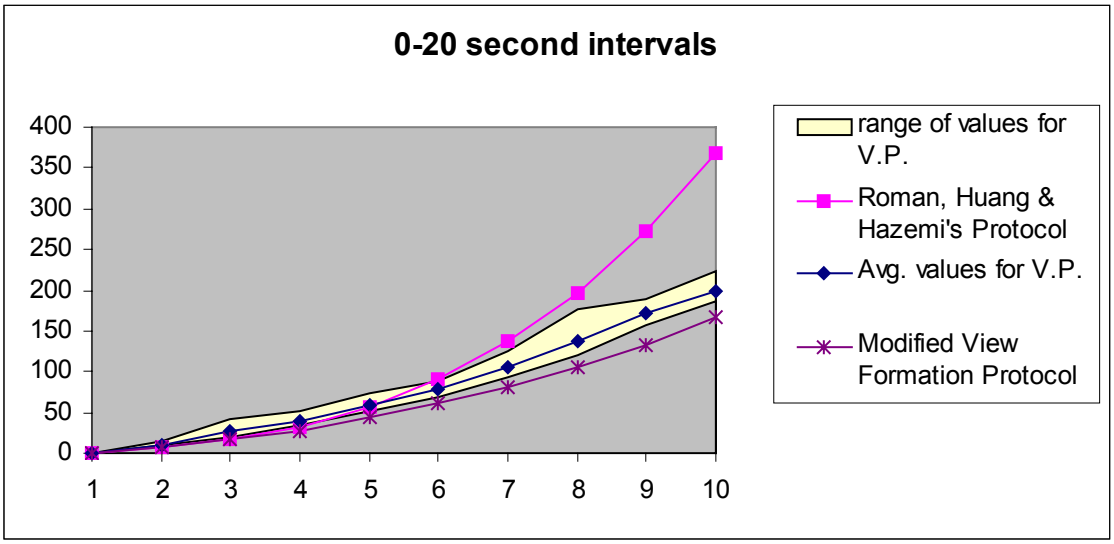

Figure 5-2 Messages vs. Joins for 0 to 20 second intervals between joins

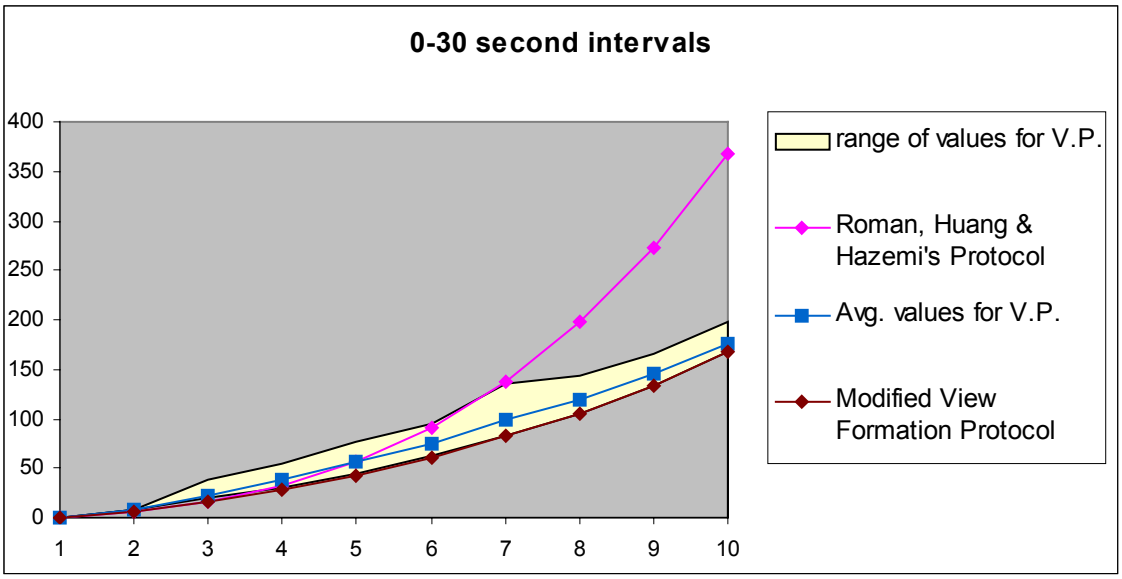

Figure 5-3 Messages vs. Joins for 0 to 30 second intervals between joins

Observations

It was observed that 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 average number of messages for the virtual partitioning algorithm's view formation protocol was comparatively less, 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. When the 0-30 second interval was used, the best case behavior for the view formation protocol was almost as good as that of the modified view formation protocol, due to the reduced likelihood of 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.

Future Work

We have observed that the performance of our modified version of the view formation protocol can be improved in the following ways.

- We assume reliable communication channels for our protocol, where all communication links have an upper bound on the message delays. If a few communication links have delays that exceed the timeout period for JOIN-VIEW messages, the hosts at the receiving end will expunge these JOIN-VIEWs from their sorted lists before their corresponding VIEW-COMMIT messages arrive. This will leave the system in an inconsistent state. The protocol may be improved so that it is not necessary to make the assumption of communication link delays having upper bounds.

- In our protocol, all hosts are capable of detecting changes in the group membership. In highly aware groups, two or more hosts may detect the same change in group membership and initiate separate instances of the protocol for effecting the same configuration change. This may be improved upon in two ways. First, the membership change detection policy may be designed to ensure that a change is detected only once. Secondly, we may implement a scheme where the change in the view is transmitted along with the JOIN-VIEW message, and in case of JOIN-VIEWS with different VIDs trying to effect the same group configuration change, the one with the higher VID may be given preference.

LIST OF REFERENCES

Abbadi A., Skeen D., and Cristian F. 1985. An efficient, fault-tolerant protocol for replicated data management. Proceedings of the Fourth ACM SIGACT-SIGMOD Symposium on Principles of Database Systems, ACM press, Portland, OR, pp. 215-229.

Birman K. and Glade B. 1995. Reliability through consistency. IEEE Software, 12(3): 29-41.

Birman K. and Joseph T. 1987. Exploiting virtual synchrony in distributed systems. 11th ACM SIGOPS Symposium on Operating Systems Principles, ACM press, Austin, TX, pp. 123-138.

Cho K. and Birman K. 1994. A group communication approach for mobile computing mobile channel: An ISIS tool for mobile services. Technical Report # 94-1424, Computer Science Department, Cornell University, Ithaca, NY.

Chockler G., Keidar I., and R. Vitenberg 2001. Group communication specifications: A comprehensive study. ACM Computing Surveys 33(4): 1-43.

Cristian F. and Mishra S. 1994. Automatic service availability management in asynchronous distributed systems. Proceedings of the Second International Workshop on Configurable Distributed Systems, IEEE press, Pittsburgh, PA, pp. 58-68.

Gong L. 2001. Project JXTA: a technology overview. Sun Microsystems Inc., Palo Alto, CA. Available at http://www.jxta.org/project/www/docs/ TechOverview.pdf. Accessed on 04/22/2002.

Macker J. 2002. Mobile Ad Hoc Networks(MANET). The Internet Engineering Task Force. Available at http://www.ietf.org/html.charters/manet-charter.html. Accessed on 05/03/2002.

Malki D., Birman K., Schiper A., and Ricciardi A. 1994. Uniform actions in asynchronous distributed systems. Proceedings of the Thirteenth ACM Symposium on Principles of Distributed Computing, ACM press, San Diego, CA, pp. 274-283.

Misra P. 1999. Routing protocols for ad hoc mobile wireless networks. Ohio State University, Columbus, OH. Available at http://www.cis.ohio-state.edu/~jain/cis788-99/adhoc_routing/. Accessed on 05/07/2002.

Perkins D., Hughes H., and Owen C. 2002. Factors affecting the performance of ad hoc networks. IEEE International Conference on Communications, IEEE Communications Society, New York, NY, pp. 2048 –2052.

Rodrigues L., Guo K., Sargento A., Van Renesse R., Glade B., Verissimo P., and Birman K. 1996. A transparent light-weight group service. Proceedings of the 15th IEEE Symposium on Reliable Distributed Systems, IEEE Computer Society, Niagara-on-the-Lake, Canada, pp. 130-139.

Roman G., Huang Q., and Hazemi A. 2000. On maintaining group membership data in ad hoc networks. Technical Report # wucs-00-26, Department of Computer Science, Washington University, St Louis, MO.

Royer E. and Toh C. 1999. A review of current routing protocols for ad hoc mobile wireless networks. IEEE Personal Communications, 6(2): 46-55.

Skeen D. 1982. Crash recovery in a distributed database system. Technical Report # UCB/ERL M82/45, Electronics Research Laboratory, University of California at Berkeley, Berkeley, CA.

Van Renesse R., Birman K., and Maffeis S. 1996. Horus: A flexible group communication system. Communications of the ACM, 39(4): 76-83.

Zhou L. and Haas Z. 1999. Securing ad hoc networks. IEEE Networks Special Issue on Network Security, 13(6): 24–30.

BIOGRAPHICAL SKETCH

Pushkar P. Pradhan received his Bachelor of Engineering degree in 2000 from the University of Mumbai. He received his Master of Science degree in computer science from the University of Florida, Gainesville, in 2002. His research interests include applications for ad hoc networks, distributed systems and mobile commerce.