A Stability Criteria Membership Protocol for Ad Hoc Networks*

Juan Carlos García, Stefan Beyer, and Pablo Galdámez

Instituto Tecnológico de Informática, Universidad Politécnica de Valencia,
46022 Valencia, Spain
{juagaror,stefan,pgaldam}@iti.upv.es

Abstract. *Consensus* is one of the most common problems in distributed systems. An important example of this in the field of dependability is *group membership*. However, *consensus* presents certain impossibilities which are not solvable on asynchronous systems. Therefore, in the case of group membership, systems must rely on additional services to solve the constraints imposed on them by the impossibility of consensus. Such additional services exist in the form of *failure detectors* and *membership estimators*.

The contribution of this paper is the upper-level algorithm of a protocol stack that provides *group membership* for dynamic, mobile and partitionable systems, mainly aimed at mobile ad hoc networks. *Stability criteria* are established to select a subset of nodes with low failure probability to form stable groups of nodes. We provide a description of the algorithm and the results of performance experiments on the NS2 network simulator.

Keywords: ad hoc networks, distributed systems, consensus, group membership, stability criteria.

1 Introduction

Distributed agreement or *consensus* is a well-known problem in distributed systems, in which a set of nodes must agree on a value (or a set of values). Solving the consensus problem allows facing other closely related problems, such as group membership and provides the basis to implement dependability protocols, such as reliable group communication or replication protocols. Informally speaking, all processes of the system propose a value and must reach an unanimous and irrevocable decision about one of the proposed values.

However, the development of distributed protocols in asynchronous systems [1] is a difficult task due to many restrictions and impossibilities [2,3]. The authors of [4,5] show that the problem of consensus and membership cannot be solved in asynchronous distributed systems in which nodes can fail. In mobile ad hoc networks these problems are increased due to the inherently unreliable nature of ad hoc networks [6,7]. In this kind of

* This work has been partially supported by EU FEDER and Spanish MEC under grant TIN2006-14738-C02-01 and by EU FEDER and Spanish MICINN under grant TIN2009-14460-C03.

R. Meersman, T. Dillon, P. Herrero (Eds.): OTM 2009, Part I, LNCS 5870, pp. 690–707, 2009.

network, nodes can move, suffer delays in packet delivery and connect or disconnect at any moment. Furthermore, battery use and bandwidth limits, combined with the broadcast nature and the related tendency for network flooding of these wireless networks create further complications for protocols aiming at consensus or consensus approximations.

In order to avoid the theoretic impossibility of obtaining a deterministic solution, researchers have studied mechanisms that add a certain level of synchrony to the asynchronous model, or use probabilistic proposals of the algorithms. One of the most commonly used mechanism its to rely on an *unreliable failure detector* [8]. A similar solution, often employed in wireless networks, is the use of *collision detectors* [9].

Group membership is a particular case of the consensus problem. Group membership protocols install the same *system view* on all nodes that form the system. If the protocol detects new changes (a node wants to join or leave the network) it must start a *view change* process to install the new *system view* on the nodes of the group. This critical dependability service is typically used to build group communication middleware or replicated applications.

In this paper, we propose a group membership protocol for ad hoc networks that relies on a set of *stable* nodes provided by a *membership estimation* service. The members of the membership view are selected by *stability criteria* [10]. These parameters determine which nodes are the most reliable ones to form a membership group. As the system grows more stable, groups merge. We believe this model to be well suited for mobile environments for two main reasons: Firstly, the system copes well with the frequent occurrence of partitions inherent in MANETs, as it is likely that partitions occur along the less stable connections and individual stable groups remain intact. Secondly, in most dependable applications only stable nodes are of interest for hosting dependability services and, in general, for ensuring algorithms progress. For example, in the context of replication our membership service also serves as a replica placement algorithm, identifying stable nodes.

2 Related Work

A lot of bibliography on ad hoc networks exists. The characteristics of these networks and the problems they present are well described in [11,12]. Ad hoc networks technology may be used to deploy distributed applications. In [6,7] problems that appear in MANET's for distributed applications are discussed. Routing protocols are an essential service for this kind of networks and have a large impact on the performance of distributed algorithms, such as membership protocols. The most commonly used protocols are described in [13,14,15].

Consensus is one of the most common problems studied in distributed systems. There are many restrictions and impossibilities to be solved in asynchronous systems as shown in [5,4,3]. In order to help to solve consensus, additional mechanisms are often employed as *failure detectors*. Their properties are shown in [8,16]. A related solution for ad hoc networks, *collision detectors*, is proposed in [9] . Some *consensus* algorithms for ad hoc networks based on the above solutions are described in [17,18]. *Group membership* for partitionable wired networks is a consensus-based problem that has been widely studied in [19,20,21,22]. Finally, different solutions for the *MANET group membership*

problem are studied in [23,24,25,26,27]. These works rely on neighbourhood or locality parameters to discover the members that can join to the group. Nevertheless, our proposal relies on other different criteria selected by the applications that choose the most stable nodes in the system. This allows to form "multi-hop" groups in a completely transparent way.

3 Model/Architecture

We use a protocol-layered architecture to provide *group membership* in ad hoc networks. This stack allows isolation of the different services that form the implementation of the final membership protocol. We observe three different services: the *routing service* is located at the bottom of the stack. Routing is an essential service in ad hoc networks, as it allows communication amongst the nodes of the network. It offers basic primitives to the upper level services to send/receive packets to/from other nodes. Although routing protocols do not usually offer full guarantees of delivery, they are capable to route packets in multi–hop scenarios. There are a wide range of available protocols, classified by their behavior and functionality. In our system we use the DSR *reactive* routing protocol. We use this pure-reactive routing protocol due we have made some optimizations to decrease its power consumption as we showed [28].

The *membership estimation service* is located at the middle of the stack. It monitorizes the system to provide an *estimated* composition. The service may provide some spurious faults or an incomplete view of the system, but it is an essential service since it allows to start communication with other discovered nodes of the system. Routing protocols do not usually offer *membership (estimation)* information. Furthermore, in our architecture this service also provides information about the stability conditions of nodes (criteria that nodes must fulfill to be considered part of a group). Basically, the algorithm works sending broadcast heartbeat messages with process estimation table every T_g time with a probability P_g. Processes that receive this messages update their estimation tables with the new information and decides to propagate the new changes with probability P_s. One process is considered failed if his heartbeat is not updated after T_f time. More information about the membership estimator and our underlying stack services can be found in [29,30].

Finally, the *membership service* is located at the top of the stack. This service installs the same *system view* (i.e. a subset of all nodes of the system) on each node that satisfies some *stability criteria*. The objective of these *stability criteria* is to select nodes that have low probability of failure or crash in the future ensuring that the *system view* does not include many unreliable nodes. We called *Membership Partition* to our *membership service*. This service can be used by an upper level protocol/application that is noticed about the strict view changes that appear in the system, managing and acting in consequence. For example, this service may be useful for an upper-level *group communication* protocol.

4 Stability Criteria

Our group membership protocol for ad hoc networks relies on a set of *stable* nodes provided by the low-level *membership estimation* service. This set of nodes is selected

by a chosen *stability criterion*. This parameter determines which nodes are the most reliable ones to form a membership group. Ad hoc networks have inherent problems like the possibility of message loss and possible occurrence of node failures. We must choose a *stability criterion* that ensures that all nodes will be running during the new view installation process of the membership protocol in order to satisfy the *consensus* (all nodes have the same view) amongst them. Furthermore, the choice of the appropiate parameter will be influenced by the needs of the application that uses the membership protocol. Next we describe some parameters that we can use as stability criteria.

4.1 Stability Criteria Based on Time

One kind of parameter to ensure the *stability* is node activity time on the network. In our approach, we use a membership estimation service that provides a *view estimation* of the system. In this service, nodes interchange messages that contain information about them. If a node emits a lot of messages during the network lifetime there is high probability that the node does not suddenly disconnect or move out of reach and can form part of a *stable nodes* subset.

The time that a node is running (that is, the time that the membership estimator has detected its presence) can be used as *stability criterion*. Nevertheless, the choice of how long a node needs to be marked as *stable* is a difficult task. For example, if we choose a low value a lot of not fully reliable nodes may enter the system.In many ad hoc applications, nodes connect-disconnect frequently due to a variety of reasons. In certain cases, some nodes only require a brief connection to retrieve a system state and are switched off again to economize their power consumption. In other cases, nodes in movement may fastly transit a partition emitting few membership estimation packets. In this case the *stable set* of nodes may constanly suffer of changes and the strict membership protocol may install a lot of views that may contain untrusted nodes.

On the other hand, if we choose a large value for the time parameter, the inclusion of new nodes in the *strict system view* is delayed and these nodes may not participate inmediately in the system. This situation may be critical for certain problems. For example, a distributed application that replicates data amongst a few nodes that suffer from low battery level may need to reallocate the state of the system to other nodes. In this case, new nodes that enter in the system (and have a pre-supposed good level of battery power) may not be considered stable in time to participate in this proccess and the application may crash.

In what follows, we propose different choices to estimate the time value that may be used to select the set of stable nodes.

- $0 - Time$: this is the most basic parameter selection. In this case, we imediateley promote all nodes that the membership estimator service detects to the *stable subset* (at 0 seconds after its detection). In essence, the service acts as a classic membership protocol. New nodes that enter the system and are detected by the others are immediately included in the *stable view* of the system. This option provides most possible updated system view but we need be aware that we are likely to overload the membership protocol.
- $N - Time$: in this case a node is marked as *stable* if it has been at least N seconds in the system (that is, membership estimation service detected it N seconds ago).

The value of N can be tuned dynamically, adjusting it to possible variations of the network environment. For example if an excessive amount of view changes are observed N can be increased.

– $N(T_g/P_g) - Time$: finally, we can use a more complex *stability criterion* which involves two parameters of the *membership estimation* service. T_g refers to the time interval after which a node decides to propagate a message with a probability P_g. We can say that it is highly probable that every T_g/P_g time interval a node propagates a message. In other words we can use as a *stability criterion* nodes that approximately emit N or more packets since they are alive.

4.2 Stability Criteria Based on Distance

Nowadays, a lot of mobile devices incorporate a *Global Positioning System* or *GPS*, that allows calculating the position of the device on the Earth. Obviously, applications on such nodes can benefit from the location device, for example to obtain distances from the actual position to other positions. We may use this information to provide a *stable* set of nodes. The membership estimation can add the *position* data to the packets to be propagated along the system. This allows to know the last position of the node. We can define different uses of the *positioning* data to select the most *stable* nodes:

– **Based on a fixed or reference point:** certain application may require a group of nodes to be near a fixed point, that may be a physical point. If the nodes interchange position information they can calculate the distances to the reference-prefixed point.
– **Distance-Radio x:** if the membership estimator interchanges data about *position* it can select which nodes that are inside a x range of meters. This may express the nodes that are going to be in the radio range of the node, decreasing the probability of communication problems.
– **Moving from/to node:** finally, we can use the *position* data to calculate a node's most recent movement and direction. With this information, we can discard nodes that are moving away from the coverage area. In fact, there are optimizations for routing protocols that use this heuristic to decide to which node to route a packet.

In general, the precision of the above calculations will be strongly related to the data update frequency. One of the characteristics of Ad Hoc Networks is the possibility of node mobility. If we have a very dynamic system with low frequency of data refreshing the information can become obsolete quickly. As commented above, an additional device that provides the location is needed by the nodes.

4.3 Stability Criteria Based on Energy Level

One of the main constraints of ad hoc networks are the limited battery power of the participants. Usually, a heterogenous range of mobile devices form the system. These devices rely on a battery in order to provide the energy to nodes. Nodes may have a diverse range of battery types that differ in terms of capacity and durability.

We can define the next criteria:

– **Energy level of the devices:** the membership estimation service can propagate the power level of the different nodes, in order to determine the most *powerful* nodes.

This means that we can form a subset of *stable* nodes that includes only nodes that have at least a minimum level of energy.
- **Signal strength:** when a node receives a message it can measure the power of the signal strength. This measure may be important because the reduction of the normal (or latest) transmission's signal power may mean that the node is exiting from the radio coverage or the node may have low battery level.

4.4 Stability Criteria Based on Application Parameters

Finally, we can introduce a different criteria to select the most convenient nodes of the system. The application that is running on the system may select the nodes depending on its necessities. This may be employed to form a strict group for the particular application necessities.

5 Membership Protocol Description

A classic membership protocol tries to install the same view on a set of nodes of a group. Groups are formed by the nodes that explicitly join or leave the group. However, in ad hoc networks nodes join to form the network itself for a certain purpose. Due to dynamic nature of such a system (nodes can move, disconnect/reconnect or become unreachable) it is impractical to compose a *strict system view*. For this reason, we reduce the *strict system view* to the more reliable nodes in order to ensure a good performance of the upper-level applications and do not consume excessive bandwidth or power by the protocol. In this case, we change the definition of a group from the classic concept of nodes that want to join to a group (by means of *join/leave* calls) to the more reliable nodes, selected by stability criteria, forming implicit groups of stable nodes (without any explicit call to *join/leave* primitives).

The behavior of our proposed membership protocol (that we called *Membership Partition*) is based on the concept of a *partition* P. Every node that runs the protocol is part of a partition P. The view of a node is formed by the nodes that are in this partition; that is, $view_i = P_i$. The minimun size of a partition ($size(P_i)$) is 1 which represents an isolated group (node) that only contains the own node identifier $P_i = \{id\}$. Each node has an integer value called *heartbeat* that is incremented every time it tries to perform a new view installation process. Each view has a view identifier $< viewID >= \{id, heartbeat\}$ consisting of the node identifier that starts the new view installation and its own heartbeat. The heartbeat of the initiator node is increased at the begin of the view installation process.

The algorithm seeks to join groups of nodes to form a new partition P' with the new view identifier and containing the merge of the partitions. We define a partition as the set of nodes that have the same view identifier, that is, are forming a group.

5.1 Stable Nodes

The membership protocol relies on a membership estimation service that is responsible for maintaining an estimation (possibly different amongst all nodes) of the system composition, but also to provide information on which nodes are considered locally *stable*

based on a certain set parameters. Each node has access to its own estimation and to the list of nodes it considers *stable*. We define the function $stable()$ which returns the set of stable nodes provided by the membership estimation. The Stable subset can be defined as $Stable_i = stable(Estimation_i)$. This set contains the nodes that satisfy the *stability criteria* on the local node.

5.2 Node Roles

When the membership protocol tries to install a new system view, nodes can adopt three different roles. The first role is the `initiator` node. This role is adopted by a node that detects a change in its set of *stable nodes*. This `initiator` is responsible for starting a new view installation process. The second role is `responder` node. This role is adopted by a node that receives a membership table request message from an `initiator`. Finally, the `nip` (node in partition) node role will be adopted by the rest of nodes in the view intallation process.

5.3 Inclusion Process

When a node detects the appearance of a new *stable* node it starts the partition merge or inclusion process. The detector node takes the `initiator` role and requests from the newly detected node its membership table (`MT_REQUEST` message). The newly detected node takes on the role of `responder` node. Once the `initiator` node has the membership table, it merges the two tables and generates the new proposal. This proposal is sent to the nodes in its partition (that take the `nip` role) and to `responder` node (`MT_PROPOSE` message). The responder propagates the proposal to its partition nodes (which also take the `nip` role).

Once the proposal has been received by each node in both partitions, these nodes respond with an *acknowledge* (or *no acknowledge*, it depends on his own decision about new proposal) message to their respective `initiator` or `responder` node. Both nodes wait to receive the messages from nodes in their own partition. If the `responder` node has collected enough *acknowledge* messages based on a *decision algorithm* it communicates this notice to the `initiator`. If the `initiator` node has been notified by the `responder` and it also has collected enough acknowledge messages from nodes in its partition, it then proceeds to propagate an `APPLY_TABLE_ACK` message to the nodes in its partition and to the `responder` node that re-sends the message to its own `nip` nodes. All nodes that receive this message will apply the proposal and establish it as the new view, ending the process. Finally, to avoid the possible coincidence that various nodes start the inclusion of a node in the group, we can add an additional probabilistic P_{change} parameter that reflects the probability to start the process every time that a new node is detected. This parameter aids to avoid some blocking conditions amongst nodes that simultaneously start the inclusion process.

5.4 Exclusion Process

The exclusion process is similar to the inclusion process with the exception that in this case the `responder` node role does not exist. The node that detects a node that is no longer *stable* sends a new *view* proposal to the nodes in its partition. These nodes accept

(or not) the new proposal and reply to the initiator node . When the initiator node has collected enough *acknowledge* messages, it sends an apply table message to all nodes in the partition (nip), in order to apply the new proposal. As we showed above, in order to avoid the possible coincidence that various nodes start the exclusion of a node in the group we can use P_{change} parameter avoiding the blocking conditions amongst nodes.

5.5 Node States

The algorithm is defined by the phases or states that nodes transit during the view installation process. Nodes can be in one of these states:

- NORMAL: this state will be adopted by all nodes when they are not involved in an update view process.
- PROPOSE_NIP: this state will be adopted by the nodes that are not an initiator or responder node in the view installation process. Once these nodes receive a view proposal, the nodes enter this state storing the new proposal and recognizing the source node with a PROPOSED_TABLE_ACK message. Eventually, nodes return to NORMAL after applying the new proposal when an APPLY_TABLE_ACK message is received.
- PROPOSE_I: this state will be adopted by a node when it receives a response message of type MT_REPLY, which contains the membership table of the responder. The node sends the new proposal to all nodes in its partition and to the responder node. The node will stay in this state until nodes in its partition respond with an acknowledge message passing to a DECIDE_I state.
- DECIDE_I: this state will be adopted by the initiator node that has received a PROPOSED_TABLE_ACK message of all nodes in its partition. The node will be kept waiting for a DECIDE_TABLE_ACK message from the responder. After this event, the node changes back NORMAL state after sending an APPLY_TABLE_ACK message to all the nodes in its partition and to the responder node.
- PROPOSE_R: this state will be adopted by the responser node when it receives the new view proposal. Next, it propagates the new proposal to all the nodes in its partition and waits for their response.
- DECIDE_R: this state will be adopted by the responder node when it receives the responses of all nodes in its partition. In this state, the node waits for a RESPONDER_APPLY_TABLE message from the initiator. The node applies the proposal and sends an APPLY_TABLE_ ACK message to all the nodes in its partition. After that, it enters NORMAL state.

5.6 Timers

We established timers in order to determine if the process can continue its execution. If a timer expires and the protocol has not completed the phase, the process is aborted and nodes return to the NORMAL state. PROPOSE_t is the maximum time that a node can be in a PROPOSE state, whereas DECIDE_t is the maximum time that a node can be in a DECIDE state.

5.7 Consensus Amongst New Proposal

When the `responder` and `initiator` nodes send the proposal to the nodes in their partition, they may respond with two message types, *acknowledge* or *no acknowledge*. The `responder` and `initiator` execute a *consensus* protocol with received messages in order to continue or stop the process. We propose three kinds of *consensus* variants: **strict** in which all nodes must reply with an *acknowledge* message to continue the new view installation process. **Medium** at least X acknowledge messages must be received for the process to continue (usually $X = (nodesinpartition)/2$). Finally, **soft** requires that at least one node sends an *acknowledge* message, in order to continue.

5.8 Additional Functions

To ease the readability of the algorithm, we provide some functions that are used in different points of the algorithm's code:

- `setProposal()`: this function sets the received proposal identifier in the proposal variable states of the process.
- `isSameProposal()`: this function determines if the packet received is referred to the same proposal that process is attending.
- `decideProposal()`: this function evaluates the received proposal and determines an *acknoledge* or *not acknoledge* response. The decision amongst them may be configured by the application constrains.
- `decision()`: this function determines if the process has suficient *acknoledge* response to continue the view installation process.

5.9 Algorithm

The following is the pseudo-code of the algorithm:

```
init
   initiatorNode_=NULL; responderNode_=NULL;
   proposal_=EMPTY; responserDECIDEACK_=false;
   heartbeat=1; state_=NORMAL;
   viewID_={i,heartbeat}; view={i};
endInit

event newStableNode (node)
   heartbeat_++; responderNode_=node;
   responderDECIDEACK_=FALSE;
   send(responderNode_,MEMBERSHIP_TABLE_REQUEST);
endEvent

event unstableNode (node)
   heartbeat++;
   initiatorNode_=ownNode;
   state_=PROPOSE_I;
   deleteProcess_=true;
   responderDECIDEACK_=true;
   createNewProposalWithoutNode(node);
   sendProposalToNodesInPartition();
   if aloneInPartition_ then
       applyProposal();
       responderNode_=initiatorNode_=EMPTY;
       state_=NORMAL; responderDECIDEACK_=false;
   endif
endEvent


event receivedPacket (packet)
   if state_==NORMAL then
      if packet==MT_REQUEST then
         state_=PROPOSE_R;
         initiatorNode_=packet->initiatorID();
         proposalID_=(packet->initiatorID(),packet->HB());
```

```
            sendTo(initiatorNode_,MT_REPLY);
        endif

        if packet==MT_PROPOSE and isSameProposal(packet->IDView()) then
            state_=PROPOSE_NIP;
            setProposal(packet->proposal)
            initiatorNode_=packet->initiatorID();
            sendTo(packet->source(),decideProposal(ACK,NACK));
        endif
        if packet==MT_REPLY and responderNode_==packet->source() then
            initiatorNode_=packet->initiatorID();
            state_=PROPOSE_I;
            newProposal=mergePartitions();
            setProposal(newProposal);
            sendToPartitionNodes(newProposal);
            sendTo(responderNode_,MT_PROPOSE);
            if aloneInPartition_ then
                state=DECIDE_I;
        endif
    endif
    if state_==PROPOSE_NIP then
        if packet==APPLY_TABLE_ACK and isSameProposal(packet->IDView()) then
            applyProposal();
            responderNode_=initiatorNode_=empty; state_=NORMAL;
        endif
    endif
    if state_==PROPOSE_I then
        if packet==PROPOSED_TABLE_ACK and isSameProposal(packet->IDView()) then
            setMessageReceivedFromNode(packet->source());
            if decision()==true then
                state_=DECIDE_I;
        endif
        if packet==DECIDE_TABLE_ACK and responderNode_==packet->source() then
            responderDECIDEACK_=true;
    endif
    if state_==DECIDE_I then
        if responderDECIDEACK_==true or (packet==DECIDE_TABLE_ACK and responderNode_==packet->source()) then
            sendToPartitionNodes(APPLY_TABLE_ACK);
            applyProposal();
            if not deleteProcess_ then
                sendTo(responderNode_, RESPONDER_APPLY_TABLE);
            responderNode_=initiatorNode_=empty;
            state_=NORMAL; responderDECIDEACK_=false;
        endif
    endif
    if state_==PROPOSE_R then
        if packet==MT_PROPOSE and initiatorNode_==packet->source() then
            setProposal(packet->proposal);
            sendToPartitionNodes(packet->proposal);
            if aloneInPartition_ then
                state_=DECIDE_R;
                sendTo(initiatorNode_,DECIDE_TABLE_ACK);
            endif
        endif
        if packet==PROPOSED_TABLE_ACK and isSameProposal(packet->IDView()) then
            setMessageReceivedFromNode(packet->source());
            if decision()==true then
                state_=DECIDE_R;
                sendTo(initiatorNode_,DECIDE_TABLE_ACK);
            endif
        endif
    endif
    if state_==DECIDE_R then
        if packet==RESPONDER_APPLY_TABLE and packet->source()==initiatorNode_ then
            sendToPartitionNodes(APPLY_TABLE_ACK);
            applyProposal();
            responderNode_=initiatorNode_=empty; state_=NORMAL;
        endif
    endif
endEvent

event timerTimeout()
    initiatorNode_=responderNode_=empty;
    responderDECIDEACK_=false; state_=NORMAL;
endEvent
```

6 Experiments

We have performed a set of experiments to test our membership protocol which we call *Membership Partition*. First, we identify some scenarios and configurations that can help us to extract some conclusions on the energetic performance and bandwidth usage of the protocol.

General simulation configuration. We used the NS2 network simulator [31] to perform our tests. This tool implements the DSR routing protocol. The membership estimator and *Membership partition* protocols were implemented by us. We made a set of 20 simulations per configuration and scenario type, varying the number of nodes from 4 to 20. Simulation time was established to 300 seconds.

We identify two kinds of scenario sizes. *Single–hop* scenarios consist in a $100 \times 100\ \mathrm{m}^2$ flat grid. In this case, all nodes are inside radio signal strength of each other. These scenarios are used to simulate enclosed settings, such as meetings or museums in which network or visitor-guide applications are running. *Multi–hop* scenarios consist in a $400 \times 400\ \mathrm{m}^2$ flat grid. These scenarios are used to simulate "open-air" settings, in which network communication is needed.

Furthermore, for each of the above scenarios we use two different mobility patterns: *static* in which nodes do not move during the simulation time and *dynamic* in which nodes move following a Random WayPoint Model. In this case, nodes move randomly to a destination at a maximum speed of 3*m/s* and stay there for a *pause time* of 25 seconds before changing speed and direction. We employethis model due we want to test the performance of our algorithm under bad scenarios (nodes may have random and "un-logical" movements that may cause different problems to establish the set of stable nodes). Finally, we set the values for the energy consumption model to be approximately the same as those of the real network interface *Lucent WaveLAN*.

Membership estimator configuration. The most important parameters of our *membership estimator* protocol are: P_g which indicates the probability that a node broadcasts its table after it increments its own heartbeat (we set this parameter at 0.7). P_s indicates the probability that a node that receives a message rebroadcast the updated table if there were changes (we set this parameter at 0.5). T_f indicates the time to consider a node as failed if its *heartbeat* is not updated after this time (we set this parameter at 4s). Finally, T_s indicates the time that a node waits to listen for packets during initialization (we set this parameter at 2s).

Membership Partition configuration. The *Membership partition* protocol needs to know which *stability criteria* must apply to determine which nodes can join the group. In the simulations we use the following criteria:

- `GOD CRITERION`: which emulates a *perfect* membership estimator. This means that we can provide an exact and precise information of the system composition at every moment without spurious mistakes. We can use it to model the behavior of a classic membership group protocol as it would be used in a wired environment. Unfortunately this membership estimator is impossible to implement in practice.
- `NO TIME CRITERION`: in this case we use a basic *Membership Partition* protocol to join every node that appears in the estimation without any restriction or condition. The strict membership protocol acts in this case as a classic membership group protocol over a wireless network. We use this configuration to compare our approach to that of classic membership services.
- `N TIME CRITERION`: in this case the *Membership Partition* protocol only selects nodes that have been in the membership estimation table at least N seconds.

– `N TG PG TIME CRITERION`: in this case the *Membership Partition* protocol only selects nodes that have been in the membership estimation table at least $N * (T_g/P_g)$ (T_g and P_g are membership estimator parameters) time seconds. That is, nodes have a high probability that they send about N packets.

In our experiments we employed (sorted from weak to strong stability conditions) the following configurations: *GOD CRITERION*, *NO TIME CRITERION*, *5 TIME CRITERION*, *5 TG PG TIME CRITERION*, *10 TIME CRITERION* and *10 TG PG TIME CRITERION*. $Propose_i$ and $Decide_i$ timers are established to 1s. The timers are used to unblock the view installation process in case that one of the phases does not progress correctly, usually due to packet loss. In the simulations the *Membership Partition* protocol starts 5s after the *membership estimation* service, in order to initialize the system with some values in the tables. We employ a *soft* type decision for nodes that switch to an *unstable* state. This means that the group excludes a node when at least one node observes the transition to *unstable*.

7 Results

We have performed experiments to compare the power consumption of the *Membership Partition* protocol in different configurations. In Figures 1 and 3 we can see the results for each configuration of the static and dynamic single–hop scenarios.The results show that the use of *stability criteria* decreases the power consumption. If we use a strong *stability criterion* we restrict the conditions for a node to join the partition, decreasing the number of *inclusion processes*. This feature seems to be undesirable, but in a dynamic environment with strong constraints, such a trade-off may be beneficial, as the systems suffers less view changes and upper-level applications do not suffer delays in their progress due to the permanent process of inclusion/exclusion of nodes. Moreover, if we employ a strong *stability criterion* we ensure that groups are formed by the most reliable nodes of the system, allowing us to make a good choice for hosting reliable services.

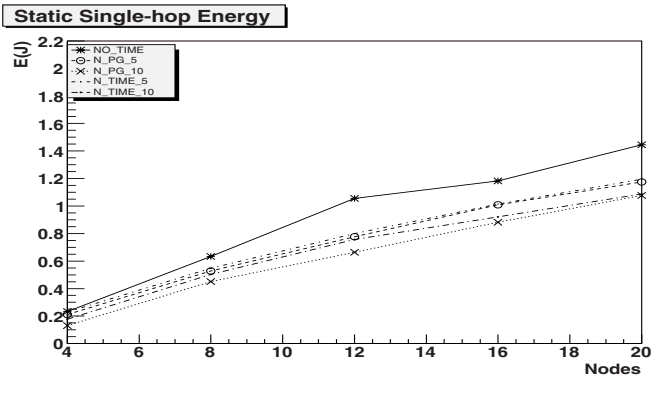

Fig. 1. Static Single–hop energy

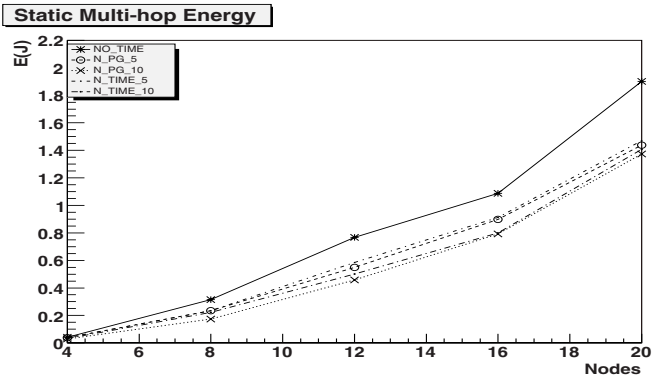

Fig. 2. Static Multi–hop energy

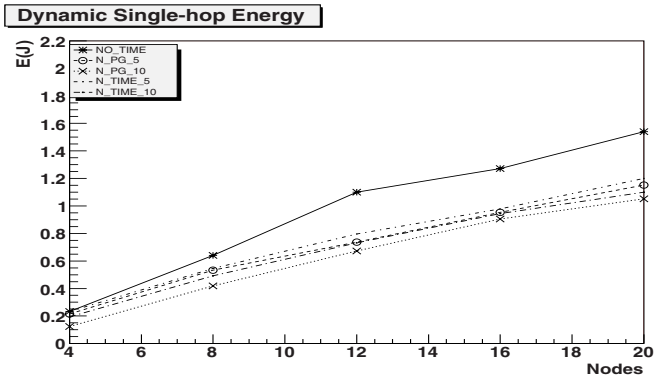

Fig. 3. Dynamic Single–hop energy

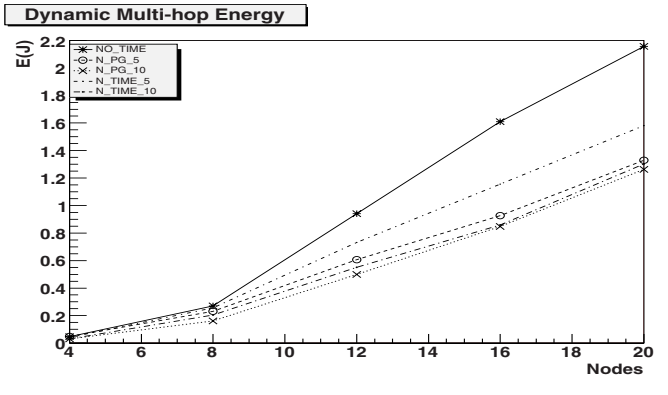

Fig. 4. Dynamic Multi–hop energy

We can see in Figures 2 and 4 the results for each configuration of the static and dynamic multi–hop scenarios. In this case, we observe that for a low number of nodes, we obtain lower power consumption than in single–hop scenarios, but a higher power consumption when the number of nodes increases. This is probably due to the possible existence of network partitions with a low number of nodes. If these occur, nodes that run *Membership Partition* form small partitions requiring less packets to interchange in every process. However, as the number of nodes grows reachability of nodes/partitions may increase, but the dynamicity causes the membership estimator to produce spurious failures due to the difficulty of maintaining its tables updated correctly. If we do not use any *stability criteria* (as *NO TIME*), we are essentially using a traditional membership service. We have included this experiment in the figures as a reference point to show the significant improvement in power consumption provided by our protocol.

To complement the results obtained in the study, we have performed simulations to compare the performance of the protocol in a single–hop scenario. We can model this

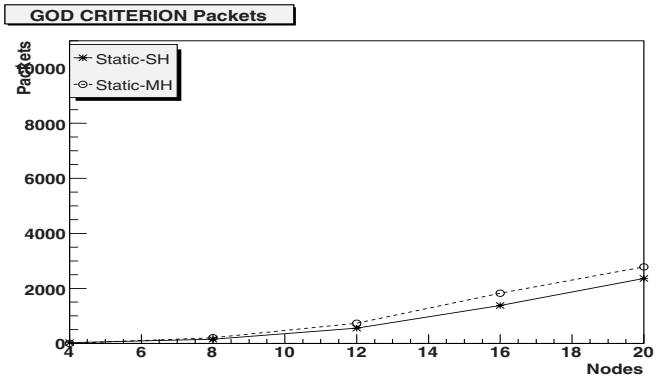

Fig. 5. GOD CRITERION Packets

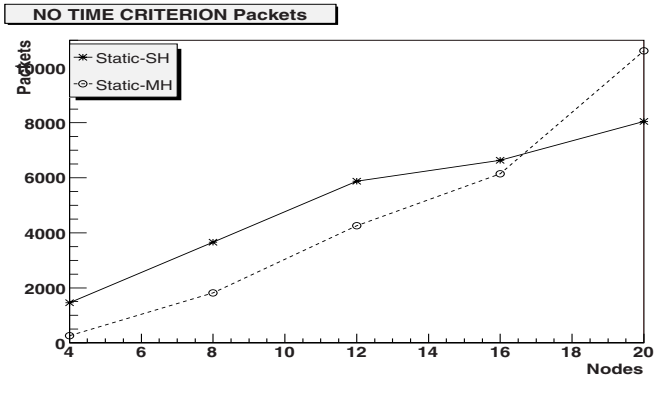

Fig. 6. NO TIME CRITERION Packets

scenario like a wired scenario in which the topology of the system is known in the case of the static experiments, whereas we can model the dynamic scenario as a conference room scenario where nodes enter and leave. We have performed experiments to compare the bandwidth usage (in terms of number of packets received/sent by a node during the simulation) of the *Membership Partition* protocol using the *NO TIME*, *N TG PG 10* and *N TIME 10* approaches comparing them with a *perfect system* (*GOD CRITERION*), that is a system that integrates a perfect *membership estimator* that retrieves the exact composition of the system without mistakes at any moment.

We can see in Figures 5, 6, 7 and 8 the results for the packets generated in each configuration. We observe that as expected *GOD CRITERION* experiments obtain the best results for all cases. This due to the system being fully connected and the algorithm rapidly forms a unique and big partition as the result of the merge of the small partitions and remains without changes during the simulation, decreasing the maintenance of the

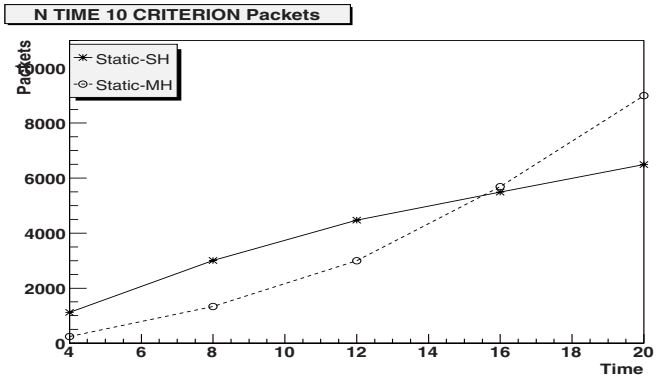

Fig. 7. N TIME 10 CRITERION Packets

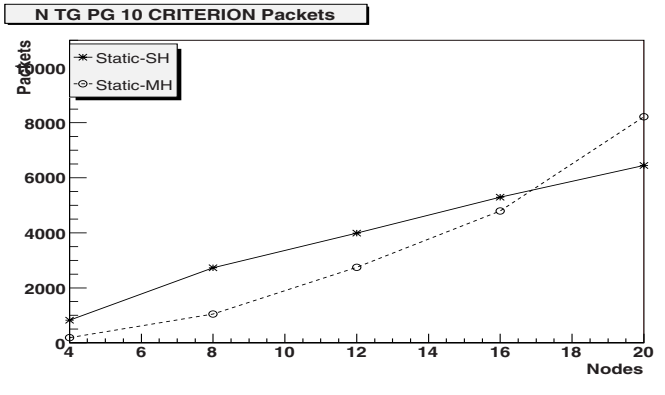

Fig. 8. N TG PG 10 CRITERION Packets

system (new view change processes) to the minimum. When we see the results for a more realistic environment (using data from a real membership estimator), we observe that the number of packets are higher. This due to the results provided by the *membership estimator* which cause a lot of spurious failures. Nevertheless, if we employ some restrictive *stability criteria* we can reduce the bandwidth usage of the protocol, as the system's groups are formed by *reliable* nodes.

Furthermore, we observe that in the *perfect system* in multi–hop scenarios more packets are generated than in single–hop scenarios, whereas in the other cases, multi–hop scenarios sent fewer packets than single–hop, when the number of nodes is low, but inverting this behavior when the number of nodes increases. When the number of nodes increase, the probability of isolation is reduced and nodes are more easily detected and updated by the *membership estimator*.

8 Conclusion and Future Work

In this work we presented a protocol stack which provides a group membership solution for ad hoc networks. In this protocol stack we find a strict membership protocol, a membership estimator protocol and a routing protocol. At the top of the layer strict membership generates a consistent view of the system that should be installed in a *stable* group of nodes. *Stable nodes* will be selected by the membership estimator protocol. The objective is to provide a subset of nodes of the system that satisfy certain *stability criterion*, in order to ensure the maximum reliability of the subset. Finally, at the bottom a routing protocol is used by the membership protocol to interchange the messages amongst nodes.

We believe that the selection of a reliable subset of nodes maximizes the probability that nodes do not fail during *view installation*. This reduces the *group membership/-consensus* problem in ad hoc networks. The subset of nodes is selected in function of a *stability criterion* that may be chosen for the particular necessities of the application. We presented results of experiments in which we observe that the use of *stability criteria* can significantly reduce the bandwidth and power consumption of nodes. In future work, we are planning to perform wider studies of particular application scenarios and aim to introduce further *stability criteria*.

References

1. Chockler, G., Keidar, I., Vitenberg, R.: Group communication specifications: a comprehensive study. ACM Computing Surveys 33(4), 427–469 (2001)
2. Schiper, A.: Practical impact of group communication theory. In: Schiper, A., Shvartsman, M.M.A.A., Weatherspoon, H., Zhao, B.Y. (eds.) Future Directions in Distributed Computing. LNCS, vol. 2584, pp. 1–10. Springer, Heidelberg (2003)
3. Fischer, M.J.: The consensus problem in unreliable distributed systems (a brief survey). Fundamentals of Computation Theory, 127–140 (1983)
4. Fischer, M.J., Lynch, N.A., Paterson, M.S.: Impossibility of distributed consensus with one faulty process. J. ACM 32(2), 374–382 (1985)
5. Chandra, T.D., Hadzilacos, V., Toueg, S., Charron-Bost, B.: On the impossibility of group membership. In: Proceedings of the 15th Annual ACM Symposium on Principles of Distributed Computing (PODC 1996), pp. 322–330. ACM, New York (1996)

6. Basile, C., Killijian, M., Powell, D.: A survey of dependability issues in mobile wireless networks. Technical report, LAAS CNRS Toulouse, France (2003)
7. Vollset, E., Ezhilchelvan, P.: A survey of reliable broadcast protocols for mobile ad-hoc networks. Technical Report CS-TR-792 (2003)
8. Chandra, T.D., Toueg, S.: Unreliable failure detectors for reliable distributed systems. J. ACM 43(2), 225–267 (1996)
9. Chockler, G., Demirbas, M., Gilbert, S., Newport, C., Nolte, T.: Consensus and collision detectors in wireless ad hoc networks. In: PODC 2005: Proceedings of the twenty-fourth annual ACM symposium on Principles of distributed computing, pp. 197–206. ACM Press, New York (2005)
10. Mostefaoui, A., Raynal, M., Travers, C., Patterson, S., Agrawal, D., Abbadi, A.E.: From static distributed systems to dynamic systems. In: SRDS 2005: Proceedings of the 24th IEEE Symposium on Reliable Distributed Systems, pp. 109–118. IEEE Computer Society, Los Alamitos (2005)
11. Baker, F.: An outsider's view of manet. Network Working Group Internet-Draft (2002)
12. Frodigh, M., Johansson, P., Larsson, P.: Wireless ad hoc networking, the art of networking without a network. Technical report, Ericsson Review No. 4 (2000)
13. Johnson, D.B., Maltz, D.A., Broch, J.: 5. In: DSR: The Dynamic Source Routing Protocol for Multihop Wireless Ad Hoc Networks, pp. 139–172. Addison-Wesley, Reading (2001)
14. Perkins, C.E., Royer, E.M.: Ad-hoc on-demand distance vector routing. In: WMCSA 1999: Proceedings of the Second IEEE Workshop on Mobile Computer Systems and Applications, vol. 90. IEEE Computer Society, Los Alamitos (1999)
15. Jacquet, P., Mühlethaler, P., Clausen, T., Laouiti, A., Qayyum, A., Viennot, L.: Optimized link state routing protocol for ad hoc networks. In: Proceedings of the 5th IEEE Multi Topic Conference (INMIC 2001), pp. 62–68 (2001)
16. Freiling, F.C., Guerraoui, R., Kouznetsov, P.: The failure detector abstraction. Technical report, Faculty of Mathematics and Computer Science, University of Manheim, Germany (2006)
17. Angluin, D., Fischer, M.J., Jiang, H.: Stabilizing consensus in mobile networks. In: Gibbons, P.B., Abdelzaher, T., Aspnes, J., Rao, R. (eds.) DCOSS 2006. LNCS, vol. 4026, pp. 37–50. Springer, Heidelberg (2006)
18. Wu, W., Yang, J., Raynal, M., Cao, J.: Design and performance evaluation of efficient consensus protocols for mobile ad hoc networks. IEEE Trans. Comput. 56(8), 1055–1070 (2007)
19. Dolev, D., Malki, D.: The transis approach to high availability cluster communication. Communications of the ACM 39, 64–70 (1996)
20. Moser, L.E., Melliar-smith, P.M., Agarwal, D.A., Budhia, R.K., Lingley-papadopoulos, C.A.: Totem: A fault-tolerant multicast group communication system. Communications of the ACM 39, 54–63 (1996)
21. Renesse, R.V., Birman, K.P., Maffeis, S.: Horus: A flexible group communication system. Communications of the ACM 39, 76–83 (1996)
22. Friedman, R., van Renesse, R.: Strong and weak virtual synchrony in horus. In: SRDS 1996: Proceedings of the 15th Symposium on Reliable Distributed Systems (SRDS 1996), vol. 140. IEEE Computer Society, Los Alamitos (1996)
23. Liu, J., Sacchetti, D., Sailhan, F., Issarny, V.: Group management for mobile ad hoc networks: design, implementation and experiment. In: MDM 2005: Proceedings of the 6th international conference on Mobile data management, pp. 192–199. ACM Press, New York (2005)
24. Mohapatra, P., Gui, C., Li, J.: Group communications in mobile ad hoc networks. Computer 37(2), 52–59 (2004)
25. Briesemeister, L., Hommel, G.: Localized group membership service for ad hoc networks. In: International Workshop on Ad Hoc Networking (IWAHN), pp. 94–100 (August 2002)

26. Roman, G.C., Huang, Q., Hazemi, A.: Consistent group membership in ad hoc networks. In: Proceedings of the 23rd international conference on Software engineering, pp. 381–388. IEEE Computer Society, Los Alamitos (2001)
27. Bollo, R., Le Narzul, J.P., Raynal, M., Tronel, F.: Probabilistic analysis of a group failure detection protocol. In: WORDS 1999: Proceedings of the Fourth International Workshop on Object-Oriented Real-Time Dependable Systems, Washington, DC, USA, vol. 156. IEEE Computer Society, Los Alamitos (1999)
28. García, J.C., Beyer, S., Galdámez, P.: Cross-layer cooperation between membership estimation and routing. In: SAC 2009: Proceedings of the ACM symposium on Applied Computing, pp. 8–15. ACM, New York (2009)
29. García, J.C., Banyuls, M.C., Galdámez, P.: Trading off consumption of routing and precision of membership. In: Proceedings of the 3rd International Conference Communications and Computer Networks, Marina del Rey, CA (USA), October 24–26, pp. 108–113. ACTA Press (2005)
30. García, J.C., Bañuls, M.-C., Beyer, S., Galdámez, P.: Effects of mobility on membership estimation and routing services in ad hoc networks. In: Stojmenovic, I., Thulasiram, R.K., Yang, L.T., Jia, W., Guo, M., de Mello, R.F. (eds.) ISPA 2007. LNCS, vol. 4742, pp. 774–785. Springer, Heidelberg (2007)
31. project, T.V.: The NS2 manual. Technical report, ISI (2004), `http://www.isi.edu/nsnam/ns/ns-documentation.html`