Group Membership Protocol: Specification and Verification*

Yuri Gurevich† Raghu Mani‡

EECS Department, University of Michigan
Ann Arbor, MI, 48109-2122, USA

1 Introduction

According to the Evolving Algebra thesis [3], evolving algebras should allow one to specify succinctly any algorithm. There exists substantial evidence confirming this thesis in the case of sequential algorithms (see the annotated bibliography in [3]). In other papers, e.g., [1, 5], evolving algebras are used to specify distributed algorithms. For this paper, we wanted to look at a time-constrained algorithm that does something useful and poses some challenge to specify and verify. Our colleague Farnam Jahanian brought Cristian's article on group membership protocols [2] to our attention. In this paper, we specify and verify one of the protocols presented in that article. It is an interesting protocol to verify as we need to specify and prove both timing as well as functional properties.

Group membership protocols [2, 6, 7] are used mainly to provide fault tolerance for distributed computing services. One possible way of ensuring service availability in a distributed system despite processor failures is to have several servers *cooperate* to provide the service (each such set of servers is termed a *server group*) and to *replicate* information relevant to the service (this is termed *service state information*) at all the sites in the network. For example, if the service in question is a C compiler then the state information may include a list of servers offering this service that are currently alive and information regarding how heavily loaded each of these servers is. The purpose of group membership and other related protocols is to ensure that the state information stored at each group member remains up-to-date and that in the steady state, all group members see the same state information – despite information propagation delays and server failures.

Central to the problem of server-group membership is *processor-group membership* which, to put it briefly, is the problem of achieving global agreement about the set of all correctly functioning processors in the system. Given a solution for the processor group membership problem, it is possible to use it to construct a solution to the server-group membership problem. The protocol we consider in this paper is a solution to the processor-group membership problem in synchronous systems.

*To appear in E. Börger editor, *Specification and Validation Methods for Programming Languages and Systems*, Oxford University Press, 1994.

†Partially supported by ONR grant N00014-91-J-1861 and NSF grant CCR-92-04742.

‡Partially supported by NSF grant CCR-92-04742.

2 Overview of the Protocol

In this section, we describe the assumptions about the system, the protocol itself and finally the goals that the protocol is supposed to achieve. In an attempt to simplify the exposition and make this paper self-contained, we allow ourselves slight changes of terminology.

In the following two subsections, we describe our interpretation of the assumptions made by Cristian about the system. It turns out that not all his assumptions are necessary.

2.1 Synchronous Communication Network

There is a fixed finite collection of processors p, each with its own clock $\mathsf{Clock}(p)$ or Clock_p. You may think about Clock_p as a real-valued function of real time. Cristian assumes that every Clock_p is strictly monotone ("successive readings yield strictly increasing values") and there is a bound on the deviations $\mid \mathsf{Clock}_p(t) - t \mid$. It follows that there is a bound on the skew between any pair of clocks. In our specification, we do not make any assumption about the connection between the clocks and real time. We do not assume directly the existence of a skew bound though a weaker form of that assumption will follow from one of our later assumptions.

Each processor hosts various processes, and processes handle tasks. Processors can be interrupted but a task is run until completion. Typically, a task has a *start deadline* (henceforth, simply *deadline*). Scheduling is assumed to be *earliest deadline first*. A task may not be scheduled well in advance of its deadline – more precisely, if the deadline of the task is d then it may not be scheduled until time $d - d_u$, where d_u is a constant called the *scheduling uncertainty*. A task may, however, be scheduled after its deadline has passed – this can happen when tasks with earlier deadlines take too long to complete, thus preventing the task in question from being scheduled in timely fashion. d_u is one of a series of system related bounds and constants we will be describing in this section. When we say that such and such bound exists we mean that it exists and is known.

Cristian speaks about correct and incorrect processors. The following abstraction seems appropriate. A processor can be *crashed*, *recovering* or *sober*. It is assumed that there is a minimum delay of d_r time units (the *recovery* bound) between the time a processor crashes and the time it next becomes sober. A processor is *correct* at a particular instant of time if it is sober and it has no pending tasks whose deadlines have been exceeded. This latter situation is termed a *performance failure*.

A reliable broadcast mechanism is assumed. It guarantees that every message m broadcast by a correct processor p will be delivered to every processor on the network. The only reason for a scheduled broadcast not reaching all the processors on the network is if the sender crashes or suffers a performance failure. It is also assumed that performance failures can be detected and turned into crashes. We make the same assumptions here.

Moreover, there is a bound on the time taken to carry m over from p to any processor q. The time can be "measured on any processor clock" [2]. It seems a little more natural to measure the time of sending on $\mathsf{Clock}(p)$ and the time of delivery on $\mathsf{Clock}(q)$. We will assume that there is a carry-over bound d_c with the following property: If the sending time of m is t_1 with respect to $\mathsf{Clock}(p)$ and the delivery time is t_2 with respect to $\mathsf{Clock}(q)$ then $0 < \mathsf{Clock}(q) - \mathsf{Clock}(p) \leq d_c$. This, in fact, is the only assumption we make relating the clocks of two different processors and follows from Cristian's assumptions related to time; of course a natural way to satisfy our assumption is to satisfy the time related assumptions of Cristian. Cristian assumes also that messages are seen in order of their deadlines and that two messages with the same deadline are seen in the same order by all correct processors. The latter assumption turns out to be unnecessary.

In this protocol, the only type of message send is a broadcast – therefore, in this document, the terms "send" and "broadcast" will be synonymous.

2.2 Informal Description of the Protocol

Each processor p hosts a *membership server* $MS(p)$ that handles the entry of the processor into a processor-group and helps maintain state information while the processor is alive, and a *broadcast server* $BS(p)$ that periodically sends a broadcast to all the other processors in the system. In addition to these, there may be other processes running on the processor. The only way these can affect the protocol are by using up time and hence delaying protocol related tasks. Tasks relevant to the protocol are – recovering from a crash, processing an incoming message (both handled by the $MS(p)$) and sending a broadcast (handled by the $BS(p)$).

$MS(p)$ maintains the state information stored at p and processes incoming messages. The state information stored at p consists of

- The *group identifier* of p's group – New groups are created each time a processor fails or recovers. In this protocol, a group identifier is a timestamp that indicates when the group was formed.
- Identifiers of the processors in p's current group – the *membership view* of p.

In [2] the state information includes a flag that indicates whether or not p is currently part of a group. We can do away with this as the desired property can be determined from whether or not the group id (or the membership view) is defined.

$BS(p)$ sends broadcasts at periodic intervals – these *heartbeats* inform other processors that p is alive. If p misses a heartbeat then the other processors conclude that the p has failed. The interval between two successive heartbeats is a system-wide constant and is denoted d_h in this document. When $BS(p)$ is scheduled, one of the following two situations can arise.

- $\mathsf{Clock}(p)$ is greater than the deadline of the currently scheduled heartbeat – This means that p has missed a deadline and therefore is, in some sense, not functioning correctly. In such a case, the $BS(p)$ concludes that p has failed and removes p from its current group.
- $\mathsf{Clock}(p)$ is less than or equal to the deadline t of the currently scheduled heartbeat – In this case, when scheduled, the $BS(p)$ sends a present message with timestamp t and sets the time of the next heartbeat to $t + d_h$.

$MS(p)$ operates as follows

- When recovering from a failure, $MS(p)$ initializes the state variables, and broadcasts a new_gp message – this message indicates to all other processors that p is rejoining the system and attempting to form a new group. The timestamp t of this message is equal to the sum of $\mathsf{Clock}(p)$ and a constant d_n which is larger than the maximum network propagation delay – the idea being to ensure that the message is seen by every correctly functioning processor q before $\mathsf{Clock}(q) = t$.
- If $MS(p)$ sees a new_gp message then one of the following two situations can arise.
  - $\mathsf{Clock}(p)$ is greater than the timestamp of the new_gp message – In such a case, $MS(p)$ assumes that p has failed and removes p from its current group.
  - $\mathsf{Clock}(p)$ is not greater than the timestamp t of the new_gp message – In such a case, $MS(p)$ cancels any pending heartbeat, sets the time for the next heartbeat to $t + d_h$ and broadcasts a present message with timestamp t. This present message indicates to all other processors that p is going to join the new group. $MS(p)$ sends no further present messages until the next new_gp message arrives. All other present messages from p are sent by $BS(p)$.

- If $MS(p)$ sees a present message m then it does the following
  - It checks its box of incoming messages for present messages from other processors with the same timestamp as m, removes all of them (including m) and computes the union v of their sender processor ids.
  - If v is identical to the membership view of p then it does nothing else.
  - If v is different from the membership view of p then it makes v the new membership view and sets the group id of p to the timestamp of m.

2.3 Summary of Bounds and Constants

d_u: an upper bound on the uncertainty in scheduling.
d_c: an upper bound on the time taken to carry a message over the network.
d_h: the heartbeat interval.
d_n: the new_gp timestamp increment.
d_r: a lower bound on the time between a crash and a subsequent recovery.

It is assumed that $d_h > d_u$, $d_n > d_c + d_u$ and $d_r > d_h + d_u$.

2.4 Goals of the Protocol

Whenever we mention a time value in the context of a processor p, it will mean the time as recorded by *Clock*(p).

G1. *Stability of local views:* Once a processor joins a group, it stays in that group until either a processor fails or one recovers and attempts to rejoin.

G2. *Agreement on history:* If p and q are joined to a common group g during a run of the system then if the next groups joined by p and q after leaving g are g^p and g^q respectively, and neither processor crashes in the interval between the time it joined g and the time it joined its next group then, $g^p = g^q$.

G3. *Agreement on group membership:* If p and q are joined to the same group and are both alive then their membership views are identical.

G4. *Reflexivity:* If p is alive and joined to a group then its id will be included in its membership view.

G5. *Bounded join delays:* There exists a time constant d_j such that if a processor becomes sober at time t then, by time $t + d_j$, it will join a new group g along with every other processor q that stays correct in the interval $[t, t + d_j]$.

G6. *Bounded failure detection delays:* There exists a time constant d_f such that if a processor belonging to group g fails at time t then, by time $t + d_f$, all the members of g that that stay correct in the interval $[t, t + d_f]$ will join a group g' that does not contain p.

3 The Program

The semantics of evolving algebras are described in [4]. In order to understand the material in this paper, the reader need only read about ground distributed evolving algebras. We use variables in this paper only to make the rules easier to read. Variables are, in fact, not needed and can easily be eliminated.

The algebra is modeled as five module templates. MembershipServer and BroadcastServer model the membership and broadcast servers of the protocol respectively. Scheduler handles recovery from crashes and scheduling of processes, MessageCarrier handles transmission of messages across the network and Custodian handles orderly delivery of messages to the membership server.

The transition rules are all named. We have also named the clauses in these transition rules that we shall be referring to often.

In this section we will observe the following conventions. Variables p and q range over processors; variable m ranges over messages. Abbreviations are written in SMALL CAPS and external functions are written in *Slanted Sans Serif*.

3.1 Vocabulary

We do not describe the vocabulary explicitly since most of it is quite obvious from the program but we explain here some of the less obvious functions and abbreviations.

If t is a term then DEFINED(t) is an abbreviation for $t \neq$ undef.

We define MessageType to be the universe containing two objects present and new_gp. Real is the universe of real numbers, Processor is the universe of processor ids. A message m is a triple $(x, y, z) \in$ MessageType $\times$ Real $\times 2^{\text{Processor}}$; x is $MesType(m)$, y is $Timestamp(m)$ and z is $View(m)$. Let Message be the universe of messages.

Messages have deadlines associated with them. The deadline of a message m (written $Deadline(m)$) is defined as follows.

1. if m is a new_gp message then $Deadline(m) = Timestamp(m)$.
2. if m is a present message then $Deadline(m) = Timestamp(m) + d_n$.

We define the following dynamic functions. $InBox(p) \in 2^{\text{Message}}$ is the set of messages that have been delivered to p but have not been seen by the membership server. $CurMes(p)$ stores the message currently being seen by the membership server. It turns out that this message has the earliest deadline among all current messages. $BCastTime(p)$ gives the time for which the next broadcast is scheduled.

The deadline for a process x (written DLINE(x)) is the minimum among the deadlines of the tasks waiting to be handled by x. The two processes we are concerned with here are the membership and broadcast servers. The relevant tasks handled by the membership server are (i) initializing the internal functions of the processor after recovering from a crash and (ii) handling incoming messages. Each incoming message corresponds to a separate task. The dynamic function $CurMes(p)$ should always store a message with the earliest deadline. DLINE($MS(p)$) abbreviates, therefore, $Deadline(CurMes(p))$. The tasks handled by the broadcast server are broadcast sends. In this protocol, there is at most one broadcast scheduled at a given processor at any given time. DLINE($BS(p)$) abbreviates, therefore, $BCastTime(p)$.

ENABLED($MS(p)$) abbreviates

$$\text{DEFINED}(CurMes(p)) \wedge (\text{DLINE}(MS(p)) \geq \mathit{Clock}(p) - d_u)$$

and ENABLED($BS(p)$) abbreviates

$$(\text{DLINE}(BS(p)) \geq \mathit{Clock}(p) - d_u).$$

Informally, APTPART(p) is the set of messages m in $InBox(p)$ such that $Deadline(m) \geq \mathit{Clock}(p) - d_u$. More formally, define the static function Apt as follows. Given an element $I \in 2^{\text{Message}}$ and two reals t_1 and t_2, it computes an element $J \in 2^{\text{Message}}$ which (viewed as a set) comprises all messages $m \in I$ such that $(Deadline(m) \geq t_1 - d_u) \wedge (Timestamp(m) \geq t_2)$. APTPART($p$)

abbreviates $Apt(InBox(p), \mathsf{Clock}(p), StartUpTime(p))$. Here $StartUpTime(p)$ is the time at which p recovered from its last crash.

3.2 Scheduler(p)

Each processor hosts a number of processes. The Scheduler agent for a processor p handles recovery from crashes and scheduling of processes on the processor. The two processes on p that we are concerned with in this specification are the membership server process $MS(p)$ and the broadcast server process $BS(p)$. These are the two processes that implement the protocol – we assume that the other processes running on the processor do not affect the protocol in any way other than using up time.

Transition Recover

if $Status(p) =$ crashed **then**
    $Status(p) :=$ recovered, $CurProc(p) := MS(p)$
endif

When a processor has recovered from a crash, the only process that can execute is the membership server. Scheduling a process on p is modeled by setting $CurProc(p)$ (the process currently running on p) to the appropriate value.

Transition Schedule

if $(CurProc(p) =$ undef$) \wedge (Status(p) =$ sober$)\wedge$
    $((\text{APTPART}(p) = \emptyset) \vee \text{DEFINED}(CurMes(p)))\wedge$
    $[((x = MS(p)) \wedge (y = BS(p))) \vee ((x = BS(p)) \wedge (y = MS(p)))]$ **then**
    if $\text{ENABLED}(x) \wedge \neg\text{ENABLED}(y)$ **then** $CurProc(p) := x$
    elseif $\text{ENABLED}(x) \wedge \text{ENABLED}(y)$ **then**
        if $\text{DLINE}(x) < \text{DLINE}(y)$ **then** $CurProc(p) := x$
        else $CurProc(p) := y$
        endif
    endif
endif

Scheduling is earliest-deadline-first and non-preemptive. To ensure non-preemption, we add the term $CurProc(p) =$ undef to the guard of transition Schedule and $CurProc(p) = MS(p)$ and $CurProc(p) = BS(p)$ to the guards of the transitions of the MembershipServer and BroadcastServer modules respectively. When the membership server or broadcast server of p completes its current task, it sets $CurProc(p)$ to undef – which enables the scheduler to run. The scheduler then schedules the next process by setting $CurProc(p)$ to the appropriate value.

$\text{APTPART}(p)$ is the set of messages from which $CurMes(p)$ is chosen. If $\text{APTPART}(p) = \emptyset$ then there are currently no messages available for the membership server to see, hence the scheduler goes ahead. If $\text{APTPART}(p)$ is not empty then there are messages available; the scheduler therefore waits till one of these messages has been selected (in other words, till $CurMes(p)$ becomes defined).

In addition to the membership and broadcast servers there may be other processes running tasks unrelated to our protocol. These tasks need to be taken into account because they can affect our protocol by taking too much time to execute and causing the protocol-related tasks to miss their deadlines. We can model this situation using the nondeterminism of transition rules. To put it another way, a transition rule need not fire the instant it is enabled. The time between the instant the transition rule was enabled to the instant it fired can be considered to be time utilized by some other process.

3.3 MembershipServer(p)

Transition Initialize

if $(Status(p) = \mathsf{recovered}) \wedge (CurProc(p) = MS(p))$ **then**
  $BCastTime(p) := \mathsf{undef}$, $CurMes(p) := \mathsf{undef}$
  $GroupId(p) := \mathsf{undef}$, $Members(p) := \mathsf{undef}$
  $StartUpTime(p) := \mathsf{Clock}(p) + d_n$
  $InTransit((\mathsf{new\_gp}, \mathsf{Clock}(p) + d_n, \{p\})) := \mathsf{true}$
  $Status(p) := \mathsf{sober}$, $CurProc(p) := \mathsf{undef}$
endif

The above rule initializes the state information and broadcasts a new_gp message. The state information stored by the membership server of a processor p consists the group identifier of p's current group ($GroupId(p)$), a set containing the members of p's group ($Members(p)$) and the time at which the processor recovered from its last crash ($StartUpTime(p)$).

Sending a broadcast m is modeled by setting $InTransit(m)$ to true – this signifies that m is currently being propagated on the network.

Add $(Status(p) = \mathsf{sober}) \wedge (CurProc(p) = MS(p))$ to the guards of the following two transitions.

Transition HandleNewGpMes

if $(MesType(CurMes(p)) = \mathsf{new\_gp})$ **then**
  if $(\mathsf{Clock}(p) > Timestamp(CurMes(p)))$ **then** Crash_MS
    $Status(p) := \mathsf{crashed}$
  else PrMesSend_MS
    $InTransit((\mathsf{present}, Timestamp(CurMes(p)), \{p\})) := \mathsf{true}$
    $BCastTime(p) := Timestamp(CurMes(p)) + d_h$
    $CurMes(p) := \mathsf{undef}$, $CurProc(p) := \mathsf{undef}$
  endif
endif

If the membership server of gets scheduled after the deadline of the message has been exceeded, it removes itself from the group it is currently a member. This "removal" is modeled by setting $Status(p)$ to crashed

Transition HandlePresentMes

if $(MesType(CurMes(p)) = \mathsf{present})$ **then**
  if $(Members(p) \neq View(CurMes(p)))$ **then** ChangeGp
    $Members(p) := View(CurrMes(p))$
    $GroupId(p) := Timestamp(CurMes(p))$
  endif
  $CurMes(p) := \mathsf{undef}$, $CurProc(p) := \mathsf{undef}$
endif

3.4 BroadcastServer(p)

Transition HandleBCast

if $(Status(p) = \mathsf{sober}) \wedge (CurProc(p) = BS(p))$ **then**
  if $(\mathsf{Clock}(p) > BCastTime(p))$ **then** Crash_BS
    $Status(p) := \mathsf{crashed}$

else PrMesSend_BS

$InTransit((\mathsf{present}, BCastTime(p), \{p\})) := \mathsf{true}$

$BCastTime(p) := BCastTime(p) + d_h$, $CurProc(p) := \mathsf{undef}$

endif

endif

3.5 MessageCarrier(p)

To model the transmission of a message from one processor to another, we introduce a Message-Carrier agent for each processor p; this agent delivers messages intended for p to p. The "delivery" is done by incorporating the message into $InBox(p)$ which is an element of 2^{Message} – the set of finite sets of messages. We view $InBox(p)$ both as an element as well as a set.

An incoming message m of type new_gp is simply added to the $InBox(p)$. Messages of type present with identical timestamps are however, bunched together into one message. If m is an incoming message (to p) of type present and there is no present message in $InBox(p)$ with the same timestamp as m, then m is simply added to $InBox(p)$. If there exists a single present message m' with the same timestamp as m then m' is deleted from $InBox(p)$ and the message $(\mathsf{present}, Timestamp(m), View(m) \cup View(m'))$ is added in its place. It is easy to see that there can never be more than one present message with the same timestamp in $InBox(p)$. The message to be incorporated into $InBox(p)$ is given by the external function *InMes*(p). More will be said about this function in section 4.

Transition DeliverIncomingMes

if (*InMes*$(p) = (a, b, c)$) **then**

if $(a = \mathsf{new\_gp})$ **then** $InBox(p) := InBox(p) \cup \{$*InMes*$(p)\}$

elseif $(Present(InBox(p), b) = \mathsf{undef})$ **then**

$InBox(p) := InBox(p) \cup$ *InMes*$(p)\}$

elseif $(Present(InBox(p), b) = m)$ **then**

$InBox(p) := (InBox(p) - \{m\}) \cup \{(a, b, View(m) \cup c)\}$

endif

endif

The static function $Present$ is defined as follows. Given $I \in 2^{\mathsf{Message}}$ and a real t, if there exists a unique present message $m \in I$ with timestamp t then $Present(I, t) = m$ otherwise $Present(I, t) =$ undef.

Note that transition DeliverIncomingMes never removes any message from $InTransit$, thus allowing $InTransit$ to grow in an unbounded manner. If we wish to keep the size of $InTransit$ bounded, we can modify the program as follows. Currently, if we wish to broadcast a message m, we set $InTransit(m)$ to true, thus sending a single copy of m to the entire group. Instead of doing this we can send a separate copy of m to each processor q, by making $InTransit$ a binary function – the first argument being the message sent and the second being its target – and replacing all updates of the form $InTransit(m) :=$ true with $InTransit(m, q) :=$ true. Then, the message carrier of each processor can remove its copy of m from $InTransit$ when it incorporates m into its $InBox$.

3.6 Custodian(p)

The protocol requires that the all the membership servers see the messages in order of their deadlines. Since the message carrier need not deliver the messages in this order, we have an agent

Custodian for every processor p which delivers a message with the minimal timestamp to the membership server.

Transition SelectCurMes
if $(Status(p) = \mathsf{sober}) \wedge (CurMes(p) = \mathsf{undef}) \wedge$
$(\textsc{AptPart}(p) \neq \emptyset) \wedge MinDl(m, \textsc{AptPart}(p))$ **then**
$CurMes(p) := m$, $InBox(p) := InBox(p) - m$
endif

Given a message m and a set I, $MinDl(m, I)$ is true if $m \in I$ and there is no message $m' \in I$ whose deadline is less than that of m.

Note that the custodian of p removes those messages that have been seen by the membership server from $InBox(p)$. However, the messages that arrive while the processor is crashed may never be seen by the membership server and hence are never removed. Thus, $InBox(p)$ can grow in an unbounded manner. To keep the number of messages in $InBox(p)$ bounded, we can have the custodian periodically remove from $InBox(p)$ those messages that can never be selected by transition SelectCurMes. For example, when the processor is crashed, we could remove all those messages whose timestamp is less than $\mathit{Clock}(p)$ and when the processor is alive, we could remove all those messages whose timestamp is less than $StartUpTime(p)$. Removal of these "unselectable" messages, however, does not relate to the protocol we are specifying; therefore, we do not deal with it in our specification.

4 Semantics: Definitions and Discussion

In subsection 4.1, we describe our "official" semantics. Real-time versions of that semantics and various other issues are discussed in subsection 4.2.

4.1 Definitions

For each processor p, let $\mathcal{E}(p)$ be the restriction of our evolving algebra $\mathcal{E}$ that involves only the five agents related to p: the scheduler, membership server, broadcast server, message carrier and custodian of p. For simplicity, we drop the processor name from the arguments of functions if it is clear from the context which processor we are referring to.

Semantics of $\mathcal{E}(p)$:

Vocabulary: Let $\sigma(p)$ be the vocabulary of $\mathcal{E}(p)$ excluding function $InTransit$.

Internal and external functions: Functions *Clock* and *InMes* are external input functions of $\mathcal{E}(p)$. From the point of view of $\mathcal{E}(p)$, their values are provided by the environment. Function $InTransit$ is an output external function; that is why it does not belong to $\sigma(p)$. The other functions in the vocabulary $\sigma(p)$ of $\mathcal{E}(p)$ are internal. Let $\sigma^-(p)$ be the internal vocabulary of $\mathcal{E}(p)$.

States: For brevity, we speak about states and runs of p rather than $\mathcal{E}(p)$. A *state* of p is a static $\sigma(p)$-algebra. An *internal state* of p is a static $\sigma^-(p)$-algebra. If S is a state, let S^- be the corresponding internal state.

Runs: Let I range over initial segments of natural numbers. A *(sequential) run* of p is a sequence $\rho_p = \langle S_n : n \in I \rangle$ of states of p such that, for each positive n, S_n^- is obtained from S_{n-1} by executing one rule $r(n)$ of $\mathcal{E}(p)$ at S_n. Notice that ρ_p uniquely determines rules $r(n)$ and each rule uniquely determines the agent whose program contains the rule. Call run ρ_p *monotone* if it satisfies the following condition

R1. *Monotonicity of the clock:* The values of **Clock** at states S_0, S_1, S_2, etc. form a strictly increasing sequence. If there is final state then **Clock** $= \infty$ in the final state.

Restrict attention to monotone runs. States S_n are *stages* of p in $\rho_p = \langle S_n : n \in I \rangle$. If $n \neq \max(I)$ then rule $r(n+1)$ *fires* at stage S_n. Notice that all stages have the same superuniverse. An *extended* $\sigma(p)$*-term* (relative to a run) is an expression built from elements of the superuniverse by means of functions in $\sigma(p)$.

Abbreviations: Let $\rho_p = \langle S_n : n \in I \rangle$ be a run of p and $a = S_n$. If k is a (possibly negative) integer and $n + k \in I$ then $a + k = S_{n+k}$. If τ is an extended $\sigma(p)$-term, then τ_a is the value of τ at stage a. Suppose that $n \neq \max(I)$ so that some rule $r = r(n+1)$ fires at a. If r assigns value x to a $\sigma(p)$-term τ, we say that p *sets* τ to x at stage a or τ *gets* value x at stage $a + 1$. If $r(n+1)$ assigns value **true** to $InTransit(m)$, we say that p *sends* message m at stage a. Call a monotone run $\rho_p = \langle S_n : n \in I \rangle$ of p *regular* if it satisfies the following two conditions

R2. *Lower bound on recovery time:* If $Status_a =$ **crashed** and $Status_{a+k} =$ **recovered** and $k > 0$ then $\mathsf{Clock}_{a+k} - \mathsf{Clock}_a \geq d_r$.

R3. *Initial state:* If a is the initial state then $Status_a =$ **crashed** and $InBox_a = \emptyset$, and $CurMes_a$, $GroupId_a$, $Members_a$, $StartUpTime_a$, $BCastTime_a$ are all **undef**.

We define a *run* ρ of the whole evolving algebra $\mathcal{E}$ to be simply a collection of runs ρ_p of p where p ranges over the processors. ρ is *regular* if all constituent runs ρ_p are regular and ρ satisfies the following condition

R4. *Carry time bound:* If p sends message $m = (x, y, z)$ to q at some stage a then there is a unique stage b of q such that
   1. $InMes(q) = m$ and
   2. There exists $m' = (x, y, z')$ in $InBox(p)_{b+1}$ where $z' \supseteq z$ and
   3. $0 < \mathsf{Clock}(q)_b - \mathsf{Clock}(p)_a \leq d_c$.

4.2 Discussion

We could use real-time semantics with either zero-time or prolonged actions, like in [1], but the material lends itself to simpler semantics which is more general in a sense.

Sequential Runs: According to [4], a sequential run is a sequence of states together with the agent-witness function. Since the sequence of states uniquely determines the agent-witness function in our case, we have simplified the definition. Also, we took advantage of the fact that agents of $\mathcal{E}(p)$ can fire only one rule a time and further simplified the definition.

Partially Ordered Runs: We restrict attention to sequential runs only to simplify the exposition. There are no significant changes in the correctness proof if one uses partially ordered runs as defined in [4].

Stages: Usually stages of a run $\rho_p = \langle S_n : n \in I \rangle$ are pairs (n, S_n). The first component ensures that the distinctness of stages and is not needed in the case of our monotone runs.

Initial states: Condition R3 can be generalized.

InMes Function: We don't care about the value of $\mathit{InMes}(p)$ at stage a unless the message carrier of p acts at a; it may be undef. It is probably more honest to remove *InMes* from the vocabularies of those stages where the message carrier is passive, but this would complicate a little the definition of run.

InMes and $InTransit$ Functions: The only connection between $\mathit{InMes}(p)$ and $InTransit$ is the condition R4. It is assumed that the environment that supplies *InMes* satisfies R4. In the case of real-time semantics (say with zero-time actions), we do not need the environment to supply *InMes*. Just add the following rule to the program of the message carrier of p:

Transition: SelectIncomingMes
if $(InMes(p) = \mathsf{undef}) \wedge InTransit(m) \wedge (Target(m) = p)$ **then**
    $InMes(p) := m$
endif

This rule seems inconsistent because there may be more than one messages to p in transit. According to the definition of evolving algebras, inconsistency is resolved by means of nondeterminism. If p has several incoming message then MessageCarrier(p) nondeterministically chooses one of them.

4.3 More Abbreviations and Definitions

Definition 1 *The predicate $Correct(p)$ is true for processor p at stage s if the following conditions are met*

1. $\mathsf{Clock}(p)_s \geq StartUpTime(p)_s$.
2. *For all* $m \in InBox(p)_s$, *if* $TimeStamp(m) \geq StartUpTime(p)_s$ *then* $Deadline(m) \geq \mathsf{Clock}(p)_s$.
3. $BCastTime(p)_s \geq \mathsf{Clock}(p)_s$.

We also define correctness in intervals

Definition 2 1. *If $a \leq b$ are stages of p, then p is correct in $[a, b]$ if it is correct at any stage c with $a \leq c \leq b$.*
2. *Let I be a real interval and $t_1 = \inf(I)$, $t_2 = \sup(I)$. Then, p is correct in I if it is correct in the stage interval $[a, b]$ where $\mathsf{C}(p)_a \leq t_1 < \mathsf{C}(p)_{a+1}$ and $\mathsf{C}(p)_{b-1} < t_2 \leq \mathsf{C}(p)_b$*

For brevity, we shorten the names of some functions – *Clock* is now C, $Timestamp$ is $TStamp$, $StartUpTime$ is $UpTime$, $CurMes$ is Mes, $MesType$ is $Type$, $GroupId$ is $GpId$ and the abbreviation DEFINED is DEF. For readability, we give the following definitions

D1. p joins group g – There is a stage s of p such that $GpId(p)_s = g$
D2. a message (a, b, c) is added to $InBox(p)$ at stage s by time t –
    $\mathsf{C}(p)_s \leq t$ and there exists a message $(a, b, c') \in InBox(p)_{s+1}$ such that
    $(a = \mathsf{present}) \rightarrow (c' \supseteq c)$ and $(a = \mathsf{new\_gp}) \rightarrow (c' = c)$
D3. a message from q with timestamp t is in $InBox(p)$ at stage s – $\exists m = (a, t, c) \in InBox(p)_s$ such that $q \in c$

D4. p sees a message with timestamp t at stage s – $Status(p)_s$ = sober and $CurProc(p)_s = MS(p)$ and $TStamp(Mes(p))_s = t$

D5. p fails at stage s – At least one of the following conditions is true at stage s and none is true at stage $s-1$.

a. $BCastTime(p)_s < \mathsf{C}(p)_s$

b. $(Type(Mes(p))_s =$ new_gp$)$ and $(TStamp(Mes(p))_s < \mathsf{C}(p)_s)$

c. $\exists m \in InBox(p)_s$ such that $(Type(m) =$ new_gp$)$ and $(UpTime(p)_s \leq TStamp(m) < \mathsf{C}(p)_s)$

4.4 What we shall be proving

The following are the properties that we will be proving about every regular run of $\mathcal{E}$. These correspond to goals G1 through G6 described in section 2.4.

Theorem 4: *Stability of local views*

For every p and all stages $a < b$ of p, the following holds. If undef $\neq GpId_a \neq GpId_b$ then there is a stage $c < b$ such that either

1. $c \geq a$ and $Status_c =$ crashed or
2. $c \geq a$ and $Type(Mes)_c =$ present and $Members_c - View(Mes)_c \neq \emptyset$ or
3. p sees a new_gp message m at c from some processor $q \neq p$ and $TStamp(m) \in (GpId(p)_a, GpId(p)_b]$.

Theorem 2: *Agreement on history*

Suppose that

1. $GpId(p)_a = GpId(q)_c \neq$ undef and
2. b is the first stage $> a$ such that undef $\neq GpId(p)_b \neq GpId(p)_a$ and
3. d is the first stage $> c$ such that undef $\neq GpId(q)_d \neq GpId(q)_c$ and

then, if $GpId(p)_b$ and $GpId(q)_d$ are not undef, they are equal.

Theorem 1: *Agreement on group membership*

If $GpId(p)_a = GpId(q)_b \neq$ undef then $Members(p)_a = Members(q)_b$.

Theorem 3: *Reflexivity*

For every p and stage a of p if $GpId_a \neq$ undef then $p \in Members_s$.

Theorem 5: *Bounded join delays*

There exists a positive real d_j satisfying the following condition. If

1. $Status(p)$ is set to sober at stage a and
2. p is correct in the interval $I = (\mathsf{C}(p)_a, \mathsf{C}(p)_a + d_j]$

then there exists a group id $g > \mathsf{C}(p)_a$ such that for every q correct in I $GpId(q)$ is set to g at some stage b of q with $\mathsf{C}(q)_b \in I$.

Theorem 6: *Bounded failure detection delays*

There exists a positive real d_f satisfying the following condition. If

1. p fails at stage a and
2. $I = (\mathsf{C}(p)_a, \mathsf{C}(p)_a + d_f]$ and $GpId(p)_a = g \neq$ undef

then there exists $g' > g$ such that

1. p never joins group g' and
2. for every q that joins group g and is correct in I $GpId(q)_b = g'$ for some stage b of q with $\mathsf{C}(q)_b \in I$.

5 Proof of Protocol

Fix any regular run ρ. In some of the proofs, we will be considering sums of stages. We define the sum of two stages a and b as follows. If a is the ith stage of p and b is the jth stage of q then $a+b$ is the number $i+j$. The initial state of any processor p is stage number 0.

5.1 Propositions Dealing With Message Sends and Receives

Proposition 1 *If p sends a message m to q at stage a then m is added to $InBox(q)$ at some stage b such that $C(q)_b < Deadline(m) - d_u$.*

Proof: Recall that $TStamp(m) = C(p)_a + d_n$ if m is a new_gp message. Examining rules HandleNewGpMes and HandleBCast (which are the only rules which can send a present message) we see that $TStamp(m) \geq C(p)_a$ if m is a present message. Also recall that $Deadline(m) = TStamp(m)$ if m is a new_gp message and $= TStamp(m) + d_n$ if m is a present message. Therefore $Deadline(m) \geq C(p)_a + d_n$ in either case. By the carry time bound constraint, $C(q)_b \leq C(p)_a + d_c \leq Deadline(m) - d_n + d_c$. But $d_n > d_c + d_u$ (see section 2.3). Therefore $C(q)_b < Deadline(m) - d_u$. □

Proposition 2 *If $Mes(p)_a = (x, y, z)$ and $q \in z$ then there exists a stage b of q at which q sends the message $(x, y, \{q\})$.*

Proof: Straightforward. □

5.2 Properties Satisfied at any Stage of any Processor

Proposition 3 *If $Status(p)_a =$ sober and $\text{DEF}(BCastTime(p)_a)$ then $BCastTime(p)_a \geq UpTime(p)_a$.*

Proof: By induction on ρ.
Basis Case: At the initial stage a_0 of p, $Status(p)_{a_0} =$ crashed, hence the claim is vacuously true.
Induction Step: Assume that the statement is true at stage $a-1$. The only interesting case is if $Status(p)_a =$ sober and $\text{DEF}(BCastTime(p)_a)$. If the value of $BCastTime(p)$ does not change between $a-1$ and a, we know by induction hypothesis that the claim holds at a. The only transitions that can change $BCastTime(p)$ or $UpTime(p)$ are Initialize, PrMesSend_MS and PrMesSend_BS.

Initialize sets $BCastTime(p)$ to undef so it cannot violate the claim. PrMesSend_BS just increments $BCastTime(p)$ so it cannot violate the claim. PrMesSend_MS sets $BCastTime(p)$ to $TStamp(Mes(p)) + d_h$. From examination of the custodian we can see that $TStamp(Mes(p))$ has to be $\geq UpTime(p)$ – therefore, firing PrMesSend_MS cannot violate the claim. □

Proposition 4 *If p sends a* present *message with timestamp t in a stage a then the $UpTime(p)_a \leq t$.*

Proof: By induction on ρ.
Basis Case: At the initial stage a_0 of p, $Status(p)_{a_0} =$ crashed, hence the claim is vacuously true.
Induction Step: Assume that the statement is true at stage $a-1$. There are two ways a present message can be sent at stage a.
Case 1: PrMesSend_BS fires at a. The timestamp of the message sent is equal to $BCastTime(p)_a$ which, by Proposition 3 is $\geq UpTime(p)_a$.
Case 2: PrMesSend_MS fires at a. The timestamp of the message sent is equal to $TStamp(Mes(p))_a$. From examination of the custodian, we can see that this is $\geq UpTime(p)_a$. □

5.3 Relationships Between Different Stages of the Same Processor

Proposition 5 *If $a < b$ then $UpTime(p)_a \leq UpTime(p)_b$.*

Proof: Observe that $UpTime(p)$ is set only when Initialize is fired and moreover, the new value is $C(p) + d_n$. Since $C(p)$ monotonically increases, we can say the same about $UpTime(p)$. □

Proposition 6 *If $GpId(p)_c = g \neq$ undef then there exists a stage $a < c$ such that*
1. $GpId(p)$ was set to g at a by the firing of ChangeGp *and*
2. ChangeGp *was not fired at any stage $b \in (a, c)$*

Proof: Observe that ChangeGp is the only clause that sets $GpId(p)$ to a value $\neq$ undef. If $GpId(p)_c = g \neq$ undef, the value must have been set by the last firing of ChangeGp. □

Proposition 7 *If $q \in Members(p)_c$ then there is a stage $a < c$ and message m such that*
1. $q \in View(m)$ and p sees m at a but not at $a + 1$
2. For all stages $b \in (a, c)$,
either $Mes(p)_b = Mes(p)_c$ or $Mes(p)_b =$ undef *and*
neither Initialize *nor* HandlePresentMes *is fired at b.*

Proof: Since DEF($Members(p)_c$), we can see that this value is set by some firing of ChangeGp before c. Therefore HandlePresentMes fires at some state $s < c$ and Initialize is not fired in $[s, c)$. Let a be the latest stage $< c$ at which HandlePresentMes fires. Examining the rules, we can see that $Mes(p)_{a+1} =$ undef.

If p sees a present message $m' \neq Mes(p)_c$ at some $b \in (a + 1, c)$ then from examination of the rules we can see that HandlePresentMes fires at some stage $b' \in [b, s)$. This is impossible since we have assumed that a is the latest stage before c at which HandlePresentMes fires. We can see from examining HandlePresentMes that $Members(p)_{a+1} = View(m)$ and since ChangeGp does not fire in (a, c), $Members(p)_c = Members(p)_{a+1}$. Therefore if $q \in Members(p)_c$, $q \in View(m)$. □

Proposition 8 *Suppose $GpId(p)_a = g$ and c is the first stage $> a$ such that p sees a* present *message m at c and $View(m) \neq Members(p)_c$ and p stays* sober *in the interval $[a, c]$. Then for all stages $b \in [a, c]$, $GpId(p)_a = g$.*

Proof: Observe that while p stays sober, the only way that $GpId(p)$ can change is if p sees a present message m such that $View(m) \neq Members(p)_c$ at some stage $c > a$. □

Proposition 9 *If $Mes(p)_b = m$ and* DEF(m) *then*
1. $C(p)_{b-1} \geq Deadline(m) - d_u$ and
2. there exists an $a < b$ such that
$m \in InBox(p)_a$ and $C(p)_{a-1} < Deadline(m) - d_u$.

Proof: Let us without loss of generality assume that b is the first stage where $Mes(p) = m$. This implies that $Mes(p)$ is set to m at $b - 1$. From examination of the custodian, we can see that $C(p)_{b-1} \geq Deadline(m) - d_u$ – this proves the first part of the claim. Consider any $q \in View(m)$. By Proposition 2, this implies that q sent a message m' to p with timestamp equal to $TStamp(m)$. By Proposition 1, m' is added to $InBox(p)$ by time $Deadline(m) - d_u$. This proves the second part of the claim. □

Proposition 10 *If p sees a* present *message m at stage a but not at $a + 1$ and a* present *message m' at stage $b > a$ then $TStamp(m) \neq TStamp(m')$.*

Proof: By contradiction. Assume that $TStamp(m) = TStamp(m')$. By Proposition 9, there exists a stage a' such that $C(p)_{a'-1} < Deadline(m) - d_u$ and $m \in InBox(p)_{a'}$. There exists a similar stage b' for m'. Proposition 9 also tells us that if at any stage c, $Mes(p)_c = m$ or $Mes(p)_c = m'$ then $C(p)_{c-1} \geq Deadline(m) - d_u$. This implies that there exists a stage d such that $C(p)_{d-1} < Deadline(m) - d_u$ and $m, m' \in InBox(p)_d$. From examination of the message carrier, we can see that this is impossible. Therefore $TStamp(m) \neq TStamp(m')$. □

Proposition 11 *If* $m = Mes(p)_s \neq$ undef *and* $m' = Mes(p)_{s'} \neq$ undef *and* $Deadline(m) < Deadline(m')$ *then* $s < s'$.

Proof: By contradiction. By Proposition 9, there exists an $a < s$ such that $C(p)_{a-1} < Deadline(m) - d_u$ and $m \in InBox(p)_a$ and that there exists an $a' < s'$ such that $C(p)_{a'-1} < Deadline(m') - d_u$ and $m' \in InBox(p)_{a'}$. Assume that $s' < s$. We know from examination of the custodian that $Mes(p)$ cannot be set to m' until time $Deadline(m') - d_u$.

Let $b' < s'$ be a stage such that $Mes(p)$ is set to m' at b' and $Mes(p)$ stays unchanged in $[b'+1, s']$. Therefore, $C(p)_{b'} \geq Deadline(m') - d_u > Deadline(m) - d_u$. Since $C(p)_{a-1} < Deadline(m) - d_u$, $a \leq b' < s'$.

Let b be the stage at which $Mes(p)$ is set to m. We know that setting $Mes(p)$ to m will also cause m to be removed from $InBox(p)$. Since p sees m at stage s and since $s' < s$, we conclude that $b' < b$. Since $a \leq b'$, we can conclude that $m \in InBox(p)_{b'}$ – this implies that there is a message with an earlier deadline than that of m' in $InBox(p)$ when $Mes(p)$ is set to m'.

The only way this can happen is if m is ineligible for selection – in this case, that implies $UpTime(p)_{b'} > TStamp(m)$. But we know that $Mes(p)$ is set to m at stage $b > b'$. By Proposition 5, $UpTime(p)_{b'} \leq UpTime(p)_b$ which implies m is *not* ineligible at b' – a contradiction. □

Proposition 12 *If* $a < b$, DEF$(GpId(p)_a)$ *and* DEF$(GpId(p)_b)$ *then* $GpId(p)_a \leq GpId(p)_b$.

Proof: By contradiction. The only interesting case is when $GpId(p)_a \neq GpId(p)_b$. By Proposition 6, if $GpId(p)_a = g$ and $GpId(p)_b = g'$ then there exists $a' < a$ such that $Type(Mes(p))_{a'} =$ present, $TStamp(Mes(p))_{a'} = g$, ChangeGp is fired in stage a', ChangeGp is not fired at any stage in (a', a) and there exists a stage $b' < b$ such that $Type(Mes(p))_{b'} =$ present, $TStamp(Mes(p))_{b'} = g'$,ChangeGp is fired in stage b' and ChangeGp is not fired at any stage in (b', b). Since we are considering the case where $GpId(p)_a \neq GpId(p)_b$, $a' \neq b'$.

Assume the claim is false – in other words, that $GpId(p)_a > GpId(p)_b$. This implies that $TStamp(Mes(p))_{a'} > TStamp(Mes(p))_{b'}$. By Proposition 11, that implies $a' > b'$. But we know that $a < b$ and that ChangeGp does not fire in (a', a). This implies $a' \leq b'$ – a contradiction. □

Proposition 13 *If* $a < c$, $m_1 = Mes(p)_a \neq$ undef, $m_2 = Mes(p)_c \neq$ undef, p *stays* sober *in the interval* $[a, c]$ *and there exists a message* $m_3 = (x, y, z)$ *and stage* s *such that* $m_3 \in InBox(p)_s$ *and* $Deadline(m_1) < Deadline(m_3) < Deadline(m_2)$ *then there exists a stage* $b \in (a, c)$ *at which* p *sees a message* $m_4 = (x, y, z')$ *where* $z' \supseteq z$.

Proof: Argument similar to Proposition 11. □

Proposition 14 *If* $m_1 = Mes(p)_b \neq$ undef, $UpTime(p)_b = t$, *and there exists a stage* s *and message* $m_2 = (x, y, z)$ *such that* $m_2 \in InBox(p)_s$ *and* $t \leq Deadline(m_2) < Deadline(m_1)$ *then there exists a stage* $a < b$ *at which* p *sees a message* $m_3 = (x, y, z')$ *where* $z' \supseteq z$.

Proof: Argument similar to Proposition 11. □

Proposition 15 *If p gets* sober *at stage a then there exists an $b > a$ such that*
1. p stays sober *in $[a, b]$ and*
2. p sees a new_gp *message m' at b such that $TStamp(m) = UpTime(p)_b$ and*
3. p does not see any message in $[a, b)$.

Proof: Since p gets sober at stage a, transition Initialize must have been fired at stage $a - 1$ sending a new_gp message (call it m) to p. By Proposition 1, m is added to $InBox(p)$ at some stage $c > s$. Examining the rules, we can see that the two ways p can crash are by the firing of transition Crash_MS or Crash_BS. In the first case, p sees a new_gp message before crashing. In the second case, $BCastTime(p)$ has to be defined. $BCastTime(p)$ becomes defined only when a new_gp message is seen by p. In both cases, p will see *some* message before crashing.

Let the first stage $> a$ in which p sees a message be b and let the message seen be m'. Since p sends new_gp message m in a, $UpTime(p)_{a+1} = TStamp(m)$. Since p does not crash in $[a + 1, b]$, $UpTime(p)$ remains $TStamp(m)$ in $[a + 1, b]$. Examining the custodian therefore, we can see that $TStamp(m') \geq TStamp(m)$. Since m is a new_gp message, $Deadline(m) = TStamp(m)$. Therefore, $Deadline(m') \geq Deadline(m)$. If $Deadline(m') > Deadline(m)$, then by Proposition 14, p sees m at some stage $d \in (a, b)$ – which contradicts our earlier assumption that m' is the first message seen by p since a. Therefore $Deadline(m') = Deadline(m)$.

We know $TStamp(m') \geq TStamp(m)$. $TStamp(m')$ cannot be greater than $TStamp(m)$ since that would mean that $Deadline(m') > Deadline(m)$. Therefore, $TStamp(m') = TStamp(m)$. Since the timestamps and deadlines of m' and m are the same and since m is a new_gp message, so is m'. This proves the claim. □

Proposition 16 *If p gets* sober *at stage a and there exists an b such that*
1. p stays sober *in $[a, b]$ and*
2. p sees a present *message at b and*
3. p does not see any present *messages in $[a, b)$*
then $TStamp(Mes(p))_b = UpTime(p)_b$

Proof: Argument similar to Proposition 15 □

Proposition 17 *If p sends a* present *message with timestamp t at stage a and there exists a stage $b > a$ such that $Status(p)_b =$* sober *and $Type(Mes(p))_b =$* present *and $TStamp(Mes(p))_b = t$ then p stays* sober *in (a, b).*

Proof: By contradiction. Let m be $Mes(p)_b$. From examining the custodian, we know that $UpTime(p)_b \leq TStamp(m) = t$. From examination of rules PrMesSend_MS and PrMesSend_BS we can see that $C(p)_a \geq t - d_u$. If there is some stage $c \in (a, b)$ such that $Status(p)_c =$ crashed), we know, from our premise, that there must be some other stage $d \in (c, b]$ such that $Status(p)_d =$ sober.

Without loss of generality, assume that d is the first such stage since c. By our failure and recovery constraints, $UpTime(p)_d > t$. Since $d \leq b$, by Proposition 5, $UpTime(p)_d \leq UpTime(p)_b$ and therefore $UpTime(p)_b > t$. This contradicts our earlier conclusion that $UpTime(p)_b \leq t$. □

Proposition 18 *Let $x = CurProc(p)_s$ and $x' = CurProc(p)_{s'}$. If* DEF(x), DEF(x'), $x \neq x'$ *and* DLINE$(x)_s <$ DLINE$(x')_{s'}$ *then $s < s'$.*

Proof: By contradiction. There are two cases.
Case 1: $x = MS(p)$ and $x' = BS(p)$. Let m denote $Mes(p)_s$. Let t denote $Deadline(m)$ and t' denote $BCastTime(p)_{s'}$. From premise, we know $t < t'$. Assume that $s' < s$. Let $a' < s'$ be

the stage such that $CurProc(p)$ is set to $BS(p)$ at a' and $BCastTime(p)_{a'} = t'$. Examining the scheduler, we know that $C(p)_{a'} \geq t' - d_u > t - d_u$. From Proposition 9, there exists a stage $a < s$ such that $m \in InBox(p)_a$ and $C(p)_a < t - d_u$. Since $C(p)_{a'} > t - d_u$, $a < a'$.

Since p sees m at s, there is some stage $c < s$ such that $Mes(p)$ is set to m at c and $Mes(p)$ stays unchanged in $[c+1, s]$. We consider two cases.

First, that $c < a'$. In that case, $Mes(p)_{a'} = m$. This means that at a', the broadcast server is scheduled when there existsa message with an earlier deadline – a contradiction.

The other case is $c \geq a'$. If $\textsc{Def}(Mes(p)_{a'})$ then by Proposition 11, $Deadline(Mes(p))_{a'} \leq t$. This once again means that at a', the broadcast server is scheduled when there exists a message with an earlier deadline – a contradiction. Therefore, $Mes(p)_{a'} = \mathsf{undef}$.

We know from Proposition 10 that there cannot be a state $d < a'$ such that $Type(Mes(p))_d = Type(m)$ and $TStamp(Mes(p))_d = TStamp(m)$. We also know that there is a stage $a < a'$ such that $m \in InBox(p)_a$. Since the only transition that removes a message from $InBox(p)$ is SelectCurMes, $m \in InBox(p)_{a'}$. Since $C(p) > t - d_u$, $m \in \textsc{AptPart}(p)_{a'}$. This however means that transition Schedule cannot fire at a' – a contradiction.

Case 2: $x = BS(p)$ and $x' = MS(p)$. Let m' denote $Mes(p)_{s'}$. Let $t = BCastTime(p)_s$ and $t' = Deadline(m')$. Let a be the latest stage $< s$ such that $BCastTime(p)_a \neq t$. Therefore, $BCastTime(p)_{a+1} = t$ and it stays unchanged in $(a+1, s]$. $BCastTime(p)$ can be set by either PrMesSend_MS or by PsMesSend_BS. In either case, we can see by examining the scheduler that $C(p)_a \geq t - d_h - d_u$.

Examining rules HandleNewGpMes and HandlePresentMes, we can see that $C(p)_a \leq t - d_h$. Assume $s' < s$. Let a' be the latest stage $< s'$ such that $CurProc(p)_{a'} \neq MS(p)$. By examining scheduler, we can see that $Mes(p)_{a'} = m'$. From examination of the custodian, we can see that $Mes(p)$ can be set to m' only when $C(p) \geq t' - d_u$. Therefore, $C(p)_{a'} \geq t' - d_u$. We know that $C(p)_a \leq t - d_h$. Since $t < t'$ and $d_u < d_h$, $a < a'$. Since $a' < s' < s$, $BCastTime(p)'_a = t$. But this means that the broadcast task had an earlier deadline than the membership server task that was scheduled at a' – a contradiction. □

Proposition 19 *If p sees a* new_gp *message at stage a then $BCastTime(p)_a \leq TStamp(Mes(p))_a + d_h$.*

Proof: By induction on ρ.
Basis Case: At the initial stage a_0 of p, $Status(p)_{a_0} = \mathsf{crashed}$, therefore p cannot see any message at a_0, hence the claim is vacuously true.
Induction Step: Assume that the statement is true for all stages $< a$. Let m be $Mes(p)_a$. Let $Type(m) = \mathsf{new\_gp}$ and let t denote $TStamp(m)$. Assume that $BCastTime(p)_a > t + d_h$. This value is not set by the previous firing of PrMesSend_MS since, by Proposition 11, the timestamp of the last new_gp message seen by p is $\leq t$. So, the value is set by the last execution of the broadcast server. This implies that there exists a stage $b < a$ such that $(CurProc(p)_b = BS(p)) \wedge (BCastTime(p)_b > t)$. But this is impossible since it violates Proposition 18. □

Proposition 20 *If* $\textsc{Def}(BCastTime(p)_a)$ *then* $BCastTime(p)_a \leq C(p)_{a-1} + d_h + d_u$.

Proof: By induction on ρ.
Basis Case: At the initial stage a_0 of p, $BCastTime(p)_{a_0} = \mathsf{undef}$, hence the claim is vacuously true.
Induction Step: Assume that the statement is true for all stages $< a$. The difference between $BCastTime(p)_a$ and $C(p)_{a-1}$ will be greatest if $BCastTime(p)$ is changed at $a - 1$. Thereafter, the difference shrinks until we get to the next stage at which $BCastTime(p)$ is changed. There are

two possible ways at which $BCastTime(p)$ can be changed.
Case 1: PrMesSend_MS fires at $a-1$. From examination of scheduler rules we can see that $C(p)_{a-1} \geq TStamp(Mes(p))_{a-1} - d_u$ and from examination of HandleNewGpMes, we can see that $C(p)_{a-1} \leq TStamp(Mes(p))_{a-1}$. Since $BCastTime(p)_a = TStamp(Mes(p))_{a-1} + d_h$, the claim is true.
Case 2: PrMesSend_BS fires at $a-1$. Examining the scheduler rules, we can see that $C(p)_{a-1} \geq BCastTime(p)_{a-1} - d_u$ and examining HandleBCast, we can see that $C(p)_{a-1} \leq BCastTime(p)_{a-1}$. Since $BCastTime(p)_a = BCastTime(p)_{a-1} + d_h$, the claim is true. □

Proposition 21 *If* $a < b$, $\textsc{Def}(BCastTime(p)_a)$ *and* $\textsc{Def}(BCastTime(p)_b)$ *then* $BCastTime(p)_a \leq BCastTime(p)_b$.

Proof: Let $t = BCastTime(p)_a$ and $t' = BCastTime(p)_b$. Without loss of generality, we can restrict our attention to the following two cases.
Case 1: A transition fired at a causes the processor to crash and b is the first stage after the crash where $\textsc{Def}(BCastTime(p))$. By Proposition 20, $t \leq C(p)_{a-1} + d_h + d_u$. From our failure and recovery constraints, we can see that $UpTime(p)_b > t$. Let $t'' = UpTime(p)_b$. We can also see from examination of the algebra that $BCastTime(p)$ is set by the first new_gp message seen by p after the crash. By Proposition 15, the timestamp of that message is equal to t''. Examining transition HandlePresentMes, we can see that $t' = t'' + d_h > t + d_h$. Therefore the claim cannot be violated in this case.
Case 2: p stays alive between a and b and b is the first stage $> a$ at which the value of $BCastTime(p)$ is different from $BCastTime(p)_a$. The value of $BCastTime(p)$ can be changed by two transitions – PrMesSend_MS and PrMesSend_BS. If the value is changed by transition PrMesSend_BS, the new value will obviously be greater than the previous one. Consider the case when the value is changed by PrMesSend_MS. We know that $BCastTime(p)_{b-1} = t$. By Proposition 19, $TStamp(Mes(p))_{b-1} + d_h \geq t$. Therefore $t' > t$. □

Proposition 22 *If* $a < c$, $v_1 = CurProc(p)_a$, $v_2 = CurProc(p)_c$, p *stays* sober *in* $[a, c]$ *and there exists stage* s *of* p *and message* $m = (x, y, z)$ *such that*
1. $m \in InBox(p)_s$ *and*
2. $\textsc{Dline}(v_1) < Deadline(m) < \textsc{Dline}(v_2)$
then there exists a stage $b \in (a, c)$ *at which* p *sees a message* $m' = (x, y, z')$ *where* $z' \supseteq z$.

Proof: Argument similar to that for Proposition 18. □

Proposition 23 *If* $a < c$, $v_1 = CurProc(p)_a$, $v_2 = CurProc(p)_c$, p *stays* sober *in* $[a, c]$ *and there exists stage* s *of* p *such that* $\textsc{Dline}(v_1) < BCastTime(p)_s < \textsc{Dline}(v_2)$ *then there exists a stage* $b \in (a, c)$ *such that* $BCastTime(p)_s = BCastTime(p)_b$ *and* $CurProc(p)_b = BS(p)$.

Proof: Argument similar to that for Proposition 18. □

Proposition 24 *Let* $m = Mes(p)_a$ *and* $m' = Mes(p)_{a'}$. *Then, if* $a' < a$, *and*
1. $Type(m') = Type(m) =$ present *and*
2. $p \in View(m')$ *and*
2. p *stays* sober *and does not see any* present *messages other than* m *and* m' *in* $[a', a]$
then $0 \leq TStamp(m) - TStamp(m') \leq d_h$

Proof: By contradiction. Let $t = TStamp(m)$ and $t' = TStamp(m')$. Assume that $t - t' > d_h$. If $m = m'$ then the claim is trivially true, so assume $m \neq m'$. By Propositions 10 and 11,

$t - t' > 0$. Since $p \in View(m')$, by Proposition 2, p sends a present message with timestamp t' at some stage $b' < a'$. Examining the membership and broadcast servers, we can see that $BCastTime(p)_{b'+1} = t' + d_h$.

By Proposition 17, p stays sober in (b', a'). From the premise, p stays sober in $[a', a]$. Therefore, p stays sober in $(b', a]$. By Propositions 22 and 23, either this broadcast is sent or is preempted by the arrival of a new_gp message. In either case, p sends a present message with timestamp $t_1 \in (t', t' + d_h]$.

By Propositions 1 and 13, p sees a present message with timestamp t_1 at some stage $a_1 \in (a', a)$. This is a contradiction since, by our premise, p does not see any present messages other than m and m' in (a', a). □

5.4 Relationships Between Stages of Two Processors

Proposition 25 *If $Mes(p)_s = (x, y, z)$ then for any processor q*
1. There exists a stage a such that $(x, y, z) \in InBox(q)_a$ and
2. There is no stage b such that $(x, y, z') \in InBox(q)_b$ where $z' - z \neq \emptyset$.

Proof: By examining the custodian, we can see that there exists an $s' < s$ such that $(x, y, z) \in InBox(p)_{s'}$. By Propositions 2 and 1, for every $r \in z$, a message $(x, y, \{r\})$ is added to $InBox(q)$ by time $Deadline((x, y, z)) - d_u$.

We can see from examination of the message carrier that any incoming new_gp messages are simply added to $InBox$ and not "bunched" together as present messages are. Therefore, for any new_gp message m, $View(m)$ always contains exactly one processor id. Therefore, if $x =$ new_gp, z contains exactly one processor id – which means that (x, y, z) is added to $InBox(q)$ at some stage a, then $(x, y, z) \in InBox(q)_{a+1}$. This proves the claim.

If $x =$ present, we can see from examining the message carrier that all present messages with the same timestamp are "compressed" into one message. By examining the custodian, we can see that this message cannot be removed from $InBox(q)$ until time $y + d_n - d_u$. We know that for every $r \in z$, $(x, y, \{r\})$ is added to $InBox(q)$ by time $y + d_n - d_u$. Therefore, there exists a stage a such that $(x, y, z') \in InBox(q)_a$ where $z' \supseteq z$. By symmetry, any message $(x, y, \{r\})$ that is added to $InBox(p)$ is also added to $InBox(q)$, hence $z \supseteq z'$. This means that $z' = z$. □

Proposition 26 *If $Type(Mes(p))_a = Type(Mes(q))_b =$ present and $TStamp(Mes(p))_a = TStamp(Mes(q))_b$ then $Mes(p)_a = Mes(q)_b$.*

Proof: By contradiction. Let $m_1 = Mes(p)_a$ and $m_2 = Mes(q)_b$. We have to show that $View(m_1) = View(m_2)$. Assume the converse. By examining the custodian, we know that there exists an $a' < a$ such that $m_1 \in InBox(p)_{a'}$ and there exists a $b' < b$ such that $m_2 \in InBox(p)_{b'}$. But, by Proposition 25, such a situation is impossible. □

5.5 The First Group of Theorems

Lemma 1 *If $x = CurProc(p)_a$, $y = CurProc(p)_b$, $\textsc{Def}(x)$, $\textsc{Def}(y)$ and $\textsc{Dline}(x)_a < \textsc{Dline}(y)_b$ then $a < b$.*

Proof: Recall that the deadline for a membership server task is $Deadline(Mes(p))$ and the deadline for a broadcast is $BCastTime(p)$. There are three cases.
Case 1: $x = y = MS(p)$. The claim follows from Proposition 11.
Case 2: $x = y = BS(p)$. The claim follows from Proposition 21.
Case 3: $x \neq y$. The claim follows from Proposition 18. □

Theorem 1 *If* $GpId(p)_a = GpId(q)_b \neq$ undef *then* $Members(p)_a = Members(q)_b$.

Proof: By induction on $a + b$.
Basis Case: $a + b = 0$. In that case $GpId(p)_a = GpId(q)_b =$ undef hence the claim is vacuously true.
Induction Step: Assume that the statement is true for $a + b < k$. There are three cases.
Case 1: $GpId(p)_{a-1} = GpId(p)_a = g$. All we have to show is that $Members(p)_{a-1} = Members(p)_a$ and the claim follows from the induction hypothesis. Assume that $Members(p)_{a-1} \neq Members(p)_a$. The only way that this can happen is if p sees a present message with timestamp g at stage $a - 1$ and transition ChangeGp is fired. Since $GpId(p)_{a-1} = g$, we know from Proposition 6 that there exists a stage $c < a - 1$ at which p sees a present message and ChangeGp is fired. From examination of rule HandlePresentMes, we can see that $Mes(p)_{c+1} =$ undef. This implies that p sees two present messages with the same timestamp. This violates Proposition 10 – a contradiction.
Case 2: $GpId(q)_{b-1} = GpId(q)_b = g$. Argument similar to Case 1.
Case 3: $GpId(p)_{a-1} \neq GpId(p)_a$ and $GpId(q)_{b-1} \neq GpId(q)_b$. This implies that transition ChangeGp is fired at both $a - 1$ and $b - 1$ which implies that $Members(p)_a = View(Mes(p))_{a-1}$ and $Members(q)_b = View(Mes(q))_{b-1}$. Since $GpId(p)_a = GpId(q)_b$, we know that $TStamp(Mes(p))_{a-1} = TStamp(Mes(q))_{b-1}$. This implies, by Proposition 26 that $View(Mes(p))_{a-1} = View(Mes(q))_{b-1}$ which implies $Members(p)_a = Members(q)_b$. □

Theorem 2 *Suppose that*
1. $GpId(p)_a = GpId(q)_b \neq$ undef *and*
2. a' *is the first stage* $> a$ *such that* $GpId(p)_{a'} \neq GpId(p)_a$ *and*
3. b' *is the first stage* $> b$ *such that* $GpId(q)_{b'} \neq GpId(q)_b$
then, if $GpId(p)_{a'}$ *and* $GpId(q)_{b'}$ *are not* undef, *they are equal.*

Proof: By contradiction. Without loss of generality assume that $a' = a + 1$ and $b' = b + 1$. Let $g = GpId(p)_a = GpId(q)_b$ and let $g_a = GpId(p)_{a'}$ and $g_b = GpId(q)_{b'}$. By Proposition 12, $g < g_a$ and $g < g_b$. Assume the claim is false. Assume without loss of generality that $g_b < g_a$.

By Proposition 6, there exists a stage $c < a$ at which p sees a present message with timestamp g and at which ChangeGp fires and for all $c' \in (c, a]$, $GpId(p)_{c'} = g$. Since $Status(p)_a =$ sober and since $GpId(p)$ does not change in $(c, a]$ we can conclude that p stays sober in $(c, a]$.

Since $GpId(p)$ changes at a and $GpId(q)$ changes at b we know that ChangeGp fires in both stages. Let $m = (x, y, z) = Mes(q)_b$. By Proposition 25, there exists stage d of p such that $m \in InBox(p)_d$ and that there is no stage d' and message $m' = (x, y, z')$ such that $m' \in InBox(p)_{d'}$ and $z' - z \neq \emptyset$.

Comparing group Ids, we can see that $Deadline(Mes(p)_c) < Deadline(m) < Deadline(Mes(p)_a)$. By Proposition 13 therefore, there exists a stage $e \in (c, a)$ and message $m' = (x, y, z')$ such that p sees m' at e and $z' \supseteq z$. We already know that $z' - z = \emptyset$, therefore $z' = z$ which implies $m' = m$. Since $e \in (c, a)$, $GpId(p)_e = g$. Since $GpId(p)$ doesn't change at stage $e + 1$, we know that $Members(p)_e = View(m)$. Since $GpId(q)$ *does* change at b, we know that $Members(q)_b \neq View(m)$. But $GpId(p)_e = GpId(q)_b = g$. Therefore $Members(p)_e = Members(q)_b$ – a contradiction. □

Lemma 2 *Let* $m = Mes(p)_a$, $t = TStamp(m)$, $m' = Mes(p)_{a'}$ *and* $t' = TStamp(m')$. *Suppose that* $p \neq q$, $a' < a$ *and*
1. q *sends a* present *message with timestamp* t *at stage* b *by firing* PrMesSend_BS. *and*
2. $Type(m') = Type(m) =$ present *and*
3. for all $s < a$, *if* $Type(Mes(p))_s =$ present *then* $p \in View(Mes(p))_s$ *and*

4. p stays sober *and does not see any* present *messages other than m' and m in $[a', a]$ then, $t - t' = d_h$.*

Proof: By contradiction. Assume that $t' \neq t - d_h$. By our premise, $p \in View(m')$. This implies by Proposition 2 that p sends a present message at some stage $a_1 < a'$ and the message sent has timestamp equal to t'. This implies that either PrMesSend_MS or PrMesSend_BS is fired at a_1. In either case, $BCastTime(p)_{a_1+1} = t' + d_h$. There are two possible cases.
Case 1: $t' < t - d_h$. We know from Proposition 24 that this cannot occur.
Case 2: $t' > t - d_h$. Since PrMesSend_BS is fired, we know a broadcast is sent. We can see from examination of the membership and broadcast servers that $BCastTime(q)$ is set to t at some stage $b' < b$ where a present message m_1 with timestamp equal to $t - d_h$ is sent.

We first show that $q \notin View(m')$ by contradiction. Assume that $q \in View(m')$. This implies by Proposition 2 that q sends a present message with timestamp t' at some stage b_1. From Lemma 1, $b_1 < b$. Examining the broadcast and membership servers, we can see that $BCastTime(q)_{b_1+1} > t$. By Proposition 21, the value $BCastTime(a)_b > t$. This however means that PrMesSend_MS is fired at b and not PrMesSend_BS – which contradicts our premise.

We have established that $q \notin View(m')$ and that q sends a present message m_1 with timestamp equal to $t - d_h$. There are two possible subcases
Subcase 2.1: There is a stage a_2 of p at which p sees a present message with timestamp equal to $TStamp(m_1)$.

By Proposition 11, $a_2 < a'$. By our premise, $p \in View(Mes(p))_{a_2}$. Therefore, by Proposition 2, there exists an $a_3 < a_2$ at which a present message with timestamp equal to $TStamp(m_1)$ is sent. By Lemma 1, $a_3 < a_1$. This implies that $BCastTime(p)_{a_3+1} = TStamp(m_1) + d_h$. Since $t - t' < d_h$ and $TStamp(m_1) = t - d_h$, p sends a message with timestamp less than $TStamp(m_1) + d_h$. The only way this can happen is if PrMesSend_MS is fired at stage a_1. This implies that $Type(Mes(p))_{a_1} =$ new_gp and $TStamp(Mes(p))_{a_1} = t'$.
Subcase 2.2: There is no stage of p at which p sees a present message with timestamp equal to $TStamp(m_1)$. By Proposition 1, there exists a stage a_2 of p and present message m_2 such that $m_2 \in InBox(p)_{a_2}$ and $TStamp(m_2) = TStamp(m_1)$. Since p *does* see m' we know from examination of the custodian that $UpTime(p)_{a'} \leq t'$. We also know that $UpTime(p)_{a'} > TStamp(m_1)$ since otherwise, Proposition 14 will be violated. Therefore, $TStamp(m_1) = t - d_h < UpTime(p)_{a'} \leq t'$. Since p is sober at a', there exists a stage a_3 such that p gets sober at a_3, $UpTime(p)_{a_3} = UpTime(p)_{a'}$ and a new_gp message with timestamp equal to $UpTime(p)_{a'}$ is sent and p stays sober in $(a_3, a']$.

By Proposition 14, this new_gp message is seen by p at some stage $a_4 \in (a_3, a')$. Since p stays sober in $(a_3, a]$, we know that p sends a present message with timestamp equal to $UpTime(p)_{a'}$ at some stage $a_5 \geq a_4$. Since the timestamp of the new_gp message seen at a_4 is in $(t - d_h, t']$ and since $t - t' < d_h$, we can see from examination of HandleNewGpMes that $BCastTime(p)_{a_5+1} > t'$. Therefore, PrMesSend_MS is fired at a_1. This implies that $Type(Mes(p))_{a_1} =$ new_gp and $TStamp(Mes(p))_{a_1} = t'$.

In both subcases, we can see from Proposition 2 that some processor r sends a new_gp message with timestamp t'. Call this message m_2. By Proposition 1, m_2 is added to $InBox(q)$. From our failure and recovery constraints, we can see that q stays sober in (b', b). Therefore, by Proposition 22, q sees m_2 at some stage $b_1 \in (b', b)$. Since q stays sober in (b', b), we know that q sends a present message with timestamp t' in reply to this new_gp message. This however, implies that $q \in View(m')$ – a contradiction. □

Lemma 3 *If $Type(Mes(p))_a =$ present then $p \in View(Mes(p))_a$.*

Proof: We prove this by induction.
Basis Case: At the initial stage a_0 of p, $Mes(p)_{a_0}$ = undef, hence the claim is vacuously true.
Induction Step: Assume that the statement is true for all stages $< a$. Let $m = Mes(p)_a$. We need concern ourselves only with the case where $Type(m)$ = present and $p \notin View(m)$. Consider a processor $q \in View(m)$. By Proposition 2, there exists a stage b at which a present message with timestamp equal to $TStamp(m)$ is sent. This implies that either PrMesSend_MS or PrMesSend_BS is fired at b.
Case 1: PrMesSend_MS is fired at b. By Proposition 2, there exists a processor r and stage c of r such that a new_gp message m_r with timestamp equal to $TStamp(m)$ is sent. We know from examination of the custodian that $UpTime(p)_a \leq TStamp(m)$. Therefore, by Propositions 1 and 14, p sees message m_r at some stage a'.

Examining the custodian, we can see that $C(p)_{a'-1} \geq TStamp(m) - d_u$. Therefore, if p crashes in the interval (a', a), we know from our failure and recovery constraints that $UpTime(p)_a > TStamp(m)$ – therefore, p stays sober in (a', a). This implies that at some stage $a_1 > a'$ PrMesSend_MS is fired. From this, Proposition 1 and examination of the custodian, we can conclude that $p \in View(m)$ – a contradiction.
Case 2: PrMesSend_BS is fired at b. Since HandleBCast sends a broadcast, $BCastTime(q)$ has been set at some stage $b' < b$ where a present message with timestamp equal to $TStamp(m) - d_h$ was sent. There are two subcases to consider.
Subcase 2.1: p sees a present message at stage $a' < a$ and stays sober in (a', a). Let $m' = Mes(p)_{a'}$. Without loss of generality, assume that p does not see any present messages other than m in (a', a).

By Lemma 2, $TStamp(m) - TStamp(m') = d_h$. By induction hypothesis, $p \in View(m')$. Therefore, by Proposition 2, p sends a present message with timestamp equal to $TStamp(m')$ at stage $a_1 < a'$. We can see that $BCastTime(p)_{a_1+1} = TStamp(m)$. There are only two ways this scheduled broadcast can be preempted.

The first is if there exists $a_2 > a_1$ at which PrMesSend_MS fires and the timestamp of the present message sent is between $TStamp(m) - d_h$ and $TStamp(m)$. In that case, by Propositions 1 and 13, there exists an $a_3 \in (a', a)$ at which p sees a present message and $TStamp(m) - d_h < TStamp(Mes(p))_{a_3} < TStamp(m))$. This is impossible since m' is the last present message seen by p before m.

The other way in which the broadcast can be preempted is if there exists a stage in (a_1, a) at which p gets crashed. In that case, by our our failure and recovery constraints, $UpTime(p)_a > TStamp(m)$. Let $a_4 \leq a$ be the first stage such that $Mes(p)_{a_4} = m$. Examining the custodian, we can see that $UpTime(p)_{a_4-1} \leq TStamp(m)$ and since p stays sober in $[a_4, a]$, we know that $UpTime(p)_a \leq TStamp(m)$ – a contradiction. Therefore, a present message with timestamp equal to $TStamp(m)$ *is* sent by p. This implies, from Proposition 1 and examination of the custodian that $p \in View(m)$.
Subcase 2.2: Between stage a and its immediately previous crash, p sees no present messages. By Proposition 16, $UpTime(p)_a = TStamp(m)$. Let a' be a stage such that $Status(p)$ gets sober at a' and p stays sober in $[a', a]$. We know that p sends a new_gp message (call it m_1) with timestamp equal to $TStamp(m)$ in $a' - 1$. By Proposition 15, p sees m_1 at some stage $a_1 \in (a', a)$. Since p stays sober in $[a', a]$, we know that there exists a stage $a_2 \in [a_1, a)$ such that p sees m_1 at a_2 and PrMesSend_MS is fired at a_2, thus sending a present message with timestamp equal to $TStamp(m)$. This, however, implies that $p \in View(m)$. □

Theorem 3 *If* DEF$(GpId(p)_a)$, $p \in Members(p)_a$.

Proof: By induction on ρ.
Basis Case: Let s_0 be the initial stage of p. We know that $GpId(p)_{s_0}$ = undef – hence the claim is

vacuously true.
Induction Step: Assume that the claim is true for all stages $\leq a$. The only interesting case is if ChangeGp is fired at a. Otherwise, $Members(p)$ remains unchanged and by induction hypothesis, the claim is true. Let $V = View(Mes(p))_a$. We can see from examining rule HandlePresentMes that $Members(p)_{a+1} = V$. By Lemma 3, $p \in V$. Therefore $p \in Members(p)_{a+1}$ – which proves the claim. □

5.6 The Second Group of Theorems

Lemma 4 *If p sees a* present *message m at stage a and b is the first stage $> a$ such that*
1. p sees a present *message $m' \neq m$ at b*
2. p stays sober *in (a, b)*
then $0 \leq TStamp(m') - TStamp(m) \leq d_h$.

Proof: By Lemma 3, $p \in View(m)$. The claim then follows from Proposition 24. □

Lemma 5 *If p sees* present *messages $m \neq m'$ at stages $a < b$, p stays* sober *in (a, b) and $TStamp(m') - TStamp(m) < d_h$ then there exists a processor $q \neq p$ and stage s of q such that q sends a* new_gp *message with timestamp equal to $TStamp(m')$ in s.*

Proof: Without loss of generality, assume that m' is the first present message seen by p after m. By Lemma 3, $p \in View(m)$ and $p \in View(m')$. Since $p \in View(m)$ we know by Proposition 2 that there is a stage $a' < a$ at which p sends a present message with timestamp equal to $TStamp(m)$. Similarly, there exists a stage $b' < b$ at which p sends a present message with timestamp equal to $TStamp(m')$. We can see from Lemma 1 that $a' < b'$.

We can show that there exists no $c' \in (a', b')$ at which a present message with timestamp in $(TStamp(m), TStamp(m'))$ is sent. We proceed as follows. If there were such a stage then, by Proposition 1, a present message with timestamp in $(TStamp(m), TStamp(m'))$ is added to $InBox(p)$. Since p stays sober in $[a, b]$, we have by Proposition 13 that p sees a present message with timestamp in $(TStamp(m), TStamp(m'))$ at some stage $c \in (a, b)$. This contradicts our assumption that m' is the first present message seen by p since m.

We know, from examination of the rules, that at stage $a'+1$, $BCastTime(p)_{a'+1} = TStamp(m) + d_h$. If this broadcast is sent then $TStamp(m') = TStamp(m) + d_h$ which is not true. Therefore, the broadcast is preempted by a firing of PrMesSend_MS and the message sent has timestamp equal to $TStamp(m')$. This, however, implies that there exists a q and stage s at which q sends a new_gp message. □

Theorem 4 *For all stages $a < b$ of p, the following holds. If* undef $\neq GpId(p)_a \neq GpId(p)_b$ *then there is a stage $c < b$ such that either*
1. $c \geq a$ and $Status(p)_c \neq$ sober *or*
2. $c \geq a$ and $Type(Mes(p))_c =$ present *and $Members(p)_c - View(Mes(p))_c \neq \emptyset$ or*
3. p sees a new_gp *message m at c from some processor $q \neq p$ and $TStamp(m) \in (GpId(p)_a, GpId(p)_b]$.*

Proof: We have to show that whenever $GpId(p)$ changes, one of the above three scenarios will hold. Without loss of generality assume b is the first stage $> a$ such that $GpId(p)_b \neq GpId(p)_a$. This implies that the value of $GpId(p)$ changes at stage $b-1$. There are only two transitions that can change the value of $GpId(p)$
Case 1: Initialize. This implies that $Status(p)_{b-1} =$ recovered – which corresponds to the first

scenario.

Case 2: ChangeGp. For the rest of the proof, we use abbreviations t_{b-1} for $TStamp(Mes(p))_{b-1}$, V_{b-1} for $View(Mes(p))_{b-1}$ and M_{b-1} for $Members(p)_{b-1}$. If ChangeGp fires, $M_{b-1} \neq V_{b-1}$. If $M_{b-1} - V_{b-1} \neq \emptyset$, we have our second scenario. Therefore assume that $V_{b-1} - M_{b-1} \neq \emptyset$. We prove by contradiction that in this case, the third scenario holds.

By Proposition 6, there is a state $a' < a$ such that ChangeGp fires in a', $TStamp(Mes(p))_{a'} = GpId(p)_a$ and neither ChangeGp nor Initialize fires in (a', a). Since ChangeGp fires in a' and $TStamp(Mes(p))_{a'} = GpId(p)_a$, we know that $GpId(p)_{a'+1} = GpId(p)_a$. Since neither Initialize nor ChangeGp fire in (a', a), we know that $GpId(p)$ does not change in (a', a). Therefore b is the first state $> a' + 1$ in which the value of $GpId(p)$ is different from $GpId(p)_a$. Therefore, p stays sober in $[a', b]$.

By Proposition 4, $UpTime(p)_{a'} \leq GpId(p)_a$. Since p stays sober in $[a', b]$, $Uptime(p)_b \leq GpId(p)_a$. If p sent a new_gp message with timestamp in $(GpId_a, GpId(p)_b]$, we know by Lemma 1 that it would have been sent before b, which implies by Proposition 5 that $UpTime(p)_b > GpId(p)_a$ – which contradicts our earlier conclusion that $UpTime(p)_b \leq GpId(p)_a$. Therefore, p does not send a new_gp message with timestamp in $(GpId(p)_a, GpId(p)_b]$.

If p sees a new_gp message with timestamp in $(GpId(p)_a, GpId(p)_b]$ from some processor other than p, we have our third scenario. Therefore, assume that there is no stage $c < b$ and processor r that satisfies the third scenario.

Since ChangeGp fires at $b-1$, we know that p sees a present message with timestamp equal to $GpId(p)_b$ at $b-1$. Since $\textsc{Def}(GpId(p)_a)$ and since $GpId(p)_a = GpId(p)_{b-1}$, we know by Proposition 6 that there exists a stage $d < a < b-1$ at which p sees a present message and that p stays sober in (d, a). Let s be latest stage $< b-1$ such that p sees a present message at s and $Mes(p)_s \neq Mes(p)_{b-1}$.

Since we have assumed that p does not see any new_gp messages with timestamps in $(GpId(p)_a, GpId(p)_b]$, we have by Lemma 5 that $TStamp(Mes(p))_s = t_{b-1} - d_h$. Consider any $q \in V_{b-1} - M_{b-1}$. By Theorem 3, $p \in M_{b-1}$, therefore, $q \neq p$. All we have to show that there exists a stage $c < b$ such that p sees a new_gp message m from q at c and $TStamp(m) \in (GpId(p)_a, GpId(p)_b]$ and we have a contradiction.

Since $q \notin M_{b-1}$, we have by Proposition 7 that $q \notin View(Mes(p))_s$. Since $q \in V_{b-1}$ we know by Proposition 2 that there exists a stage e of q at which q sends a present message with timestamp equal to t_{b-1}. Let e' be a stage of q such that q gets sober at e' and stays sober in $[e', e]$.

It easy to see that q cannot send a present message in (e', e) that has timestamp less than t_{b-1}. We proceed as follows. If there were such a message (call it m_1) then we can see from examination of the algebra that $t_{b-1} - d_h \leq TStamp(m_1) < t_{b-1}$. Since q sends m_1, by Proposition 1, it is added to $InBox(p)$. $TStamp(m_1) \neq t_{b-1} - d_h$ since $q \notin View(Mes(p))_s$. $TStamp(m_1) \not> t_{b-1} - d_h$ implies, by Proposition 13, that p will see a present message in $(s, b-1)$ with timestamp equal to $TStamp(m_1)$. But we know that p does *not* see any present messages in $(s, b-1)$ other than $Mes(p)_s$ and $Mes(p)_{b-1}$.

Therefore the first present message sent by q since e' has timestamp equal to t_{b-1}. This however implies that $UpTime(q)_e = t_{b-1}$ which in turn implies that q sends a new_gp message with timestamp t_{b-1} st stage $e' - 1$. By Propositions 1 and 14, it follows that p sees this message – a contradiction. □

Lemma 6 *If p sees a* new_gp *message from q with timestamp t at stage a and a* present *message with timestamp t at stage $b > a$ and* $\textsc{Def}(Members(p)_b)$ *then $q \notin Members(p)_b$.*

Proof: There are two cases.

Case 1: p has seen no present messages between the time $Status(p)$ last became sober and stage b.

From examination of the algebra we can see that this implies that $Members(p)_b$ = undef - which proves the claim.

Case 2: p has received at least one present message between the time $Status(p)$ last became sober and stage b. Assume the claim to be false. In other words, assume that $q \in Members(p)_b$. Let $m = Mes(p)_b$. Let b' be the latest stage before b at which p sees a present message $m' \neq m$.

By Lemma 4, $TStamp(m) - TStamp(m') \leq d_h$. By Proposition 7, $q \in View(m')$. This implies, by Proposition 2, that q sends a present message at some stage c with timestamp equal to $TStamp(m')$. By Proposition 2, q sent a new_gp message at some stage d with timestamp equal to t. By Proposition 4, $UpTime(q)_c \leq TStamp(m') < t$. Since sending a present message does not change the value of $UpTime(p)$ we have $UpTime(q)_{c+1} < t$. Since q sends a new_gp message with timestamp t, we can see that $UpTime(q)_{d+1} = t$. Therefore, by Proposition 5, $c + 1 < d + 1$, hence $c < d$.

This implies that q crashes somewhere in the interval (c, d). Examining the scheduler, we can see that $C(q)_c \geq TStamp(m') - d_u$ which implies, by our failure and recovery constraints, that $C(q)_d > TStamp(m') + d_h > t$, which implies that $UpTime(q)_{d+1} > TStamp(m)$ – a contradiction. □

Theorem 5 *There exists a positive real d_j satisfying the following condition. If*
1. $Status(p)$ is set to sober *at stage a and*
2. p is correct in the interval $I = (C(p)_a, C(p)_a + d_j]$
then there exists a group id $g > C(p)_a$ such that for every q correct in I
$GpId(q)$ is set to g at some stage b of q with $C(q)_b \in I$.

Proof: Let $t = C(p)_a$. Let d_j be any constant $> 2d_n$. If $Status(p)$ is set to sober at stage a, then it sends a new_gp message m with timestamp $t + d_n$.

Since p is correct in the interval $[t + d_n - d_u, t + d_j]$, we can conclude that there exists a stage $b > a$ such that p sees m at b and PrMesSend_MS is fired at b. This causes a present message with timestamp $t + d_n$ to be sent.

Consider any q that is correct in the interval $[t+d_n-d_u, t+d_j]$. Since q is correct in the interval $[t + d_n - d_u, t + d_j]$, q sees m and and q sees a present message with timestamp $t + d_n$. Let c be a stage of q at which q sees a present message with timestamp $t+d_n$. Since p sends a present message with timestamp $t + d_n$, $p \in View(Mes(p))_c$.

By Lemma 6, either $Members(q)_c$ = undef or $p \notin Members(q)_c$. In either case, $Members(q)_c \neq View(Mes(q))_c$. This implies that at some stage $d > c$, $GpId(q)$ is set to $t + d_n$. Since q stays correct in $[t + d_n - d_u, t + d_j]$, we can conclude that $C(q)_d \leq t + d_j$. □

Lemma 7 *If p sees* present *message m at stage a and q sees m in stage a' then $GpId(p)_a = GpId(q)_{a'}$.*

Proof: By induction on $a + a'$.

Basis Case: $a + a' = 0$. This means that both p and q are at their initial stages. Since $Mes(p)_0 = Mes(q)_0$ = undef, the claim is vacuously true.

Induction Step: Assume that the statement is true for $a + a' < k$. Consider $a + a' = k$. Let $g = GpId(p)_a$ and $g' = GpId(q)_{a'}$. Assume $g \neq g'$. Without loss of generality, let $g' > g$. The only interesting case is if p sees a present message m at a and q sees the same message m at a'.

By Propositions 6 and 11, $g < TStamp(m)$ and $g' < TStamp(m)$. By Proposition 6, there exists stage $b' < a'$ at which q sees a present message with timestamp equal to g', that $GpId(q)_{b'+1} = g'$ and that there exists stage $b < a$ at which p sees a present message with timestamp equal to g and that p stays alive in the interval (b, a).

Therefore, by Propositions 25 and 13 and the fact that $g < g' < TStamp(m)$, there exists a stage $c \in (b, a)$ at which p sees a present message with timestamp equal to g'. By Proposition 26, $View(Mes(p))_c = View(Mes(q))_{b'}$. Therefore, $Mes(p)_c = Mes(q)_{b'}$.

By induction hypothesis, therefore, $GpId(p)_c = GpId(q)_{b'}$. This implies by Theorem 1 that $Members(p)_c = Members(q)_{b'}$ which implies that $GpId(p)_{c+1} = GpId(q)_{b'+1} = g'$. But $c + 1 \leq a$ and $GpId(p)_{c+1} > GpId(p)_a$. This contradicts Proposition 12. Therefore $GpId(p)_a = GpId(q)_{a'}$. □

Lemma 8 *Let I be the interval $(t, t + \delta)$, where $\delta > d_h + d_n + d_u$. Then, if q is correct in I, q sends a* present *message with timestamp $t' \in (t, t + d_h + d_u]$.*

Proof: Consider any q correct in I. Let s be the latest stage of q such that $C(q)_s \leq t$. Consider the value of $BCastTime(p)_s$. Since q is correct in I, we know that $UpTime(q)_s \leq t$. We also know that $Status(q)_s =$ sober.

Let $s' < s$ be a stage such that q becomes sober at s' and stays sober in $(s', s]$. This implies that q sends new_gp message m_1 with timestamp equal to $UpTime(q)_s$ at s' and stays sober at all stages in $(s', s]$.

By Proposition 1, m_1 is added to $InBox(q)$ at some stage in (s', s) and from the definition of correctness that there exists a stage $s_1 \in (s', s]$ such that q sees m_1 at s_1 and PrMesSend_MS is fired at s_1. If this were not the case, q will not be correct at s.

The firing of PrMesSend_MS sets $BCastTime(q)$ to some value that is not undef. Since q does not crash in $[s_1, s]$, we know that $\textsc{Def}(BCastTime(p)_s)$. Since q is correct at s, we know that $BCastTime(q)_s \geq C(q)_s$. By Proposition 20, $BCastTime(q)_s \leq C(q)_s + d_h + d_u$. By our initial assumption, $C(q)_s \leq t$.

We can also deduce that $C(q)_s \geq t - d_h - d_u$. We proceed as follows. Assume that $C(p)_s < t - d_h - d_u$. This means that $BCastTime(q)_s < t$. Consider the value of $BCastTime(q)$ at $s+1$. If it does not change at s, then $BCastTime(q)_{s+1} < t$. Since $C(q)_{s+1} > t$, this implies that q is incorrect at $s+1$ – a contradiction. Now consider the case where $BCastTime(q)$ *does* change at s. There are two clauses that can change $BCastTime(q)$ – PrMesSend_MS and PrMesSend_BS. If PrMesSend_BS fires at s, then we can see by examining the scheduler that $BCastTime(q)_s < t - d_h$. This means that $BCastTime(q)_{s+1} < t -$ which makes q incorrect at $s + 1 -$ a contradiction. If PrMesSend_MS fires at s then $BCastTime(q)_{s+1} = TStamp(Mes(q))_s + d_h$. We can see by examining the scheduler that $TStamp(Mes(q))_s < t - d_h$. Therefore $BCastTime(q)_{s+1} < t -$ which makes q incorrect at $s + 1 -$ a contradiction. By a similar argument, we can show that $BCastTime(p)_s > t - d_h$.

What is the value of $BCastTime(p)$ at $s+1$? The highest possible value of $C(q)_s$ is t. Hence by Proposition 20, $BCastTime(q)_s \leq t + d_h + d_u$. Since q is correct at $s+1$, $BCastTime(q)_{s+1} > t$. Using an argument similar to the one in the previous paragraph, we can show that $BCastTime(q)_{s+1} \leq t + d_h + d_u$. Therefore $BCastTime(q)_{s+1} \in (t, t + d_h + d_u]$.

Since q stays correct in this interval, this broadcast will either be sent or will be preempted by the arrival of a new_gp message whose timestamp is less than $BCastTime(p)_{s+1}$. Since q stays correct in $(t, t + d_h + d_u]$, the timestamp of this new_gp message is $> t$. Therefore, q sends a present message with timestamp $t' \in (t, t + d_h + d_u]$. □

Theorem 6 *There exists a positive real d_f satisfying the following condition. If*
1. p fails at stage a and
2. $I = (C(p)_a, C(p)_a + d_f]$ and $GpId(p)_a = g \neq$ undef
then there exists $g' > g$ such that
1. p never joins group g' and

2. for every q that joins group g and is correct in I
$GpId(q)$ is set to g' at some stage b of q with $C(q)_b \in I$.

Proof: Let $t = C(p)_a$. Let d_f be any real constant $> d_h + d_u + d_n$. Since p fails at time t, we have by Lemma 1 and our failure and recovery constraints that p cannot send a present message with timestamp in $(t, t + d_h + d_u]$.

Consider any q correct in I. By Lemma 8, q sends a present message with timestamp $t' \in (t, t + d_h + d_u]$. By Proposition 1 and the definition of correctness, every processor that is correct in I sees a present message with timestamp t'. By Proposition 26, all processors correct in I will see the same present message m with timestamp t'.

Consider any $r \neq q$ that stays correct in I. Since both q and r stay correct in I, both of them see m. We know that since p does not send a present message with timestamp t', $p \notin View(m)$. Since q is correct in I, there will be a stage b_q in which q sees m, $C(q)_{b_q} \in I$ and HandlePresentMes fires. There is a similar stage b_r of r. By Lemma 7, we know that $GpId(q)_{b_q} = GpId(r)_{b_r}$. Therefore, by Theorem 1, $Members(q)_{b_q} = Members(r)_{b_r}$. We also know that $p \notin View(m)$. Therefore, $GpId(q)_{b_q+1} = GpId(r)_{b_r+1}$ and p is not contained in either $Members$ set. □

References

[1] E. Börger, Y. Gurevich and D. Rosenzweig. The Bakery Algorithm: Yet Another Specification and Verification. To appear in E. Börger, editor, *Specification and Validation Methods for Programming Languages and Systems*, Oxford University Press, 1994.

[2] F. Cristian. Reaching Agreement on Processor-Group Membership in Synchronous Distributed Systems. *Distributed Computing*, 6:175–187, April 1991.

[3] Y. Gurevich. Evolving Algebras: An Attempt to Discover Semantics. In G. Rozenberg and A. Salomaa, editors, *Current Trends in Theoretical Computer Science*, pages 266–292. World-Scientific, 1993.

[4] Y. Gurevich. Evolving Algebras 1993: Lipari Guide. To appear in E. Börger, editor, *Specification and Validation Methods for Programming Languages and Systems*, Oxford University Press, 1994.

[5] J. Huggins. Kermit: Specification and Verification. To appear in E. Börger, editor, *Specification and Validation Methods for Programming Languages and Systems*, Oxford University Press, 1994.

[6] F. Jahanian, R. Rajkumar and S. Fakhouri. Processor Group Membership Protocols: Specification, Design and Implementation. In *Symposium on Reliable Distributed Systems*, 1993.

[7] A. M. Ricciardi and K. P. Birman. Using Process Groups to Implement Failure Detection in Asynchronous Environments. In *11th ACM Symposium on Principles of Distributed Computing*, pages 341–353, 1991.

[8] W. R. Stevens. *UNIX Network Programming*. Prentice-Hall, 1990.