Flexible Intrusion Tolerant Group Membership Protocol

Narasimha Prasad Subraveti, Soontaree Tanaraksiritavorn, and Shivakant Mishra
Department of Computer Science
University of Colorado, Campus Box 0430
Boulder, CO 80309-0430, USA.
Email: [subravet | tanaraks | mishras]@cs.colorado.edu

Abstract

Intrusion-tolerant group membership protocols constitute an important part of intrusion-tolerant group communication systems. These protocols maintain a consistent system-wide view of correct group members in the presence of malicious failures. This paper presents a new intrusion-tolerant group membership protocol. This protocol provides two unique features. First, it introduces a new membership state called a suspended membership state. A suspended group membership state provides a good balance between the amount of time a malicious/compromised group member gets to launch attacks before being removed from the group and the increased vulnerability to denial-of-service attacks if a suspected member is removed too early from the group. Second, it introduces a clean, logical separation between the functionality of detecting malicious processes and removing malicious group members from the group. This logical separation aids in simplifying the group membership protocol design and efficiently detecting suspicious process behaviors. The protocol has been implemented and the paper provides a detailed performance analysis.

1 Introduction

High availability and trustworthiness are perhaps the most important requirements of modern computing systems. Group communication services have been successfully used to construct a large number of highly available, fault-tolerant, and high performance applications that can run in a heterogeneous, wide-area network [1, 2, 3, 7, 17, 20, and 33]. Informally, these services are comprised of a set of protocols that provide system-level support for object replication in the presence of node and communication failures.

A group membership protocol is an important part of a group communication service. Informally, it maintains a system-wide, consistent view of correct group members at any point in time. Designing a correct group membership protocol is very complex because of the difficulty in distinguishing between a process failure and a communication failure in an asynchronous distributed computing system. A large number of group membership protocols have been designed and implemented over the past 15 years [1, 2, 3, 8, 17, 20, and 33]. Almost all of these protocols (exceptions: [16, 25, 31]) have been designed to tolerate only crash/performance failures [9] of communication or computing components. Given the recent threat of security attacks with malicious intentions on modern computing systems, it is important that the next generation of group communication services be designed to tolerate more complex types of component failures, such as Byzantine failures.

This paper presents the design, implementation, and evaluation of a group membership protocol that can tolerate Byzantine failures. This group membership protocol is a part of a trustworthy group communication system called FITS (**F**lexible **I**ntrusion **T**olerant Group Communications

System) that we are currently building. FITS consists of a trustworthiness detector [21], a trustworthy group membership protocol (presented in this paper), and an atomic broadcast protocol. It provides two unique features. First, unlike most group communication services proposed in the past, FITS is designed to support object replication in the presence of Byzantine failures. Second, FITS explicitly incorporates techniques to address the fundamental problem of correctly detecting Byzantine failures in a timely manner. It is intended to be a practical solution for providing object replication support in a hostile computing environment, where the security of a small number of nodes may be compromised and adversaries may attempt to launch malicious security attacks including denial-of-service attacks.

The group membership protocol presented here provides group membership support in the presence of Byzantine process failures. In particular, there are three important contributions that this group membership protocol provides. First, it introduces a new concept of *suspended group membership state*. A suspended group membership state provides a good balance between the amount of time a malicious/compromised group member gets to launch attacks before being removed from the group and the increased vulnerability to denial-of-service attacks if a suspected member is removed too early from the group. Suspected membership state is a new concept that has the potential to significantly improve the robustness of group communication systems. The second contribution of this paper lies in introducing a clean logical separation between the functionality of detecting malicious processes and removing malicious group members from the group. Existing intrusion-tolerant group communication systems tend to integrate the failure detection process with group membership or atomic broadcast protocol. This logical separation aids in simplifying the group membership protocol design and efficiently detecting suspicious process behavior. Finally, a prototype of the proposed group membership protocol has been built, and the paper provides performance evaluation. This is important, because there is a significant lack of any quantitative analysis of the cost of intrusion tolerance at present. There is a critical need for performance data quantifying the cost of providing intrusion tolerance in modern computing systems. The performance evaluation discussed in the paper has provided useful insight into detecting malicious behaviors of group members and removing suspected processes from the group.

The rest of this paper is organized as follows. Section 2 describes some of the related work in building intrusion-tolerant group membership protocols. Section 3 describes the main motivation behind designing a new intrusion-tolerant group membership protocol. This section also introduces the new suspended group membership state. Section 4 describes the details of our group membership protocol and Section 5 describes the prototype implementation and performance evaluation of the protocol. Finally, Section 6 concludes the paper.

2 Related Work

Group Membership Protocols are a common paradigm in providing process group level consistency. To build highly available and correct services, there was a need to replicate servers. This led to development of fault tolerant group communication systems. Several group communication systems were built to handle different kinds of faults. The earlier efforts to build fault tolerant reliable group communication systems were equipped to only handle crash failures. Some of the earlier systems that handled crash failures are Isis [3], Consul [17], Totem [1], Ensemble [28], Horus [33], Spread [2], Pinwheel [10] and Timewheel [20].

Related projects aiming to provide intrusion tolerance at the middleware level using a group communication service include Rampart [29, 30, 31], ITUA [25, 26, 27], and SecureRing [15,

16]. Practical Byzantine Fault Tolerance [4] is another work that describes a state machine replication protocol that correctly tolerates Byzantine faults. Rampart [29] is the first group membership protocol that aims to tolerate malicious intrusions. It uses a three-phase commit protocol and provides strong consistency guarantees. Rampart's fault detection mechanism relies on fault detection from an external fault detector. The Three-phase commit protocol starts when at least one-third of group members agree to remove a faulty member. Rampart uses a manager-based structure, where the manager is responsible for suggesting the new view. Rampart can handle only one fault at a time. If multiple failures occur, they are treated one by one in a sequential order.

SecureRing [16] uses the logical token ring mechanism to disseminate data and maintain membership information. SecureRing also tolerates malicious faults and it claims to have lower cost of digital signatures due to message packing. Message digests in a signed token allow a single digital signature to cover multiple messages. Like Rampart, the membership protocol starts when at least one-third of group members agree to remove a faulty member. SecureRing can handle multiple faults during a single membership round by adding all faulty members into a to-be-remove set.

ITUA [26] adopts the Rampart approach, but adds capability to handle multiple faults in a single round. ITUA group membership protocol provides a suspect interface [of the form *suspect* to all the protocols in its group communication stack with the capability that any protocol in that stack (including the group membership protocol) can invoke the interface if it suspects the peer protocol has deviated from its specification. The leader of the group initiates the group membership protocol when at least one-third of the group member suspects a faulty member. The group membership ends when all faulty members are removed from the group and the new view is installed. The protocol uses many time-outs to measure faulty behavior; hence it can suffer from the long idle time of applications during view installation if new faulty member is detected when the protocol is in the last phase.

3 Motivation

Intrusion-tolerant group membership protocols must deal with two difficult issues. First, they need to deal with the difficulty of detecting Byzantine failures. In particular, group membership protocols are time consuming and any application implemented on top of group communication system becomes unavailable while the underlying group membership protocol is in progress. As a result, these protocols must ensure that they are invoked only when it is fairly certain that the security of a member has been compromised. Second, they must ensure that a compromised group member gets as little time as possible to launch malicious attacks on the group communication system. In particular, a compromised group member should be removed from its group as soon as possible, so that it does not get much time to inflict damages in the group communication system. These two issues are contradictory in nature. The first issue entails that the group membership protocol be invoked only after it is fairly certain that the security of a group member has been compromised. However, because of the inherent difficulty in detecting Byzantine failures, it may take a relatively long period of time to be fairly certain that the security of a group member has been compromised. Thus, the first issue essentially results in delaying an invocation of a group membership protocol. The second issue, on the other hand, entails that a group membership protocol be invoked as soon as there is even a slight suspicion that a group member is faulty [32].

For a group membership protocol to start, more than 1/3rd of the total number of group members should agree on the removal of the suspected group member. The problem with this requirement is that this can result in a long waiting period for taking action on suspected group members. We observed in [19] that the detection time to suspect a compromised group member by 1/3rd of the group members is significantly larger than the detection time by a first group member. During the time interval between the time the first group member suspects a compromised group member and the time when 1/3rd of the group agree on the suspicion of that member, there is a high potential that this "suspect" member can harm the system. This would be potentially dangerous as the suspected group member cannot only harm the system but also spread its effects and consequently achieve inconsistencies in the group (despite its removal from the group). All existing intrusion tolerant group membership protocols [16, 26, and 29] suffer from this problem. This issue can be addressed by making a slight modification to the mechanism of initialization of the group membership protocol. The mechanism of preventing/limiting the effects of a suspected member from harming the group activity is handled by the Pre-Membership phase of our protocol.

We introduce this phase so that a malicious group member who is suspected by another group member is immediately suspended ("ineffective with respect to the group activity") on consent from the sequencer (leader) so that it is prevented from causing any damage to the group membership and hence the application service built on top of it. A member that is suspended enters a new membership state called *Suspended Membership State*. This mechanism waits to start the actual membership (Three Phase Commit) protocol only after it is guaranteed that the members in the group have suspected the actual faulty/malicious processes and have the agreement from more than 1/3rd of the group in doing so. If there is a case of false alarm introduced by the leader or by any other member, the protocol reinstates the suspended process when the sequencer (leader) is not able to gather enough proof to brand the member as faulty/malicious. We proceed to remove the faulty/malicious members through the Three Phase Commit protocol.

4 Protocol Details

4.1 Assumptions

In our protocol, we assume that there are 'n' members in the group. Each view of a group is represented by a group id (gid). We assume a timed asynchronous communication model [11], where processes in the group do not need to synchronize clocks. Hence the model is asynchronous; but we use timeouts as a means to handle the difficult problem of distinguishing between a communication failure and a failure/corruption of the group member. We assume that any group member can suspect failure/corruption of another group member.

We assume the existence of a trustworthiness detector and an atomic broadcast protocol. The trustworthiness detector of FITS [21] has been specifically built for a group communication system. Its design is based on two important principles: (1) focusing on observable effects, and (2) detection in depth. This detector is generic in the sense that it is independent of the actual broadcast or group membership protocol being supported by the group communication system. It acts on inputs received from a variety of sources, including the atomic broadcast and group membership protocols. Based on the information received from all these sources and a policy file, the detector raises a suspicion event whenever it suspects a group member. The detector provides a generic interface through which an atomic broadcast or a group membership protocol can communicate any abnormal behaviors they observe.

The atomic broadcast protocol of FITS is a rotating sequencer-based protocol similar to the Pinwheel atomic broadcast protocol described in [10]. This protocol adopts a rotating sequencer/leader mechanism, i.e. the group member that assigns global ordering on broadcast messages changes on every broadcast of a control message. The sequence of the leader is fixed in advance based on its unique rank. The member with highest rank acts as the leader first; the member with next lower rank acts as the leader next, and so on. This rotating feature helps in balancing the processing load, which is the main limitation of a fixed leader-based broadcast protocol. Since the leader performs the task of a sequencer in this reliable broadcast protocol, we use the terms leader and sequencer interchangeably. The member's fault detector (trustworthiness detector) invokes the suspect function thereby instructing the group membership protocol to broadcast a suspect message to all the members of the group. We assume a public key cryptosystem where each group member possesses a private key PK_i known only to itself with which it can sign messages digitally. We also assume that each group member can obtain the public keys of other group members as needed with which it can authenticate the messages.

Each Member maintains four lists i.e. Member List (ML), Suspected List (SL), Suspended List (SusL) and Faulty List (FL) to keep track of the status of all members in the group.

1. **Member List (ML):** List of all correct members that are currently part of the group. For convenience let us say that there are 'n' correct members.
2. **Suspected List (SL):** List of members that is suspected as being faulty/corrupt. Each member has its own copy of the Suspected List.
3. **Suspended List (SusL):** List of members that have been suspended by the sequencer. Now each member 'i' in the above list is the header to a list of all members that agree on the suspension of the i^{th} Suspended member. Let us call this list- SusL[i].
4. **Faulty List (FL):** List of members that are considered faulty and are to be removed by the group membership protocol.

Members discard any messages sent from any member in SusL. Each group member has to agree to another member's view of status of the group. The system is in the stable and consistent state when there is agreement on the above. We adopt a sequencer based Pre-Membership protocol until the Three-phase Commit protocol is initiated. The output of the group membership protocol at any time will be the view generated from the two lists i.e. the member list (ML) and Faulty List (FL).

4.2 Group Membership Protocol

The intrusion tolerant group membership protocol can be classified into three independent protocol modules that are closely coupled with the reliable broadcast mechanism. We classify them as follows:

Pre-Membership Suspension Protocol: This protocol is started when members of a group raise suspicion/suspicions on one or more of the current group members. The other group members and the leader (sequencer) decide on whether to remove the suspected member or reinstate the suspected member in case of a false alarm.

Three Phase Commit Protocol: It is agreed upon that one or more group members have to be removed and the suspended list is empty. The members in the faulty list (FL) are to be removed from the group. These members are removed based on the standard 3 Phase commitment strategy with the following phases:

- Consensus on New group
- New View Agreement
- Commit and Message Stabilization.

Group Join Protocol: The group members have to agree upon any new member joining the group. A mechanism to ensure the group members join correctly is to allow more than one group member to agree on allowing the new member to join the group. We devise a method of incorporating "Trusted Group Join" using the standard three-phase commitment strategy. The members in the group go through the Pre-Join phase before the actual Join phase where the member joins the group.

4.3 Pre-Membership Suspension Protocol

An important requirement to build an intrusion tolerant service is to ensure that the application is providing service to its clients as long as possible. This leads to a situation where it is important to remove the faulty/malicious member as soon as possible but without sacrificing on correctness of choice of the faulty/malicious member. We introduce the Pre-Membership phase where the application is still providing service to the clients but a suspension protocol would be run in the background. The state transition diagram of the pre-membership phase is shown in Figure 1.

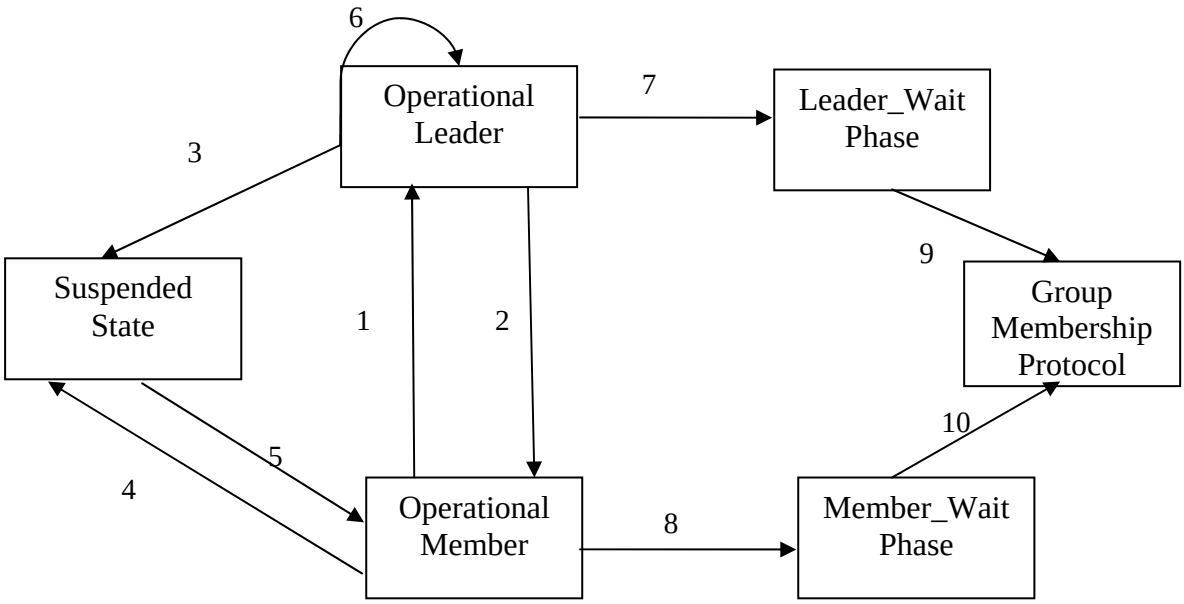

Figure 1: State Transition Diagram for the Pre-Membership Suspension Protocol

1) Operational Member multicasts a Control message (Suspect, Ordering)
2) Operational Leader multicasts Control message (Update_Ordering, Suspect, Suspend or Reinstate).
3) Operational Leader multicasts Suspend (m') and adds m' to its SusL.
4) Operational Member receives Suspend (m') from Leader and adds m to its SusL.
5) Operational Member receives Reinstate (m') from Leader and removes m from its SusL.
6) Operational Leader receives Suspect (m') message from member m

7) Leader moves to wait phase when gathering |Suspend[i]| > n/3 for some i
8) Member moves to wait phase when gathering |Suspended[i]| > n/3 for some i
9) Leader multicasts Suggest_New_View message
10) Member receives Suggest_New_View message

Since we adopt a rotating sequencer in our multicast protocol, it is imperative to handle the different cases that arise carefully. We assume that a group is a priory in existence, and its members are ranked by the protocol initiator. This protocol begins when a member in the group of processes issues a suspect message based on the suspicion raised by the Trustworthiness Detector (TD) [21]. A group member 'm' receives a suspect event from its TD. The member broadcasts a suspect (m, m') message i.e. *member m suspects m'* to the sequencer (Operational leader). It updates its Data Structures and starts a timer - Snd_Suspect [m']. It waits for the sequencer to respond to the suspect message. On expiry of the timer (which means there is no response from the sequencer), TD of the member is informed of a suspicion on the sequencer. The sequencer waits for one or more group members (other than its own suspicion) to receive suspect (m, m') messages on the suspected member by populating a data structure called Agreement List (AL). The value of |AL [i] = m'] is an indicator of how many members agree on the suspicion of a particular member m' and when this value crosses a threshold (flexibly set by the sequencer). The sequencer updates the suspended list by checking if there is an entry of m' in SusL[i] where i = m'. If there is none, add SusL [m']. Then, add the signed message, Suspect (m, m') to SusL [m']. We add the signed message as a confirmation that there is a member m who suspects m'. The sequencer finally multicasts a Suspend (m, m') message i.e. *member m' is suspected by m.* Each member starts the Recv_Suspend (m') timer on receipt of this message. If this timer expires, it means m' is in the suspended state for quite a period of time without further suspicions on the member. So, we assume it is wrongly suspected and we should remove it from the suspended list so that it can participate in future communication of the group. The sequencer then issues a reinstate (m') message and performs an out-of-band state transfer to the rejoining member.

All operational members who receive a Suspend (m, m') message from the sequencer update their own SusL [m'] lists; start the timer Recv_Suspend [m'] and check if the number of suspicions on a particular suspended member exceeds $1/3^{rd}$ the total number of the group. If this timer expires, and the member is not reinstated, the member informs its trustworthiness detector to raise a suspicion on the sequencer as a suspended member should be either deemed faulty or be reinstated into the group within the expiry of the above timer. We check if the number of members under SusL [m'] is more than $1/3^{rd}$ +1. We require to use this value because of the assumption that there can be at most one-third of group members that can turn malicious and the value $1/3^{rd}$ +1 means that at least one correct group member agree that m' is faulty. The operational member enters the Member_Wait phase whenever the number of members in its SusL [m'] exceeds $1/3^{rd}$ +1. Similarly, the Leader (sequencer) enters the Leader_Wait phase when it has received support for the suspension from more than $1/3^{rd}$ of the group members and starts the Leader_Wait timer. On expiry of the timer the suspected members are moved into either the faulty list or are reinstated into the group.

When this wait phase is reached, the member stops accepting application messages, the sequencer role does not rotate when it reaches the Leader_Wait phase. The suspected members are updated as being either faulty members in the FL list of each of the members or reinstated in the event of Leader_Wait timer expiry. Each member knows who the next sequencer in the group is, based on the rank. Once the sequencer enters its wait phase, it issues a Suggest_New_View message to the group. Each member also maintains a queue of members that are to be reinstated so that when it assumes the role of the Sequencer, it has to reinstate all the members who are in the Reinstate

Queue (RQ). Another important function is the clearance of the buffers created to model suspicion supports in Suspended List. A buffer clearance timer is started when this list is formed and it expires when the members are moved either to the Faulty List (FL) or are reinstated. This phase culminates when the leader issues Suggest_New_View message; thereby indicating the beginning of the membership protocol to remove the faulty/malicious members.

4.4 Three Phase Commit Protocol

The Three Phase Commit protocol starts when any group member receives a Suggest_New_View message from the Leader. The protocol we use to remove the faulty/malicious members is very similar to Rampart, except that we handle multiple failures received as input from the Pre-Membership Suspension protocol. The state transition diagram explains the stages of a group member while executing the Three Phase Commit protocol. It is important to know the leader role does not change once this protocol begins and no suspect messages are entertained except for suspicions on the Leader. Hence membership failures raised in the Pre-Membership Suspension protocol guide the removal of the members in the Faulty List (FL). The leader election protocol is one of the most common problems in distributed systems. So we adopt the standard strategy of deputy initiating a leader election protocol and ensuring message consistency.

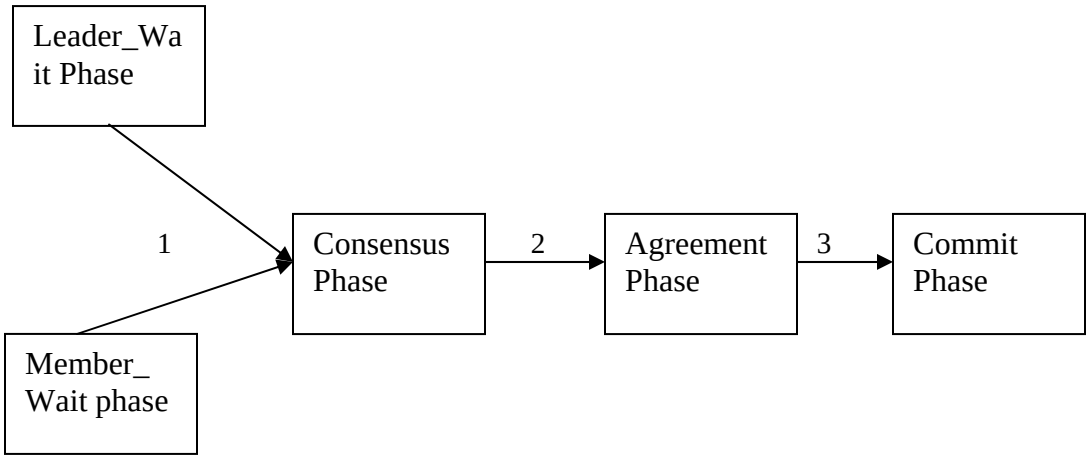

Figure 2: State Transition Diagram for a Group Membership Protocol

1. Received Suggest_New_View from the leader(sequencer)
2. Received Ready_to _Commit from the leader(sequencer)
3. Received Commit_New_View from the leader(sequencer)

The first phase of the Three Phase Commit protocol is the Consensus phase. This is the phase where the sequencer broadcasts to all the members of the new view, |ML-FL-SusL|. As soon as a group member enters this phase, it starts a timer for the leader to proceed to the next phase. Once this timer expires, and if the sequencer does not issue the "Ready to commit" message, the Trustworthiness Detector [21] is informed to raise a suspicion on the leader thereby starting the leader election protocol.

During the Consensus phase, the group member broadcasts an acknowledgement if it agrees on the new view. It starts a timer on this broadcast, which on expiry informs the TD to suspect the

Leader as faulty or malicious. Once the Leader receives the acknowledgments from majority of correct members (more than $2/3^{rd}$ of the current value of |ML-FL-SusL|), then the Leader sends a "Ready to Commit" message. On receipt of this message, the group members move to the "Agreement" phase. The group members start a timer so as to ensure the correct operation of the Leader and raise a suspicion if its behavior is inconsistent with the group. The Leader must issue a "Commit" message to all the members of the group within the timer expiry. During the Agreement phase, the Leader waits for more than $2/3^{rd}$ of the |ML-FL-SusL| (majority response) to respond to its "Ready to Commit" message. On receipt of majority response, the Leader issues a Commit message. On receipt of the Commit message, the protocol commits to the new view and each group member update the membership status of the group accordingly. The Leader ensures final message stabilization at the end of this protocol.

4.5 Group Join Protocol

One of the requirements of a group membership protocol is to allow membership joins provided the members are correct and verified to be so. This protocol provides a mechanism for new members to join securely. We assume the existence of a Public Key infrastructure. An important thing to note is the join protocol is an independent protocol i.e. it cannot run in the background while a member or a set of members are being suspected/removed. It is clear that when a member has to be added to a group or a set of processes, then it should be efficiently authorized to join and also authenticated. One of the key issues while building a join protocol is the matter of trust. If the joiner(i.e. the member who intends to join) contacts one member in the group and issues proof to the group, that he is correct and has authenticated with a member of the group, then we cannot automatically include the process into the group. This is because the group member that took part in the join protocol may have been suspected/deemed faulty. The alternative solution is entrust the protocol to rely on the Byzantine agreement i.e. more than $1/3^{rd}$ of the group have to agree on the new member to be included in the group. This ensures that atleast one correct member has accepted the member to join the group. The state transition diagram for each group member is given below. The dashed-joiner and interactions with operational member/leader (in figure 3 shown below) indicate the control flow in the Pre-join protocol.

Pre-Join protocol:

We adopt the strategy of having an out of band mechanism through which the member that intends to join the group (joiner) contacts various members of the group and issues his intent to join the group with his signature. This is the start point of our Join protocol. The biggest challenge during the Pre-Join phase is to securely join a new member to the group with proof that more than $1/3^{rd}$ of the group having accepted his intention to join. This can be solved by a challenge-respond protocol between the leader and the joiner of the group. Let PK_j, PK_m,PK_l represent the public key of the joiner, operational member and operational leader respectively.
The joiner i.e. the joining member contacts one or more members of the group with a Join_Interest (gid, PK_j) message i.e. SK_j (gid, PK_j). The members of the group respond back with a ticket of their own (although without timestamp). This ticket is the acknowledgement to the Join_Interest message i.e. Ack_Join_Interest (memberid, SK_m (gid, PK_j)). This is collected from more than $1/3^{rd}$ of the group members. The next phase is the challenge-respond phase where the leader challenges the joiner before allowing him to join the group. The joiner sends a Request_to_Join (gid, PK_j) message. The leader generates a challenge by creating a nonce which is encrypted with PK_j i.e. PK_j (nonce) in response to the Request_to_Join message and requests the joiner to send the nonce signed with the leader's public key. The joiner responds to the challenge by extracting the nonce and sending this back encrypted with the public key of the

leader. After this message, the joiner is authenticated by the leader. So, after the leader issues an acknowledgement of receipt of the response to the challenge he set, then a signed message (gid, X) is sent by the joiner, where X is the collected Ack_Join_Interest tickets. This cryptographic authentication mechanism ensures protection from replay attacks or any other man-in-the-middle attacks. The leader issues an Ack_Joiner_Request_to_Join message to the joiner and starts the Group Join protocol. Since, the joining member is still not a part of the group, it waits until it receives the New_Group message.

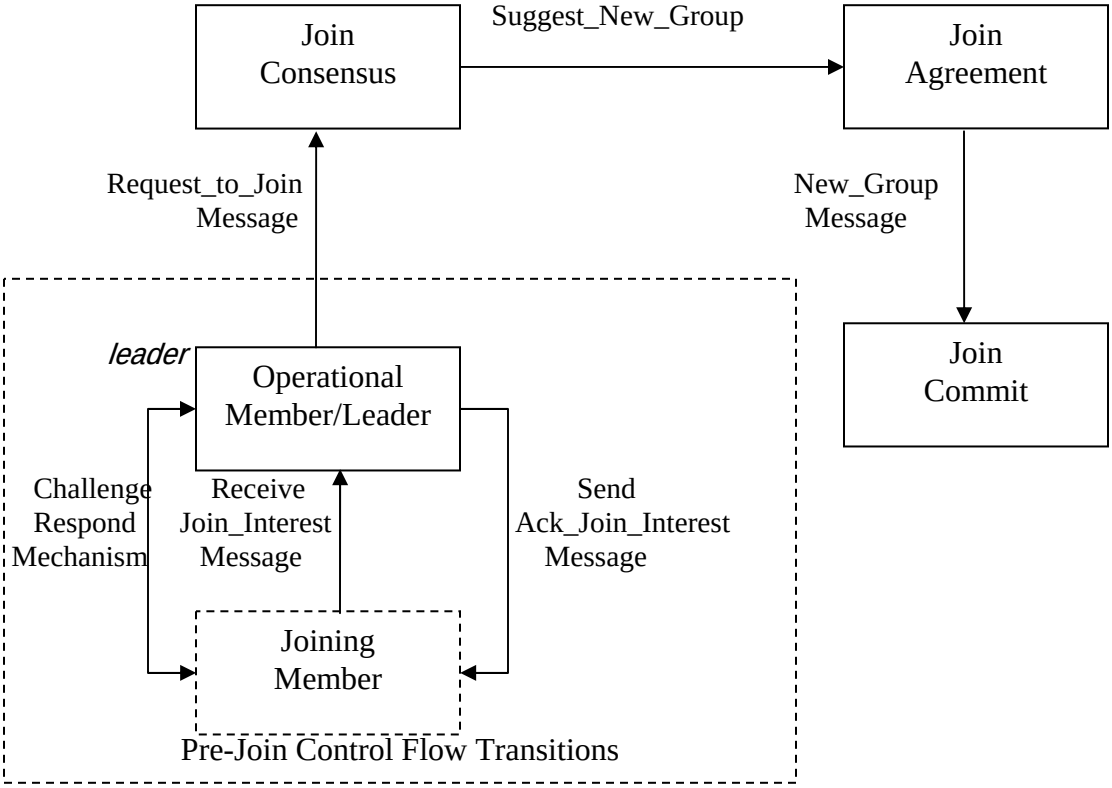

Figure 3: State Transition Diagram of a Group-Join protocol with the Control flow transitions in the Pre-join phase.

Join Protocol:

We propose a join protocol which is similar to the Three Phase Commit protocol used to remove group members. So each member goes through the following phases once the joiner is approved to join the group by the Pre-Join phase. On receipt of the signed Request_to_Join from the leader, each group member moves to the Join Consensus phase. The current sequencer performs the role of leader and allows the joiner to enter the group when more than 1/3rd of the total group members have acknowledged the Request_to_Join message. On receipt of more than 1/3rd the number of Ack_Request_to_Join messages the sequencer issues a Suggest_New_Group message (gid, pid of the new joiner, signature of joiner, proof for New View) and all group members enter the Join Agreement phase on receipt of the Suggest_New_Group message. The application is stopped at this phase. The Leader awaits Suggest _Ack_New_Group messages from more than 2/3rd the number of members in the group. On receipt of more than 2/3rd Suggest_Ack_New_Group messages, the leader issues a New_Group view message to the group. All members move to the Join Commit Phase on receipt of this message. The leader should also transmit state of the

protocol to the new group member. We assume this is done by an out-of-band mechanism of state transfer. The group members form the new view, include the group member; but they do acknowledge New_Group messages. On receipt of this message all members check their state and see to that it's up to date with inclusion of the new group member.

5 Implementation and Performance

As observed in [30], all intrusion-tolerant group membership protocols adopt a three-phase commit protocol mechanism to remove failed members from a group. Our protocol also makes use of this mechanism. The main difference between our group membership protocol and other intrusion-tolerant group membership protocols is the introduction of the new state called suspended group membership state. This state is introduced to balance the effect of false alarm and tardiness of confining malicious behaviors. So, we have focused on the effect of this new suspended group membership state in our performance measurements. In particular, we want to measure the additional cost imposed by the suspended group membership state and the benefit that we can get from it.

We implemented our intrusion group membership protocol in C++ in NS2 that was developed at UC Berkeley. We also implemented the rotating-sequencer based atomic broadcast protocol using negative acknowledgement mechanism. To measure the performance of our group membership protocol, we experimented with groups of sizes four to twelve. All group members join a single multicast network. During group formation, each member is assigned a unique rank i, where $i = 1$ to n and n is the group size. The member with the highest rank is the first leader. We simulated failures of one and two members in the group. For one member failure, we triggered a member to send Suspect (m, m') immediately after the group was formed. No background traffic was generated in this test. So the performance that we measured represents the protocol cost only. Based on our failure model, a group whose size is larger than six members can tolerate two failures. For these groups, we triggered two member failures detected by different members in the group at the same time.

In order to measure the effect of the suspended group membership state, we compare our protocol with a standard three-phase commit based group membership protocol (e.g. [29], termed as STD GMP in Figures 4 and 5). For STD GMP, the protocol cost consists of the time interval starting from the beginning of the first phase until the end of the third phase. Recall that the first phase starts after one-third of the group members have suspected a faulty member. For our protocol (termed as FITS GMP in Figures 4 and 5), the protocol cost consists of the time interval starting from the beginning of the pre-membership suspension phase until the end of the third phase of the three-phase commit protocol. Recall that the pre-membership suspension phase starts when a first member suspects a group member and sends a suspect message.

Figures 4 and 5 show the protocol cost in milliseconds for removing one and two group members respectively. The line marked "STD GMP" illustrates the protocol cost for a standard three-phase commit group membership protocol, while the line marked "FITS GMP" illustrates the protocol cost for our intrusion-tolerant group membership protocol. The difference between these two lines is the time a compromised group member remains suspended before being considered faulty.

There are three observations we make from these two figures. First, the protocol cost increases with increase in the size of the group. This is an expected behavior, because the number of messages exchanged in the group by the group membership protocol depends directly on the size

of the group. Second, there is a sudden increase in protocol cost when group size increases from 6 to 7 and from 9 to 10. This is because the three-phase commit protocol starts only after one-third of the group members have suspected a faulty member. This number (one-third) increases when the group size increases from 6 to 7 and from 9 to 10.

Finally, the time a faulty group member remains in suspended group membership state varies from about 50 milliseconds to 100 milliseconds. This is the time interval during which this member is considered a normal (non-faulty) group member by earlier intrusion-tolerant group membership protocols. In other words, the introduction of the suspended group membership state results in reducing the time a compromised group member gets to launch attacks by as much as as 100 milliseconds. Notice that the application running on top of the group communication system observes a similar behavior, whether FITS group membership protocol is used or a standard three-phase commit group membership protocol is used. We can see that the introduction of the suspended group membership state does not adversely affect an application, but does reduce the amount of time a compromised group member gets to launch attacks and damage the state of a group communication system.

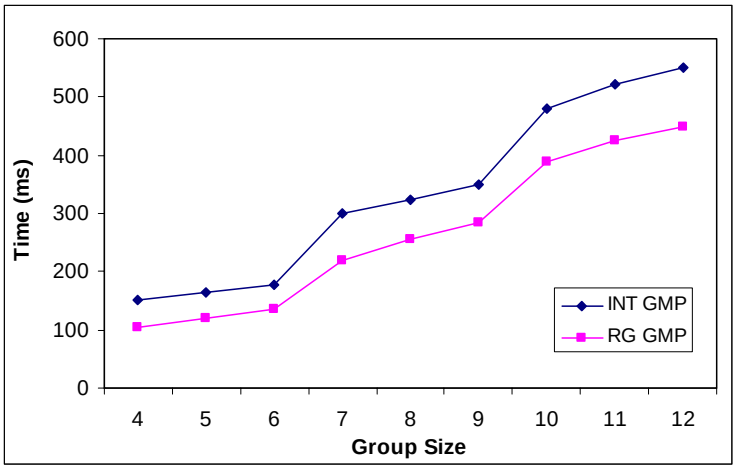

Figure 4: Protocol cost when one member in the group is faulty.

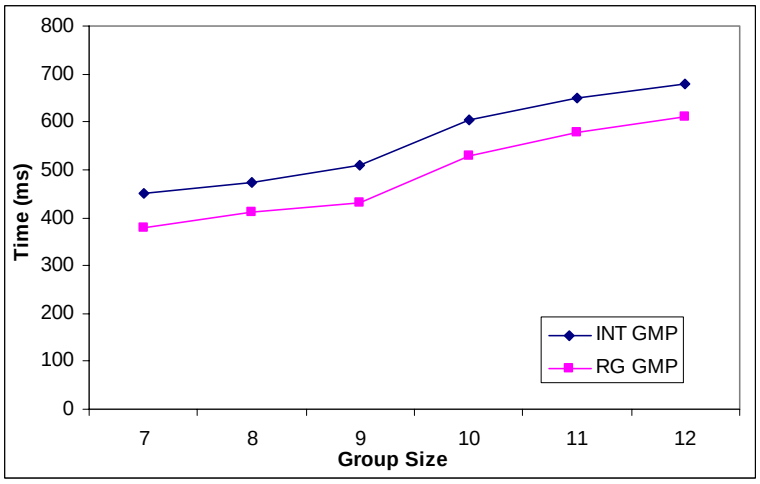

Figure 5: Protocol cost when two members in the group are faulty.

6 Discussion

With increasing use of computing systems to construct critical as well as non-critical applications, it is clear that high availability and trustworthiness are the most important requirements of modern computing systems. Building intrusion-tolerant group communication systems is a step towards fulfilling these requirements. A group membership protocol is a very important and perhaps the most complicated part of a group communication system. Research in building intrusion-tolerant group membership protocols is at a preliminary stage at present. In this paper, we have described the design, implementation, and performance evaluation of a new intrusion-tolerant group membership protocol.

The main contribution of this protocol is the introduction of a new group membership state called the suspended membership state. A group member is suspended as soon as it is suspected by a small number of group members. However, the protocol to remove a suspected group member from its group is initiated only after the suspicion has been confirmed by at least $1/3^{rd}$ of the group members. Once a member is suspended, it is not allowed to participate in any group communication activity. As a result, a member whose security is compromised gets only a very small amount of time (until it is suspected by a small number of group members) to launch any attacks in its group. It may be reinstated into the group if the initial suspicions turn out to be false at some later point in time. This concept of suspended membership state allows us to severely limit the damages a compromised group member can cause.

We have implemented this group membership protocol and measured its performance when a single member is compromised, and when two members are compromised. To evaluate this performance, we have also measured the performance of a three-phase group membership protocol that does not include a suspended membership state. The performance measurements show that the introduction of the new suspended membership state reduces the amount of time during which a compromised group member can launch attacks by as much as 100 milliseconds. We are currently building a complete group communication system called FITS integrating the group membership protocol proposed in this paper with the trustworthiness detector proposed in [21], an atomic broadcast protocol similar to the one proposed in [10], and an appropriate cryptographic package integrated to the system.

References

[1] Y. Amir, L.E. Moser, P.M. Melliar-Smith, D.A. Agrawal, and P. Ciarfella, The Totem single-ring ordering and membership protocol. *ACM Transactions on Computer Systems*, 13(4), pages 311-342, Nov 1995.

[2] Y. Amir and J. Stanton. The Spread Wide Area Group Communication System. Technical Report CNDS-98-4, Johns Hopkins University, 1998.

.

[3] K. Birman, Building Secure and Reliable Network Applications, Manning, 1996.

[4] M. Castro and B. Liskov. Practical Byzantine Fault Tolerance. In *Proceedings of the 3rd Symposium on Operating systems design and Implementation*, Feb 1999.

[5] T. Chandra, V. Hadzilacos, and S. Toueg. The weakest failure detector for solving consensus. In *Proceedings of the Eleventh ACM Symposium on Principles of Distributed Computing*, pages 147-158, Aug 1992.

[6] T. Chandra and S. Toueg. Unreliable failure detectors for asynchronous systems. In *Proceedings of the Tenth ACM Symposium on Principles of Distributed Computing*, pages 325-340, Aug 1991.

[7] G. V. Chockler, I. Keidar, and R.Vitenberg. Group Communication Specifications: A Comprehensive Study, In *ACM Computing Surveys*, 33(4), pages 1-43, Dec 2001.

[8] T. Courtney, et al,. Providing intrusion tolerance with ITUA. In *Proceedings of the 1st Workshop on Intrusion Tolerant Systems*, Washington D.C., June 2002.

[9] F. Cristian. Understanding fault-tolerant distributed systems. *Communications of ACM*, 34(2), pages 56-78, Feb 1991.

[10] F. Cristian, S. Mishra and G. Alvarez. High Performance Asynchronous Atomic Broadcast. Distributed Systems Engineering Journal, 4(2) June 1997.

[11] F. Cristian and C. Fetzer. The Timed Asynchronous Distributed System Model, IEEE Transactions on Parallel and Distributed Systems, 1999.

[12] M. Cukier, et al. Intrusion Tolerance Approaches in ITUA. In *Supplemental of the 2001 International Conference on Dependable Systems and Networks*, pages B64-B65, 2001.

[13] B. Dutertre and V. Crettaz. Intrusion-tolerant enclaves. In *The 2002 IEEE Symposium on Security and Privacy*, Oakland, CA, May 2002.

[14] B. Dutertre, H. Saidi, and V. Stavridou. Intrusion-tolerant group management in enclaves. In *International Conference on Dependable Systems and Networks (DSN'01)*, pages 203-212, Goteborg, Sweden, July 2001.

[15] K.P. Kihlstrom, L.E. Moser, and P. M. Melliar-Smith. The securing protocols for secure group communication. In *Proceedings of the IEEE 31st Hawaii International Conference on System Sciences*, Kona, Hawaii, Jan 1998.

[16] K.P. Kihlstorm, L.E. Moser and P.M. Melliar-Smith. The SecureRing Group Communication System, *ACM Transactions on Information and System Security 4(4*), page 371-406, Nov 2001.

[17] S.Mishra, Peterson, L. L., and Schlichting, R. L. Consul: A Communication Substrate for Fault-Tolerant Distributed Programs. *Distributed Systems Engineering Journal 1*,Dec 1993.

[18] S. Mishra. A middleware for constructing highly available, fault-tolerant, and attack tolerant services. In Pr*oceedings of the 17th ISCA International Conference on Computers and Applications*, San Francisco, CA, Apr 2002.

[19] S. Mishra and L.Wu. An evaluation of flow control in group communication. *IEEE/ACM Transaction on Networking*, 6(5), Oct 1998.

[20] S. Mishra, C.Fetzer and F.Cristian "*The Timewheel Group Communication System,*" *IEEE Transactions on Computers*, 51(8) , pages 883 -899, Aug 2002.

[21] S. Tanaraksiritavorn and S. Mishra. A Trustworthiness Detector for Intrusion-Tolerant Group Communication Systems. In Proceedings of the 2004 Hawaii International Conference on Computer Sciences, Honolulu, Hawaii, January 2004.

[22] P. Pandey, Reliable Delivery and Ordering Mechanisms for an Intrusion Tolerant Group Communication System, MS Thesis, University of Illinois at Urbana-Champaign.

[23] P. A. Porras, and P. G. Neumann. EMERALD: Event Monitoring Enabling Responses to Anomalous Live Disturbances. In *Proceedings of the 19th National Information Systems Security Conference*, Baltimore, USA, Oct 1997.

[24] R.Power. 2002 CSI/FBI computer crime and security survey. Technical report, Computer Security Institute Publication, Spring 2002.
Available at http://www.gocsi.com/forms/fbi/pdf.html.

[25] H.V. Ramasamy, P.Pandey, J. Lyons, M. Cukier, and W. Sanders. Quantifying the cost of providing intrusion tolerance in group communication systems. In *Proceedings of the 2002 IEEE International Conference on Dependable Systems and Networks*, June 2002.

[26] H.V. Ramasamy. Group Membership Protocol for an Intrusion Tolerant Group Communication System, MS thesis, University of Illinois at Urbana-Champaign, 2002.

[27] H.V.Ramasamy, M. Cukier, and W. Sanders. Formal Specification and Verification of a Group Membership Protocol for an Intrusion-Tolerant Group Communication System. In *Proceedings of the 2002 Pacific Rim International Symposium on Dependable Computing*, (PRDC-2002), pp 9-18, 2002.

[28] M. Hayden, and van R. Van Renesse. Optimizing Layered Communication Protocols. *Technical Report TR96-1613, Dept. of Computer Science, Cornell University, Ithaca, NY 14850, USA. Nov 1996.*

[29] M.K.Reiter. A Secure Group Membership Protocol. In *Proceedings of the IEEE Symposium on Research in Security and Privacy*, pages 176-189, 1994.

[30] M.K.Reiter. Secure Agreement Protocols: Reliable and Atomic Group Multicast in Rampart, In *Proceedings of the 2nd ACM conference on Computer and Communication Security*, pages 68-80, 1994.

[31] M.K.Reiter. The Rampart toolkit for building high-integrity services. In *Theory and Practice in Distributed Systems*, volume LNCS 938. Springer Verlag 1995.

[32] N.Subraveti, S. Tanaraksiritavorn and S. Mishra. Issues in building intrusion tolerant group membership protocols. In the 16th *ISCA International Conference on Parallel and Distributed Computing Systems (PDCS 2003)*, Reno, NV, August 2003.

[33] R.VanRenesse, K.P. Birman, and S. Maffeis. Horus: A flexible group communication system. *Communications of the ACM*, 39(4), Apr 1996.

APPENDIX: Pseudo Code for Pre-Membership Phase

Global:
State[i] = {Operational Member, Operational Sequencer, Suspend, Member_Wait, Sequencer_Wait, Consensus, Join Consensus, Agreement, Join Agreement, Commit, Join Commit}
ML: Member List; SL: Suspected List; SusL: Suspended List; FL: Faulty List;
RQ: A Queue of processes that are to be reinstated by the Leader;

Function Member (State[i], s)
Start
LSN, GSN, gid, m, m', s, MyRank: integers
Current_State: String
Event-While the message is a GMP message start membership protocol:

While (Recvd_Message (M, State[i]) is not a Join message) **do**

Event-Received message M:
if (Cryptocheck(M) fails) **then**
Inform_TD (m); //Inform TD to raise suspicion on member who sent the message
exit;
else // the Cryptographic verification of the message is successful

Current_State = State[i];

Event-Suspect (m') initiated by the Trustworthiness Detector event:
if (M = Send_Suspect_Message (m, m') & Current_State = Operational Member) **then**
Increment LSN;
Update the Suspected list (SL) to include m'
Start **Snd_Suspect (m')** timer;
Run Multicast (m, gid, Send_Suspect_Message, Current_State, LSN, GSN);
endif

Event-Received Suspect (m') from member m:
if (M = Recv_Suspect_Message (m, m') and Current_State = Operational Member) **then**
if (m' is one of the suspended members) **then**
Update SuspendedList[i] i.e. SusL[i] =m' by building the list of nodes that suspect m' (SusL[i]);
//Check if the total number of nodes exceeds the requirement to start GMP
if (SusL[i].totalnodes > $\lfloor (n-1)/3 \rfloor$) **then**
Update Faulty List to include m' i.e. SusL[i]
Start **Member_Wait** timer;
Current_State= Member_Wait;
Run Member_Wait (Current_State, s);
endif
// I am the next sequencer; so change state to Operational Sequencer
if (MyRank = s – 1) and m = s) **then**
Current_State = Operational Sequencer;
Run Leader (Current_State);
endif

```
    else
   // Build the database of members suspecting m'. Clear buffer after timer expiry
        Start Buffer_Clearance timer;
        Create/Update SuspendedList[i] i.e. SusL[i] =m' by building the list of nodes that suspect
        m' (SusL[i]);
   endif
endif
```

Event-Received Suspend (m') from leader s:

```
if (M = Recv_Suspend_Message (s, m') and Current_State = Operational Member) then
// If the current member is the suspended member
        Start Recv_Suspend (m') timer;
        if (m'=MyRank) then
                Current_State=Suspend;
                Run Suspend (Current_State, s);
                break;
        else if ( m'=s) then
                Inform_TD(s); //Inform TD to raise suspicion on sequencer
                break;
        else
                Update the Suspended List to include m'
endif
```

Event-Received Reinstate (m') from leader s:

```
if (M = Recv_Reinstate_Message (s, m') and Current_State = Operational Member) then
        Update RQ to remove m'
        if ( m' is part of SusL)
                Remove reinstated member from SusL
                Add the reinstated member to ML
        else
                Inform_TD(s); //Inform TD to raise suspicion on sequencer
                break;
        endif
endif
```

Event-Received Ordering (s, GSN, LSN) from leader s:

```
if (M=Update_Ordering (s, New_GSN, last_received_LSN) ) then
        if ( New_GSN=GSN+1) then // if GSN is the correct one received
                if (last_received_LSN=LSN-1) then
                        GSN=new_GSN; // if the LSN last sent
                else
                        Send message with LSN=last_received_LSN+1;
                endif
        else
                Request all messages from GSN+1 to New_GSN //Use Negative ACKs
        endif
endif
```

Event-Expiry of Snd_Suspect (m') or Member_Wait timer
if (Snd_Suspect(m') **or** Member_Wait timer expires) **then**
Inform_TD(s); //Inform Trustworthiness Detector to raise suspicion on sequencer
break;
endif

Event-Expiry of Recv_Suspend(m') timer
if (Recv_Suspend(m') timer expires) **then**
if (m' belongs to SusL and is not a part of ML or FL) **then**
Create/Update the RQ by including m'
endif

Event-Expiry of Buffer_Clearance timer

Clear Suspended List[i] i.e. SUSL[i] =m';
Inform_TD(s); //Inform Trustworthiness Detector to raise suspicion on sequencer
break;

endif // Successful Cryptocheck
end while-do
Run Join (Current_State, s); // If the multicast message is not a membership protocol message
end *Member*

//This explains how the protocol behaves when a correct member is accidentally suspected and enters Suspended Membership State.
Function *Suspend* (Current_State, s)
Start
Seq_rank: integer
While (Recvd_Message (M, State[i]) and Snd_Suspend (m') has not expired) **do**
Seq_rank=s;
if (Cryptocheck(M) fails) **then**
Inform_TD (m);
exit;
else if (M = Recv_Reinstate_Message (s, m') and Current_State = Suspend) **then**
// Let the current sequencer do the State transfer
Current_State=Operational Member;
Run State_Transfer(s, Current_State);
endif
end while-do
End *Suspend*

// This function explains the Member_Wait phase where the sequencer becomes the leader and no suspect messages are entertained
Function *Member_Wait* (Current_State, s)
Start
While (Recvd_Message (M, State[i]) and Member_Wait timer has not expired) **do**
// While only the faulty list is updated
if (M=Update_Member_Wait(m')
Update Faulty List to include m'
endif
if (M=Recv_Suggest_New_View(gid, ML, FL, SusL, SL, s) **then**

```
                Current_State=Consensus;
                Start Leader_Consensus timer;
                // Run the Standard Three Phase Commit Protocol
               Run ThreePC (MyRank, gid, ML, FL, SusL, SL, s, Current_State)
        endif

Event-Expiry of Leader_Consensus timer
 if ( Leader_Consensus expires) then
        Inform_TD(s); //Inform Trustworthiness Detector to raise suspicion on sequencer
        break;
endif

end while-do
End Member_Wait

Function Leader (State[i])
Start
Current_State=State[i];
MyRank, m, m', gid, t, s, LSN, GSN: integers
AL=List of Members that agree to suspect a particular member m';

Event-Reinstate Queue (RQ) is not empty:

if (RQ is not empty) then
    while (RQ[i] not in FL or ML for all i ) do
        Run Multicast(s, gid, Send_Reinstate_Message, Current_State, LSN, GSN);
        Remove RQ[i] from the SusL;
        Run State Transfer ();
     end while-do
endif

Event-Received Suspect (m') from member m:
if ( M = Recv_Suspect_Message (m, m') and Current_State = Operational Leader) then
    if (m' is one of the suspended members) then
        Update Suspended List by building the list of nodes that suspect m' (SusL[i]);
        //Check if the total number of nodes exceeds the requirement to start GMP
        if (SusL[i].total nodes > ⌊(n−1)/3⌋) then
           Update Faulty List to include m' i.e. SusL[i]
            Start Leader_Wait timer;
            Current_State= Leader_Wait;
            Run Leader_Wait (Current_State, s);
        endif
    else // A suspect message has arrived but wait for agreement
        Add m to the Agreement List where AL[i] =m';
        if (|AL|>= t) then
           Increment LSN, GSN;
           Run Multicast (s, gid, Send_Suspend_Message, Current_State, LSN, GSN);
           Current_State=Operational Member;
            Run Member (Current_State);
        endif
```

```
    endif
endif

Event-Received Suspect (m') from Trustworthiness Detector:
if (M = Send_Suspect_Message (s, m') & Current_State = Operational Leader) then
        Add s to the Agreement List where AL[i] =m';
        Update the suspected list (SL) to include m'
        if (|AL|>=t) then
            Increment LSN, GSN;
            Run Multicast (s, gid, Send_Suspend_Message, Current_State, LSN, GSN);
        else
            Increment LSN;
            Run Multicast(s, gid, Send_Suspect_Message, Current_State, LSN, GSN);
        endif
        Current_State=Operational Member;
        Run Member (Current_State);
endif

Event-Received Suspend (m') from leader s:

if (M = Recv_Suspend_Message (s, m') & Current_State = Operational Leader) then
        Start Recv_Suspend timer
        Update the Suspended List to include m'
endif

Event-Expiry of Leader_Wait timer
if ( Leader_Wait expires) then
             Run Multicast (s, gid, Suggest_New_View, Current_State, LSN, GSN);
endif

Function Leader_Wait (State [i], s)
Start
while (Recvd_Message (M, State[i]) and Leader_Wait timer has not expired) do
        if ( M = Recv_Suspect_Message (m, m') and Current_State = Leader_Wait) then
            if (m' is one of the suspended members and not in FL) then
        Update Suspended List by building the list of nodes that suspect m' (SusL[i]);
        //Check if the total number of nodes exceeds the requirement to start GMP
                if (SusL[i].total nodes > ⌊(n−1)/3⌋) then
                        Update Faulty List to include m' i.e. SusL[i]
                endif
              endif
        endif
end-while-do

while (SusL[i] is not empty) do
        Run Multicast(s, gid, Send_Reinstate_Message, Current_State,LSN, GSN);
        Update the RQ by removing m';
        Run State Transfer ();
        Current_State=Operational Member;
        Run Member (Current_State);
```

end-while do

Event-All Suspended Members are Faulty
if (SusL[i].total nodes > $\lfloor (n-1)/3 \rfloor$ for all i) **then**
// Send the Suggest New View Message
Run Multicast (gid, Suggest_New_View, Current_State, LSN, GSN);
Current_State=Consensus;
Run Leader-ThreePC (s, gid, ML, FL, SusL, SL, Current_State)
e**ndif**

Event-Expiry of Recv_Suspend(m') timer
if (Recv_Suspend(m') timer expires) **then**
Run Multicast(s, gid, Send_Reinstate_Message, Current_State, LSN, GSN);
Update the RQ by removing m';
Run State Transfer ();
Current_State=Operational Member;
Run Member (Current_State);
endif

End *Leader_Wait*