

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. This protocol maintains 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, which provides two unique features. First, it introduces a new membership state called a suspended membership state. This new 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.

1. Introduction

Group communication services have been widely 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, 4, 8, and 15]. 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. Almost all of existing group membership protocols (exceptions: [6, 11, 13]) have been designed to tolerate only crash/performance failures 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 (Flexible Intrusion Tolerant Group Communication System) that we are currently developing. It consists of a trustworthiness detector [9], a trustworthy group membership protocol (presented in this paper), and an atomic broadcast protocol. 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.

There are three important contributions that our group membership protocol provides. First, it introduces a new concept of *suspended group membership state*, which 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, we introduce 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 behaviors. Finally, a prototype of the proposed group membership protocol has been built, and the paper provides performance evaluation.

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 implementation and performance evaluation of the protocol. Finally, Section 6 concludes the paper.

2. Related Work

Related projects aiming to provide intrusion tolerance at the middleware level using a group communication service include Rampart [12, 13], ITUA [11], and SecureRing [6]. Rampart is the first group membership protocol aiming to tolerate malicious intrusions. It uses three-phase commit protocol and provides strong consistency guarantees. Rampart detects a fault from the external fault detector. When at least one-third of group members agree to remove a faulty member, three-phase commit protocol starts. Rampart also uses manager-based structure so each view of membership has a manager responsible for suggesting the new view. Rampart can handle one fault at a time. If multiple faults occur, it will be treated separately one by one.

SecureRing uses logical token ring as an underlying mechanism to disseminate data and maintain membership information. SecureRing also tolerates malicious faults and it claims to use fewer cost of digital signature. 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 adopts Rampart approach but adds capability to handle multiple faults in a single round. It uses an interface-Suspect in the protocol stack to report faults. 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 so it can suffer from the long idle time of applications during view installation if new faulty member is always detected when the protocol is in the last phase.

3. Motivation

Intrusion-tolerant group membership protocols must deal with two difficult issues. First, group membership protocols are time consuming and any service running 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 [14].

For a group membership protocol to start, more than $1/3^{\text{rd}}$ of the total number of group members should agree on the removal of the suspected group member. We observed in [9] that the detection time to suspect a compromised group member by $1/3^{\text{rd}}$ of the group members is significantly larger than the detection time by a first group member. During the period the first group member detects a suspect, till $1/3^{\text{rd}}$ of the group agree on the same suspected member, we see that there is a high potential that this “suspect” member can harm the system and consequently achieve inconsistencies in the group (despite its removal from the group). This issue can be addressed by making a slight modification to the mechanism of initialization of the group membership protocol.

We introduce the Pre-membership 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 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 $1/3^{\text{rd}}$ 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

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 i.e. the processes in the group do not need to synchronize clocks. We use timeouts as a means of handling the difficult problem of distinguishing between a communication failure and a failure of the group member. We use the trustworthiness detector [9] and an atomic broadcast protocol from FITS system.

The atomic broadcast is the rotating sequencer-based protocol similar to the Pinwheel atomic broadcast protocol in [5]. The group member that assigns global ordering on broadcast messages rotates on every broadcast of a control message. The sequence of the leader is fixed in advance based on its unique rank. The member with the 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. 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 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 given below:

- 1) **Member List (ML):** List of all correct members that are currently part of the group. For convenience let us call say 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. The system is in the stable and consistent state when each group member agrees on another member’s view of the group. We exercise a sequencer based Pre-Membership protocol until the

Three-phase Commit protocol is initiated. The output of the group membership protocol will be the view generated from the member list (ML) and Faulty List (FL).

The intrusion tolerant group membership protocol can be classified into 3 independent protocol modules which are closely coupled with the reliable broadcast mechanism. We classify these as follows:

Pre-Membership Suspension Protocol: This protocol is started when members of a group raise suspicion on one or more of the current group members. The other group members and the 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 which are consensus on new group, agreement on new view and protocol 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 require 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 3 phase commitment strategy. The members in the group go through the Pre-Join phase before the actual Join phase where the member actually joins the group.

4.1. Pre-Membership Suspension Protocol

We introduce a 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.

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 priority in existence so it is also safe to assume that each member is ranked by the protocol initiator. This protocol begins when a member in the group issues a suspect message based on the suspicion raised by its Trustworthiness Detector (TD) [9]. 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 issues 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.

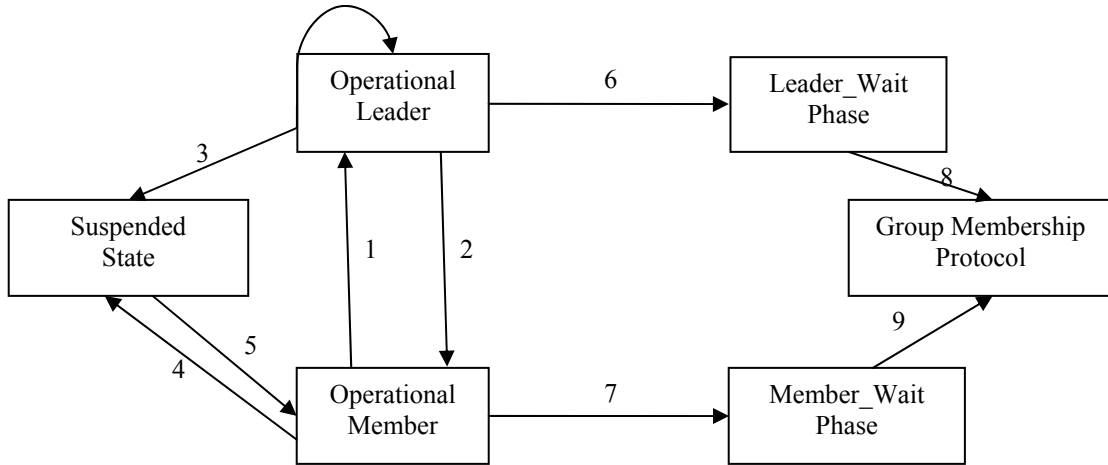


Figure 1: State Transition Diagram for the Pre-Membership Suspension Protocol

- 1) Operational Leader which has the highest rank in group receives multicast message.
- 2) Operational Leader multicasts an Update (GSN, LSN) or Suspect (m).
- 3) Operational Leader multicasts Suspend (m) and add 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 remove m from its SusL.
- 6) Leader moves to wait phase when gathering $|\text{Suspend}[i]| > n/3$ for some i
- 7) Member moves to wait phase when gathering $|\text{Suspended}[i]| > n/3$ for some i
- 8) Leader sends Suggest_New_View
- 9) Member receives Suggest_New_View

The sequencer updates the suspended list by checking if there is an entry of m' in $\text{SusL}[i]$ where $i = m'$. If there is none, add $\text{SusL}[m']$. Then, add the signed message, suspect (m, m') to $\text{SusL}[m']$ as a confirmation that there is a member m who suspects m' . The sequencer finally multicasts suspend (m, m') message i.e. *member m' is suspected by m* . Each time when sequencer moves a member m' to the suspended state, it starts a timer $\text{snd_suspend_wait}[m']$ corresponding to m' . If this timer expires, it means m' is in the suspended state for quite a period of time without further suspicions on him. So, we assume it is wrongly suspected and we should remove it from the suspended list so that it can participate in the 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 $\text{SusL}[m']$ lists and check if number of members who suspect m' (in $\text{SusL}[m']$) is more than $1/3^{\text{rd}} + 1$. Since we assume there can be at most one-third of group member that turns malicious, $1/3^{\text{rd}} + 1$ 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 $\text{SusL}[m']$ exceeds $1/3^{\text{rd}} + 1$. Similarly, the sequencer enters the Leader_Wait phase when it has received support for the suspension from more than $1/3^{\text{rd}}$ of the group members.

When this wait phase is reached, the member stops accepting application messages and the sequencer role does not rotate for simplicity of the following removal phase of faulty member. The suspected member is removed from SusL to FL if more than $1/3^{\text{rd}}$ of Suspend messages are received or it can be reinstated back to operational member in the event of Leader_Wait timer expiry. Each member maintains a queue of members that are to be reinstated so that when it later assumes the sequencer role, it has to reinstate all members in the reinstate queue. This phase ends when all members reach the Member-Wait phase and the Leader issues a Suggest_New_View message signaling the start of the three phase commit protocol.

4.2. 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 in Figure 2 shows 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 exercised except for suspicions on the Leader. Hence membership failures raised in the Pre-Membership

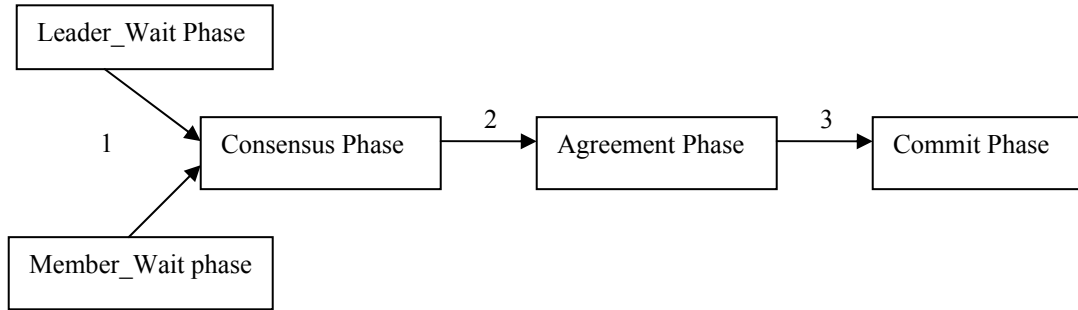


Figure 2: State Transition Diagram for a Group Membership Protocol

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)

Suspension protocol guides 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.

The first phase of the Three Phase Commit protocol is the Consensus phase. The sequencer broadcasts to all the members of the $|ML-FL-SusL|$ list the new view i.e. the updated member lists, suspended list and the suspected list. 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 will detect this and 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 suspects the Leader is 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 updates the membership status of the group accordingly. The Leader ensures final message stabilization is reached at the end of this protocol.

4.3. Group Join Protocol

This protocol provides a mechanism for new members to join securely. We assume the existence of a Public Key infrastructure. Since the join protocol is an independent protocol, it cannot run in the background while a member or a set of members are being suspected/removed. 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 a proof to the group, that he is correct and has authenticated with a member of the group, we cannot immediately include that member into the group because the group member that took part in the join protocol may have been suspected/deemed faulty. The alternative solution is to 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 at least one correct member has accepted the member to join the group. The state transition diagram for each group member is given in Figure 3. The dashed-joiner and interactions with operational member/leader 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 which 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 sufficient 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.

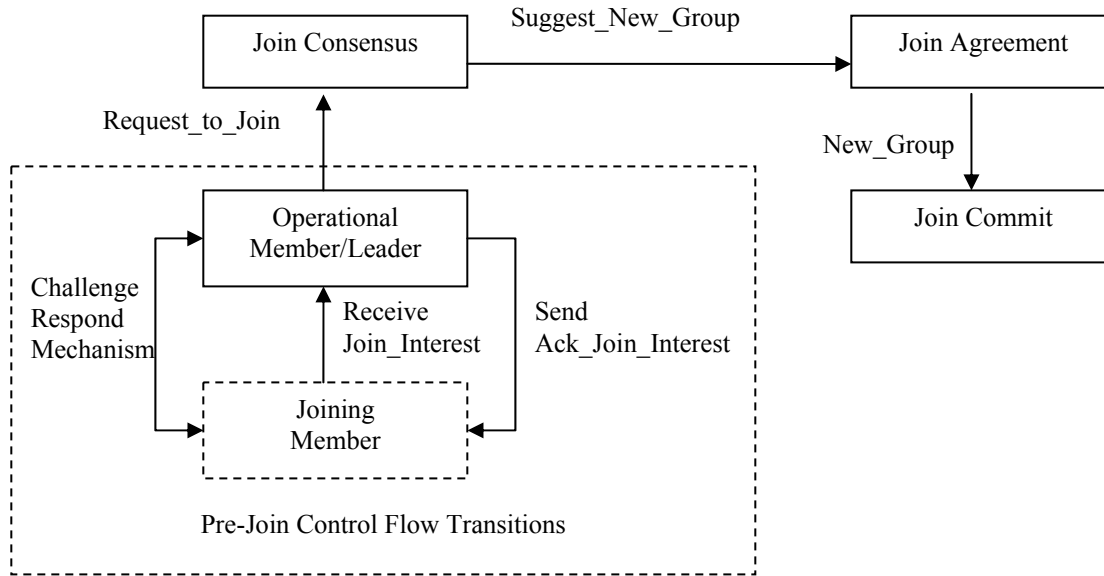


Figure 3: State Transition Diagram of a Group-Join protocol with the Control flow transitions in the Pre-join phase.

Also, let SK_j , SK_m , SK_l denote the private keys of the joiner, member and leader respectively.

The joiner contacts one or more members of the group with a `Join_Interest (gid, PKj)` message. 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, SKm (gid, PKj))`. This is collected from more than $1/3^{\text{rd}}$ 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, PKj)` message. The leader generates a challenge by creating a nonce which is encrypted with PK_j i.e. $PK_j(\text{nonce})$ 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 indicating that it would allow the member to join 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.

Join Protocol:

We propose a join protocol which is similar to the Three Phase Commit protocol. 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 $1/3^{\text{rd}}$ of the total group members have acknowledged the `Request_to_Join` message. On receipt of more than $1/3^{\text{rd}}$ the number of `Ack_Request_to_Join` messages the sequencer issues a `Suggest_New_Group` message $(gid, pid \text{ of the new joiner, signature of joiner, proof for New View})$ 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 $2/3^{\text{rd}}$ the number of members in the group. On receipt of more than $2/3^{\text{rd}}$ `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, including 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

Our group membership protocol is different from others in that we add a suspended state to balance the effect of false alarm and tardiness of confining malicious behaviors. Almost all group membership protocols make use of similar three-phase commit protocol. In this section, we want to measure the additional cost for suspended 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 underlying reliable multicast protocol using negative acknowledgement mechanism. We did not use complicated reliable mechanism or flow control because this paper concentrates solely on group membership protocol. The test simulates group size of four to twelve. All group members join a single multicast network. During group formations, 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 simulate failure of one and two members in the group. For one member failure, we trigger a member to send suspect (m, m') immediately after group was formed. No background traffic is generated in this test so that the number that we measured represents the protocol cost only. Based on our failure model, the group whose size is larger than 6 members can tolerate two failures. We trigger two member failures detected by different members in the group at the same time.

In order to illustrate the effect of our “suspended state”, we compare our protocol with the regular three-phase group membership protocol. In that protocol, we measure the protocol cost when it initiates its first phase which happens after at most one-third of group members agree on moving a faulty member. While our protocol cost starts when a member suspects a group member and sends suspect message. The difference between two graphs here shows that the introduction of the new suspended membership state can reduce the amount of time during which a compromised group member can launch attacks.

Figure 4 and 5 show the protocol cost in milliseconds for removing one and two group members with this implementation. The line marked “RG GMP” illustrates the time used for regular three-phase commit group membership protocol while the line marked “INT GMP” illustrates the time used for our intrusion group membership protocol. We can see that the introduction of suspended state does not create much extra cost. However, this is the amount of time that we can avoid the compromised group member to harm the system while the underlying applications can still be able to make progress.

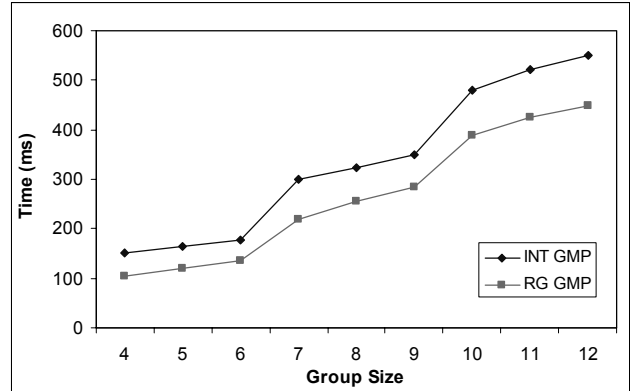


Figure 4: Protocol cost when one member in the group is faulty.

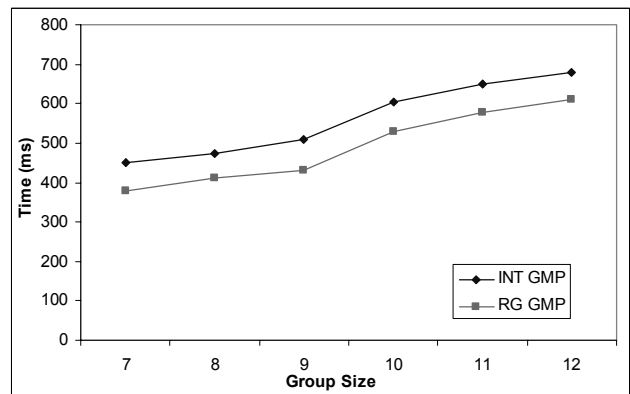


Figure 5: Protocol cost when two members in the group are faulty.

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 1/3rd of the group members. Once a member is suspended, it is not allowed to participate in the group communication activities. 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 *unsuspended* 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 50 milliseconds. We are currently building a complete group communication integrating the group membership protocol proposed in this paper with the trustworthiness detector proposed in [9], an atomic broadcast protocol similar to the one proposed in [5], and an appropriate cryptographic package.

7. Acknowledgements

The authors would like to thank suggestions from Damon McCoy in development of the Join Protocol.

8. 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] 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.
- [4] 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.
- [5] F.Cristian, S. Mishra and G. Alvarez. High Performance Asynchronous Atomic Broadcast. *Distributed Systems Engineering Journal*, 4(2) June 1997.
- [6] 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.
- [7] S. Mishra. A middleware for constructing highly available, fault-tolerant, and attack tolerant services. In *Proceedings of the 17th ISCA International Conference on Computers and Applications*, San Francisco, CA, Apr 2002.
- [8] S. Mishra, C.Fetzer and F.Cristian "The Timewheel Group Communication System," *IEEE Transactions on Computers*, 51(8) , pages 883 -899, Aug 2002.
- [9] 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.
- [10] 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.
- [11] H.V. Ramasamy. Group Membership Protocol for an Intrusion Tolerant Group Communication System, MS thesis, University of Illinois at Urbana-Champaign, 2002.
- [12] M.K.Reiter. A Secure Group Membership Protocol. In *Proceedings of the IEEE Symposium on Research in Security and Privacy*, pages 176-189, 1994.
- [13] M. Reiter. The Rampart toolkit for building high-integrity services. In *Theory and Practice in Distributed Systems*, volume LNCS 938. Springer Verlag 1995.
- [14] 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.
- [15] R. Van Renesse, K.P. Birman, and S. Maffeis, "Horus: A flexible group communication system," .In the *Communications of the ACM*, 39(4), April 1996