

A RING PROTOCOL FOR A CLUSTER BASED DISTRIBUTED SYSTEM

Turhan Tunali, Kayhan Erciyes and Zehra Soysert

tunali, erciyes, soysert@ube.ege.edu.tr

International Computing Institute
Ege University Bornova Izmir

ABSTRACT

A communication protocol is designed and partially implemented for a distributed system model. The protocol operates on hierarchical rings and forms the communication backbone of a cluster based distributed system model that is designed to provide various distributed system functions such as total event ordering, process group management, multicasting and fault tolerance. The two level hierarchy of the cluster based system model allows intracluster communication to be carried out independent of other clusters and hence provides parallelism in communication of cluster nodes. The intercluster communication which is done among cluster "representative"s allows the "leader" to uniquely dictate various decisions to the whole system. The semantics of fault management for the cluster based model is also given and the interrelation of this fault mode with various layers of the system architecture is given by state transition diagrams.

1. INTRODUCTION

In distributed system applications, the implementation of fault tolerant techniques such as data replication, increases the network traffic considerably, causing congestion and other complications. Various techniques have been developed to decrease the message complexity while providing high availability [1,2]. The key concepts used in these concepts are the group communication protocols and multicasting. Among the important contributions to the field are Isis [3], Horus [4], Totem [5] and Transis [6] projects. These are all event-driven asynchronous distributed systems. Although these systems can be successfully used in applications such as transaction processing, banking and stock market trading and replicated database systems, their real-time capabilities are limited. Only Totem has features for real-time, however, the extended virtual synchrony model used is still based on an asynchronous model.

On the other hand, synchronous models can yield better performance for real-time applications. For example, Delta-4 project uses synchronized clocks for the applications and logical clocks for multicast message ordering [7]. AAS is developed for air traffic control and uses a clock-driven and event-triggered approach [8]. MARS is developed for process control and uses a clock-driven and time-triggered approach [9]. However, for geographically distributed systems, the situation is further complicated in synchronous models due to the requirement of a globally synchronized clock. Moreover, it is well known that, these models require high communication bandwidth to yield acceptable performance in real-time applications.

In this work, a cluster based approach is applied to a synchronous system model with the expectation of reducing the network overheads while providing a suitable platform for real-time applications. This approach needs the development of a fault tolerance model that should be glued to the underlying communication protocol within the clusters and between the

clusters. This fault tolerance model should not be confused with the fault tolerance concept at the application layer. The latter is entirely independent of fault tolerance at the communication layer and any popular scheme such as replication can be used at that layer. At the communication level, the difficulty arises due to the crash failure model accepted. To maintain the intracluster and intercluster communication, a model that contains “ordinary cluster node”, “cluster representative” and “leader” concepts is developed. The communication protocol is designed to run these three processes at appropriate levels. In case of failures, the protocol rebuilds the system and starts functioning again.

The paper is organized as follows: In Section 2, the distributed system model is given and its interrelation with communication is stressed. In Section 3, the ring based protocol is explained with the algorithms and state diagrams. In Section 4, the implementation is explained and preliminary results of the ongoing implementation are discussed. Concluding remarks are given in Section 5.

2. DISTRIBUTED SYSTEM MODEL

The distributed system model is formed of clusters of computing nodes. Depending on the represented layer in the architecture, the word “node” will be used both for processors and processes. The nodes within a cluster are assumed to be physically located close to each other as in a local area network. Each cluster has a “representative” that communicates with the “leader”. The leader requests information from cluster representatives via the ring protocol that is operating at “higher level”, in other words, the ring only consists of the cluster representatives including the leader. Each representative, receiving this request, passes it to the nodes within their respective clusters. Within each cluster, the communication is also done by using the ring protocol, but this time, the protocol is used at a “lower level”, in other words, each ring consists of the nodes of a cluster including the cluster representative. The point to be noted here is that, exactly the same protocol is used in these two different levels of communication. Figure 1 illustrates the two levels of communication.

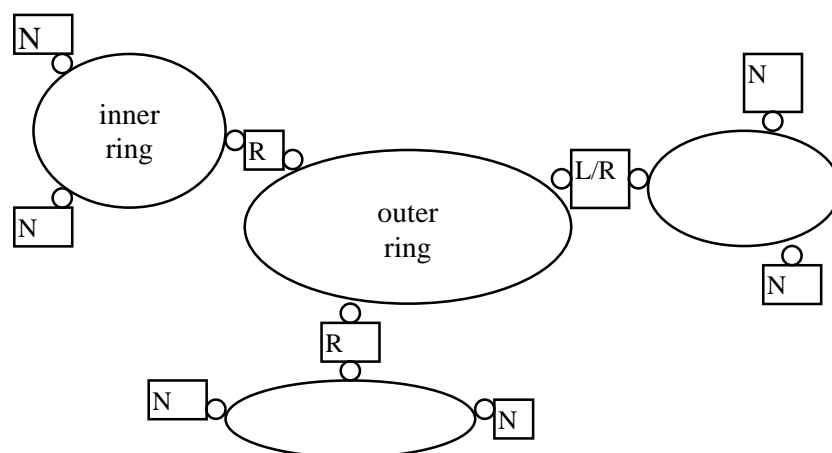


Figure 1: The two level communication hierarchy where L is the leader, R is a representative and N is an ordinary node

The system architecture considered in this study consists of various layers: The lowest level is the distributed clock synchronization. This is because of the assumption that the system model is synchronous. At this level, the cluster based model is implemented with nodes being the processors. The clock synchronization algorithm is based on the clock leader’s request of clock information from the nodes and upon receipt of the clock information from the system,

the clock leader decides on the new clock and dictates this to the whole system via the cluster representatives. This operation is repeated in every period and the clocks are synchronized.

Above the clock level, there sits the distributed scheduler whose operation is similar to that of clock level as far as the cluster model is concerned. The nodes are again the processors, the clusters are the same clusters, however, the representatives and leader of this level are not necessarily the same nodes as that of the clock level. In other words, the cluster topology is considered to be fixed for all layers of the architecture, however, the formation of the representatives and the leader of this level is entirely independent of that of the clock level.

Above the scheduler is the group membership layer. The cluster based model is again applied to this layer. The clusters are again defined by the original topology, but this time the nodes are the processes running on the processors of the respective clusters and belonging to a particular group. That is, for each process group, the cluster based model is applied once to determine the group leader, cluster representatives of the group and the nodes in clusters belonging to the group.

Above structure stresses that the cluster based model is repeatedly applied to the various levels of the distributed system architecture. In a sense, these architectural layers of clock synchronization, distributed scheduler and group management form one dimension of the overall model. The other dimension is the two level hierarchy of the cluster based model that is repeatedly applied to the architectural layers. It is important to see that the two dimensions are entirely independent of each other.

3. PROTOCOL FOR CLUSTER BASED MODEL

With the two independent dimensions pointed out in the previous section, we can now introduce the failure model associated with each dimension. We assume crash failure model for processes and omission failure model for the communication links. Thus, above the group layer an "active replication scheme" can easily be implemented via the cluster based model. It is not our purpose to elaborate the fault model of this dimension. It is well known that methods such as "primary/backup model" or "state machine replication" are widely implemented to solve fault tolerance problems of this dimension. On the other hand, the cluster based model needs a special semantics to solve its own fault tolerance problem. This section describes the protocol that imbeds fault tolerance semantics particularly developed for the cluster based model. It is an essential feature of the whole system.

The general operation of the protocol is as follows: The representative controls the circulation of a message frame within its cluster. This message frame will be called "inframe" and the intracluster ring will be called "inner ring". The inframe contains two main sections. The first section, called "global information", contains the information that is sent by the leader such as the global clock value or ordered event list. This section is "read only" for the nodes and it dictates the leader's decision about the previous cycle. The second section is filled by the nodes. It contains node information to be sent to the leader such as the local clock value or local event list with time stamps. The structure of inframe is shown in figure 2-a. Every node in the cluster knows the address of its predecessor, successor and representative in the cluster ring. If the predecessor of a node is the representative, then, that node also knows the address of the predecessor of the representative. All the nodes in all of the clusters know the address of the leader.

The leader controls the circulation of the message frame in the outer ring. This frame will be called the "outframe". As in inframe, the outframe also contains two main sections. The first

section contains the information that is sent by the leader such as the global clock value or ordered event list. This section is "read only" for the representatives. The second section is filled by the representatives. It contains the forwarded node information such as the local clock value or local event list with time stamps. The structure of outframe is shown in figure 2-b. Every representative knows the address of its predecessor and successor in the outer ring. If the predecessor (successor) of a representative is the leader, then the representative also knows the address of the predecessor (successor) of the leader.



Figure 2: Frame structures

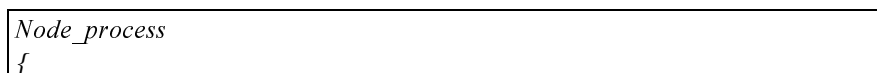
The protocol can be examined in three different modes of operations: Namely, *NODE*, *REPRESENTATIVE* and *LEADER*. When a *NODE* process is activated, it is furnished with the address of the cluster representative that it intends to join and the identification number of the cluster. It submits a "join request" to the representative with its address and id. Then it waits for new ring information that contains the addresses of a predecessor, successor and the leader together with some sequencing information. It waits the inframe to fill up the local information. As the inframe arrives, it reads the first section that contains the globally finalized information. It then fills up the appropriate fields to report its local information and starts waiting for the next inframe. If the inframe is late, it sends point to point "Are you alive?" messages to its predecessor, successor and representative. If either of its predecessor or successor is dead, it informs the representative and waits for a new information. The representative bypassing the dead node provides new predecessor or successor address to the node and the node acknowledging this, starts waiting for the inframe.

If the representative is dead, the node waits for the new representative information. As it receives new representative information, it acknowledges this and waits for inframe. The successor of the dead representative becomes the new representative. Having the address of the dead representative's predecessor, the new representative repairs the ring and informs this predecessor. The predecessor receiving new ring information goes into inframe wait state. Figure 3 shows the operation of the node mode. Figure 4 gives the state diagram of the node algorithm.

When a *REPRESENTATIVE* process is activated, it is furnished with a successor address on the cluster ring. It issues a new_rep inframe to let the cluster nodes know that it is the new representative. The node algorithm assures that every node in the cluster has its predecessor and successor address before the new representative process is activated. The new_rep inframe contains the following:

- the address of the new representative
- the address of the predecessor of the new representative
- The last received global information and its sequence number
- The ring address information to be filled in by all the nodes
- The local information field to be filled by the nodes

Upon receiving the new_rep inframe, the nodes update their records, fill in their addresses and acknowledge. After assuring the status of its nodes, the new representative sends a new_rep message to the leader. This message contains the following:



```

send_message(join_request, my_node, repr);
wait_for_new_ring_info;
cycle
  Wait_for_event;
  switch (event_type);
    INFRAME_RECEIVED :
      if (not_outer_ring_trouble)
        read_global_info;
        write_local_info;
        release_inframe;
        start_timer(inframe);
    INFRAME_TIMEOUT:
      send_message(are_you_alive, my_node, pred, successor, repr);
      start_timer(are_you_alive);
    NEIGHBOR_DEAD:
      send_message (dead_neighbor, dead_neighbor_addr, my_node,
repr);
    REP_DEAD:
      if (my_node==successor_of_dead_rep)
        send_message (I_am_your_new_successor, my_node,
pred_of_dead_rep)
        wakeup(Representative_process);
    NEW_LEADER_INFRAME_RECEIVED:
      update (new_leader_info);
      release_inframe;
    NEW_REP_INFRAME_RECEIVED:
      update (new_rep_info);
      release_inframe;
  endcycle
}

```

Figure 3: Node algorithm

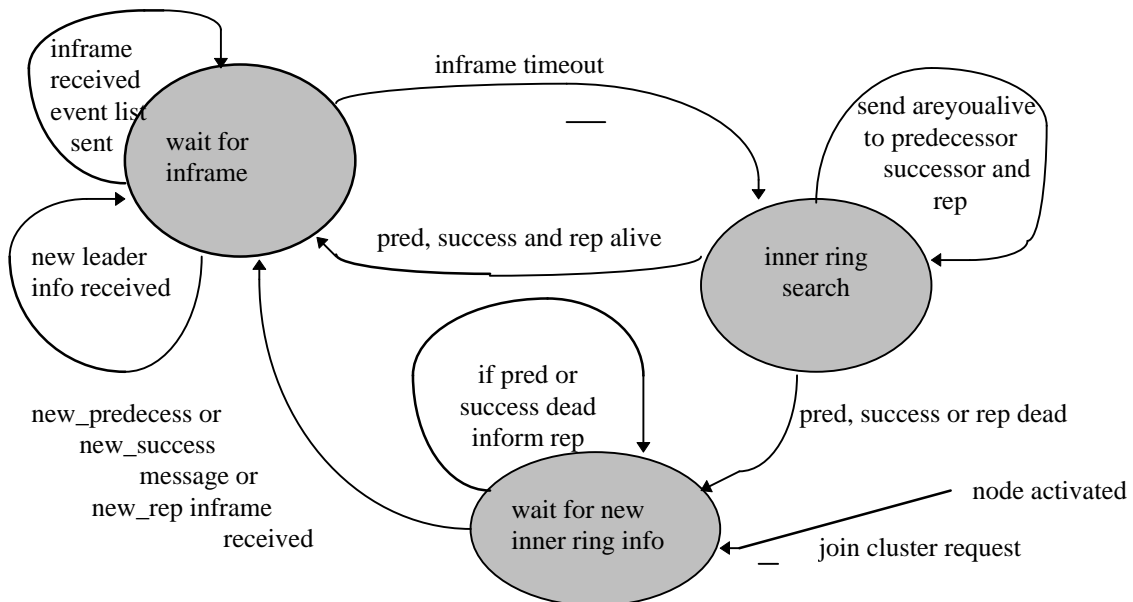


Figure 4: State diagram of the node protocol

- the address of the new representative

- cluster identification number
- the last received global information and its sequence number
- the local information gathered

The new representative starts waiting for outer ring information from the leader. The leader, forming the new outer ring, furnishes the new representative with a predecessor address, a successor address, the missing global information and the last global information that sorts the local information sent by the new representative. Now the new representative is ready for routine operations.

The representative issues an inframe and waits. When it receives the inframe it waits for the outframe. As the outframe is received, it reads the global information, writes the local information and forwards the outframe to its successor. It then issues the next inframe with the most recent global information to be forwarded. If the outframe arrives before the inframe, the representative holds the outframe for a limited time. If inframe is still late, it releases the outframe with empty cluster information and cluster trouble bit set.

If the inframe is late, it starts a "cluster search" timer and sends "I am alive" to all of its fellow cluster nodes. If everyone is alive and the timer goes off, it issues the last inframe again. If a node is dead, it discards the node from the ring and furnishes new addresses to the predecessor and successor of the dead node. Then it retransmits the last inframe. If there is a join request from a node, the representative modifies the ring, informs the related nodes and retransmits the last inframe.

If the outframe is late, the representative sends point to point "Are you alive?" messages to its outer ring predecessor, successor and the leader. If either of its predecessor or successor is dead, it informs the leader and waits for a new information. The leader bypassing the dead representative provides new predecessor or successor address to the representative and the representative acknowledging this, starts waiting for the outframe.

If the leader is dead, the representative waits for the new leader information. As it receives this information, it acknowledges this and waits for outframe. The successor of the dead leader becomes the new leader. Having the address of the dead leader's predecessor, the new leader repairs the outer ring and informs this predecessor. The predecessor receiving new ring information goes into outframe wait state. Figure 5 shows the operation of the representative mode. Since the representative acts similar to a leader in the inner ring and similar to a node in the outer ring its state diagram is the combination of a node and a leader.

When the leader process is activated, it issues a new leader outframe that contains the following:

- the address of the new leader
- the address of the predecessor of the new leader
- The last received global information and its sequence number
- The ring address information to be filled in by all the representatives
- The local information field to be filled by the representatives

Upon receiving the new_leader outframe, the representatives update their records, fill in their addresses and acknowledge. They also issue a new leader inframe to their clusters. After assuring the status of its representatives, the new leader is ready for routine operations. It sorts the collected local information and issues an outframe containing the global information.

<i>Representative_process</i>

```

{
  issue_inframe(new_rep,my_addr,my_pred,last_glob_info,
ring_addr_slots,local_info_slots);
  Wait_for_new_rep_ack;
  send_message(join_request,my_addr,my_cluster_id,last_global_info,last_local_
info,leader);
  wait_for_new_outer_ring_info;
  cycle
    Wait_for_event;
    switch (event_type);
      INFRAME_RECEIVED :
        read_local_info;
        if (outframe_in)
          update(cluster_info_slot);
          release_outframe;
          start_timer(outframe);
          issue_inframe(data,glob_info,local_info_slots);
          start_timer(inframe);
        start_timer(issue_inframe);
      ISSUE_INFRAME_TIMEOUT:
        issue_inframe (outer_ring_trouble);
      OUTFRAME_RECEIVED :
        read_global_info;
        if (inframe_in)
          update(cluster_info_slot);
          release_outframe;
          start_timer(outframe);
          issue_inframe(data,glob_info,local_info_slots);
          start_timer(inframe);
        start_timer(release_outframe);
      RELEASE_OUTFRAME_TIMEOUT:
        set (cluster_trouble);
        release_outframe;
        start_timer(outframe);
      INFRAME_TIMEOUT:
        send_message(I_am_alive,my_node,all_cluster_nodes);
        start_timer(are_you_alive);
        start_timer(cluster_search);
      CLUSTER_SEARCH_TIMEOUT:
        issue_inframe(data,glob_info,local_info_slots);
        start_timer(inframe);
      A_NODE_IS_DEAD:
        stop_timer(cluster_search);
        send_message(new_pred,dead_node_pred_addr,my_node,dead_no
de_successor);
        send_message
(new_success,dead_node_succ_addr,my_node,dead_node's_pred);
        issue_inframe(data,glob_info,local_info_slots);
        start_timer(inframe);
      OUTFRAME_TIMEOUT:
        send_message(are_you_alive,my_node,out_pred,out_successor,lea
der);
        start_timer(are_you_alive);
      OUT_NEIGHBOR_DEAD:
        send_message
(dead_neighbor,dead_neighbor_addr,my_node,leader);
      LEADER_DEAD:
        if(my_node==successor_of_dead_leader)

```

```

        send_message      (new_successor,my_addr,      my_node,
pred_of_dead_leader);
        wakeup(Leader_process);
        NEW_LEADER_OUTFRAME_RECEIVED:
            update (new_leader_info);
            release_outframe;
            start_timer (outframe);
        NEW_NODE_JOIN_REQUEST:
            send_message
(new_successor,new_node's_addr,my_node,my_pred);
            send_message
(rep's_pred,new_node's_addr,my_node,my_successor);
            issue_inframe(data,glob_info,local_info_slots);
            start_timer(inframe);
    endcycle
}

```

Figure 5: Representative algorithm

If the outframe is late, it starts an "outer ring search" timer and sends "I am alive" to all of its fellow cluster representatives. If everyone is alive and the timer goes off, it issues the last outframe again. If a representative is dead, it discards the representative from the outer ring and furnishes new addresses to the predecessor and successor of the dead representative. Then it retransmits the last outframe. If there is a new representative message, the leader modifies the ring, informs the related representatives and retransmits the last outframe. Figure 6 shows the leader algorithm. Figure 7 shows the state diagram for the leader process.

4. IMPLEMENTATION

The node, representative and leader algorithms are partially implemented on a network of workstations. The network consists of three clusters, each having three nodes with one of them having two of the nodes on the same machine. All nodes are UNIX workstations located in Ege University campus. The clusters are all on different 10 Mbps Ethernet segments. The outer ring is 100 Mbps FDDI. The implementation includes the synchronization of inframe and outframe. The synchronization rate achieved is an essential measure for the performance of the protocol when no crash occurs. Therefore, the maximal rate achieved at this level gives a good idea about the cycle period both for the inner ring and outer ring. Four different sizes are used for the messages: 1K, 2K, 4K and 8K. The outframe issue period is taken as 100 msec. Table 1 shows the average completion times of ring cycles based on 30 observations in milliseconds. As can be seen from the table, outframe issue period is well above the cluster periods to guarantee that inframes are ready waiting for the outframe. No cluster issue period is enforced for the moment. Representatives issue their inframes as they receive outframes.

5. CONCLUSION

A cluster-based distributed system model is considered for a ring-based hierarchical communication protocol developed together with its imbedded fault-tolerant cluster membership algorithm. The cluster concept is considered on a hardware basis such as the computers on the same Ethernet segment. The fault-tolerance algorithm developed allows the protocol to maintain communication in case of crash failures. For any resource management

```

Leader_process
{
issue_outframe(new_lead,my_add,my_pred,last_glob_info,ring_addr_slots,local_info_slots);
start_timer(outframe);
cycle
    Wait_for_event;
    switch (event_type);
        OUTFRAME_RECEIVED :
            read_cluster_slots;
            form_global_info;
            issue_outframe(data,glob_info,local_info_slots);
            start_timer(outframe);
        OUTFRAME_TIMEOUT:
            send_message(I_am_alive, my_node ,all_cluster_reps);
            start_timer(are_you_alive);
            start_timer (outer_ring_search);
        A_REPRESENTATIVE_IS_DEAD:
            stop_timer (outer_ring_search);
            send_message (new_pred,dead_rep
_pred_addr,my_node,dead_node_successor);
            send_message (new_success,dead_rep_succ_addr,my_node,dead_node's_pred);
            issue_outframe(data,glob_info,local_info_slots);
            start_timer(outframe);
        OUTER_RING_SEARCH_TIMEOUT:
            issue_outframe(data,glob_info,local_info_slots);
            start_timer(outframe);
        NEW_REPRESENTATIVE:
            send_message (new_successor,new_node's_addr,my_node,my_pred);
            send_message (rep's_pred,new_node's_addr,my_node,my_successor);
            issue_outframe(data,glob_info,local_info_slots);
            start_timer(outframe);
    endcycle
}

```

Figure 6: Leader algorithm

functionality, the protocol can be applied with tuned parameters such as the type of information carried in inframes and outframes and the length of the periods of frame issuing. This generic approach would then yield to a situation in a distributed network where n frames circulate, each of which corresponds to a discrete resource management function.

Message size	Inframe-Cluster1	Inframe-Cluster2	Inframe-Cluster3	Outframe-Outer ring
1K	5.254	13.696	3.003	19.456
2K	9.821	19.195	6.803	28.225
4K	14.179	25.132	10.872	43.421
8K	24.725	39.341	20.930	62.665

Table 1: Ring cycle completion times

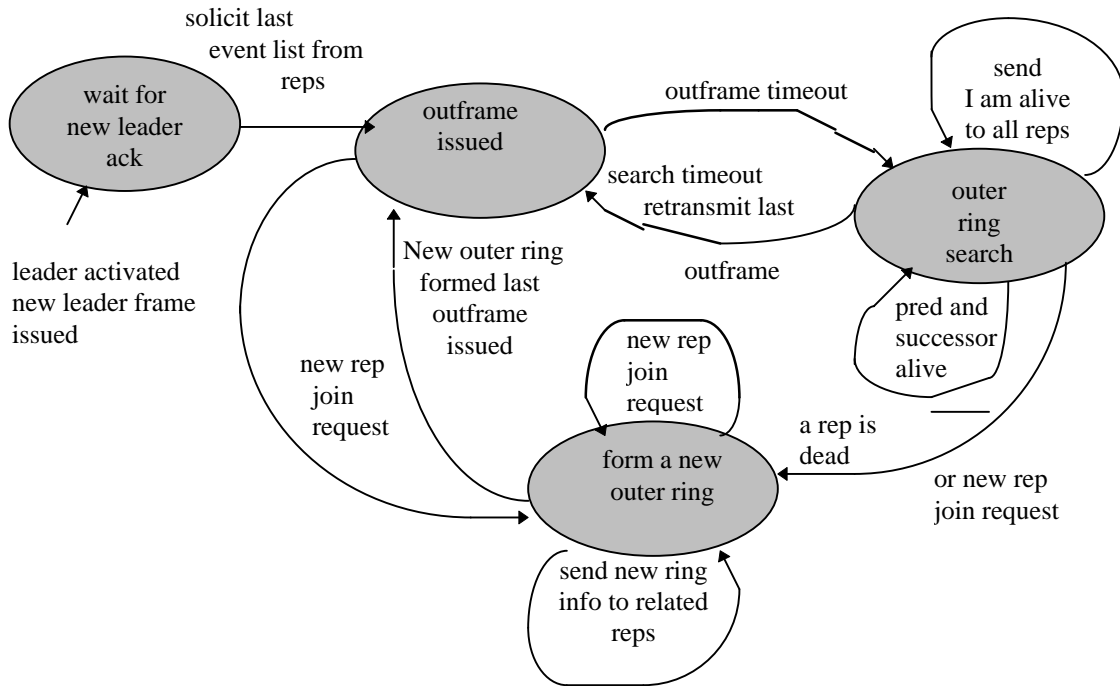


Figure 7: State diagram of the leader protocol

REFERENCES

- [1] Malki, D. (1994) "Multicast Communication for High Availability", Ph. D. Dissertation, Hebrew University in Jerusalem
- [2] Amir, Y. (1995) "Replication Using Group Communication Over a Partitioned Network", Ph. D. Dissertation, Hebrew University in Jerusalem
- [3] Birman, K.P. and Cooper, R. (1993) "The ISIS Project: Real experience with fault-tolerant programming system", Technical Report, Department of Computer Science, Cornell University.
- [4] Van Renesse, R., Birman, K.P. and Maffeis, S. (1996) "Horus: A Flexible Group Communication System", Communications of the ACM, Special Section on Group Communication, 39 (4).
- [5] Moser, L.E. et.al. (1996) "Totem: A Fault Tolerant Multicast Group Communication System, Communications of the ACM, Special Section on Group Communication, 39 (4).
- [6] Malki, D. and Dolev, D. (1996) "The Transis Approach to High Availability Cluster Communication", Communications of the ACM, Special Section on Group Communication, 39 (4).
- [7] Shivakant, M. and Schlichting, R. (1992) "Abstractions for Constructing Dependable Distributed Systems", Technical Report, TR92-19, Department of Computer Science, University of Arizona.
- [8] Cristian, F. (1993) "Understanding Fault-Tolerant Distributed Systems" Technical Report, Department of Computer Science, University of California, San Diego.
- [9] Mullender, S. (1993) "Distributed Systems", ACM Press, Second Edition.