

Kayhan Erciyeř

Ahmet řahan

erciyes, sahan @bornova.ege.edu.tr

International Computing Institute

Ege University Bornova, Izmir.

TURKEY

GENERIC APPLICATIONS ON A DISTRIBUTED REAL-TIME SYSTEM MODEL

ABSTRACT

The developed distributed, real-time system model is composed of hierarchical rings that have synchronous packet delivering characteristics. The model is designed as fault tolerant, so the system have a critical continuous packet deliverancy and reliability properties. The implemented model gives interface to many applications to be the upper layer of the new structure. In this way, Group Communication and Clock Synchronization generic applications were designed and implemented.

In scope of this study, The middle member of hierarchical topology, Representative, was implemented. Other parts of the system model, Node and Leader was explained in another document (Erciyes, Akay, 2001 [3]). In addition, the functional flow diagrams of the Group Communication and Clock Synchronization applications were generated. The complexity analysis of the developed system model and the performance results of it with its upper layer applications are also measured for several scenarios. With all of this work, it was aimed that the developed model could be used as base layer for applications that had real-time, distributed and fault-tolerant properties.

INTRODUCTION

The analysis and design phase of a distributed system model is very important. The critic questions before design are that how and where the distribution would be done. If the communication backbone is not reliable and is open to packet losses, the system model must be designed as fault-tolerant to prevent the high probability of packet loss. However, This will increase the network message traffic. The other question is that what the data is kept in the system members and also what the security of data with authorization permissions will happen. All of these questions should be carefully thought.

As it was said before, the implementation of fault-tolerant techniques increase message flow and network traffic. To eliminate these problems, some distributed system concepts were designed. The main principles of these projects are group communication and multicasting. The examples are Isis [1], and Totem [1]. All of

these is a kind of event-driven or in other words asynchronous systems. However, the real-time attributes of these models are limited, only Totem [2] has some real-time properties.

Synchronous systems which include periodic functionalities, can yield better performance for real-time applications. Although virtual clock mechanisms and ordered message processing principles are applied on these systems, for geographically distributed systems, the providance of a common clock synchronization is a complicated situation. In addition to this, these systems require high communication bandwidth to reach an acceptable performance for their real-time characteristics.

The aim of this study is to revise, develop and implement a synchronous distributed real-time system model that was designed previously (Tunali, Erciyes, Soysert, 1998 [3]). And then, designing and developing two generic applications: Clock Synchronization and Group Communication. All this work is done to show that the generated model could be base layer for many real-time applications.

The developed distributed system model is composed of hierarchical clusters of processing nodes which are connected by some communication medium. Tokens are delivered on hierarchical rings as periodically and time critically. Thus, a synchronous flow is implemented. The hierarchical ring nature reduces the network overheads of the system with its fault-tolerance concept implementation. In case of failures, the system rebuilds itself and starts again.

After the implementation of the base system model, the generic applications are designed. The first is clock synchronization and the second is group communication. both of them has different tokens on the hierarchical rings. Each token collects application messages and moves them to the coordinator of the ring. The main coordinator of applications is the header of the hierarchical topology, Leader.

The paper is organized as the following: In section 2, the base distributed real-time system model is explained. In section 3, the generic applications will be given. In section 4, the complexity analysis of the real-time distributed model is discussed. In section 5, the implemantation information by discussing regular work and member failure scenarios will be given and in section 6, the conclusion remarks will be defined.

DISTRIBUTED REAL-TIME SYSTEM MODEL

The distributed real-time system model is formed of hierarchical clusters of node members. The node represents a processor or a process. The bottom level clusters look like a local area network which consists of node members. Each cluster has a coordinator which is called as Representative. In three-level design, two kinds of Representative are available: at low level, Sub-Representative, and the coordinator of rings that have Sub-Representatives members is Super-Representative. At the highest level, the leader is coordinator. It can be called as header of the hierarchy. It manages all the system by issuing a packet on ring where Super-Representatives are members.

Leader issues a token periodically on its ring, that is called as outframe. When Representatives receive the token, they also issue a token called as inframe or midframe to collect data from their nodes. The token structure consists of a control data section, Representative or Node member data slots and global read only information slot. The global area is mainly written by Leader to inform all members of the system for any application related information. Each Representative or Node member writes their application related data into their slots. The coordinators receive those data when they receive tokens on ring again. The hierarchical view of the system is displayed in Figure-1.

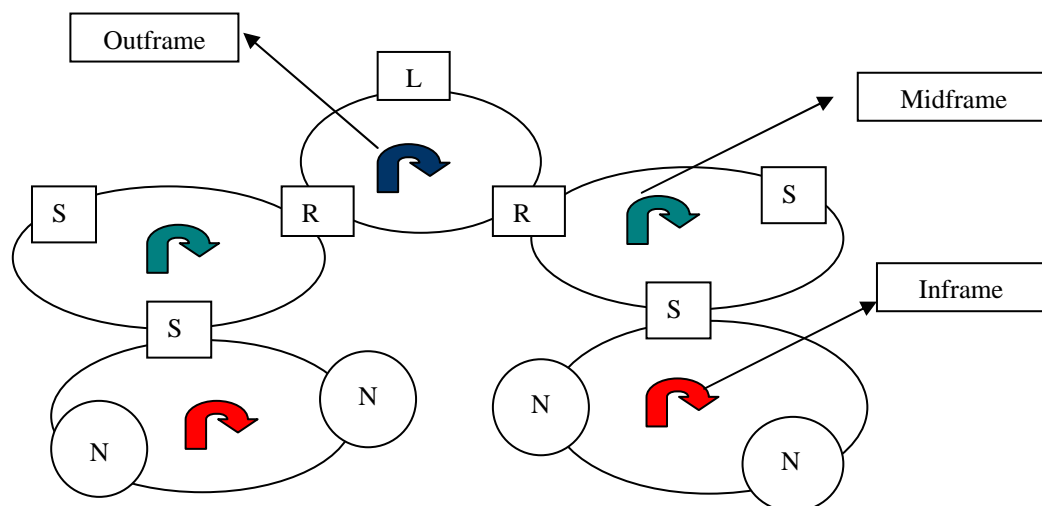


Figure-1: Three-layer Hierarchical Model

(L:Leader, R: Super-Representative, S: Sub-Representative, N:Node)

Above the system model, two more applications are located. The first is Clock Synchronization and the second is Group Communication. Clock Synchronization application gathers initially real clock data of node members and the later circulations it collects virtual clock values. The generation and setting of virtual clock values are the responsibility of Leader and Representatives. Group communication application waits for asynchronous group packet request. If it comes at any time, it reaches to Leader. Then, the Leader manages the access of group packet deliver. The details of applications will be given in the next section.

The join and remove operations for system membership can be occurred at any time. There is no restriction. However, each this operation breaks the regular work of the related ring. After join or remove operation is successfully completed by the coordinator of the ring, the regular cycle starts again.

There are some crash-recovery properties of the base layer, distributed real-time system model. First of all, if a node member crashes, after a critical-time pass, packet timeouts occur. A neighbor of the dead node informs the Representative or Representative detects the problem itself. At that point, the Representative removes the dead node from its ring list and starts the packet delivery in regular way after a ring check. If Leader or a Representative crashes, the mechanism works differently. In a ring, each inner ring successor of the coordinator has a responsibility of restarting a new coordinator in case of coordinator crashes. When a coordinator crashes, its successor detects it after a critical timeout pass. Then it informs the predecessor of the dead coordinator for the new coordinator and starts the new coordinator. The new coordinator gathers ring members' information and issues a ring check message. After receiving this message on ring again, the regular work starts again. All these explanations are also valid for Leader crash.

Another important subject is addressing structures too. Each node member keeps basically four addresses: Its predecessor, successor, representative and leader addresses. In three-level design, node keeps its sub and super representative addresses not leader's. In addition the inner ring successor neighbor of ring coordinator also has extra address which is the predecessor of the coordinator. Especially, it is required when the coordinator crashes and to inform the predecessor node of the dead coordinator about the new starting coordinator address. Thus, when predecessor receives a ring frame, it can send it to correct place. The same definitions are also valid for Representatives. When a ring regeneration cycle occurs, all members are informed for changing coordinator addresses. Because, when being a Leader or Super-

Representative chance comes to a node or a Sub-Representative, they must own the address of the upper layer coordinator for join operations. Otherwise, the system will halt or crash.

PROTOCOLS FOR GENERIC APPLICATIONS

After developed thereal-time distributed system model, two generic applications are located above it. Although these applications use different tokens on the hierarchical rings, they are closely related with each other. The general architecture is given in Figure-2.

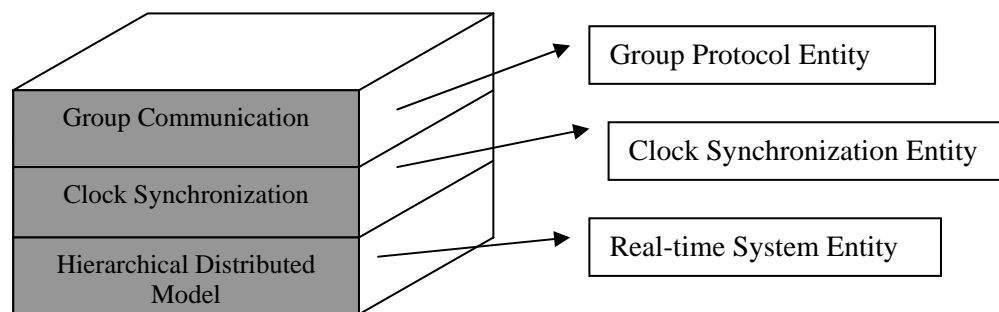


Figure-2: The Architecture of Distributed System components.

Each entity has tokens that circulates on the hierarchical distributed system model rings so the applications are on the higher layers. The reason of being Group Communication protocol at the highest layer is that Clock Synchronization application generates a virtual clock value. Group Communication actually requires this value, because it needs a metric for comparing and sorting the group member message timestamps. Therefore, Clock Synchronization is under the Group Communication entity.

When it comes to work of clock synchronization, the flow diagram is shown in Figure-3. The mechanism was prepared based on two-level hierarchy of the system.

When the first collect message comes from Leader, Nodes initially send its real clock values. At later steps, they send their active virtual clock values. Actually, three types of messages can reach to the nodes: "COLLECT", "SET" or "NONE". If "None" comes, it means that increment your virtual clock value as frame period parameter. If "Collect" comes, it writes its clock data into its slot in inframe. If "Set"

comes, it changes its clock value by adding a constant communication delay to the new coming value.

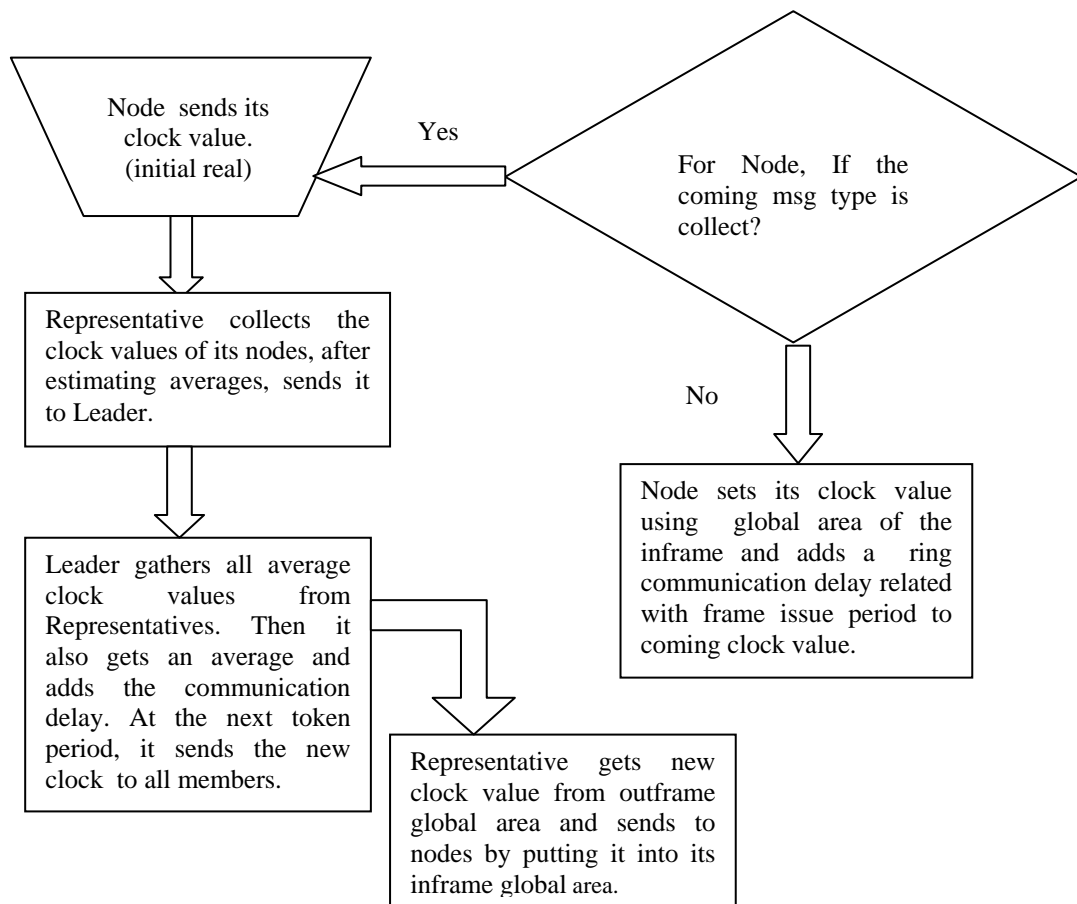


Figure-3: Clock Synchronization Work-Flow Diagram.

Representative does two important jobs during clock synchronization. First, it has an interface role between Nodes and Leader. It sends messages from nodes to Leader or vice versa. The second, if a collect message comes from Leader, after collecting node clock values, it estimates an arithmetic average using those values and sends it to Leader.

After Leader issues a collect message and receives all clock values, it also estimates an average value. It also adds a communication delay like Nodes. The communication delay is related with frame issue and arrival time values. At the next period, it sends this new clock value to all system by a new "SET" message. This work cycle can work forever. However, if some kind of member crashes or joins occurs, the synchronization will be interrupted until the regular work starts again.

Group Communication work flow approach has some common properties with Clock Synchronization. However, the mechanism is a bit more complex. Like Clock Synchronization, it has also a special message. This is inserted into the data section of hierarchical model generic message. Depending on the message type, group communication is directed by Node, Representatives and Leader. The general work flow based on three-level hierarchy model is given in Figure-4.

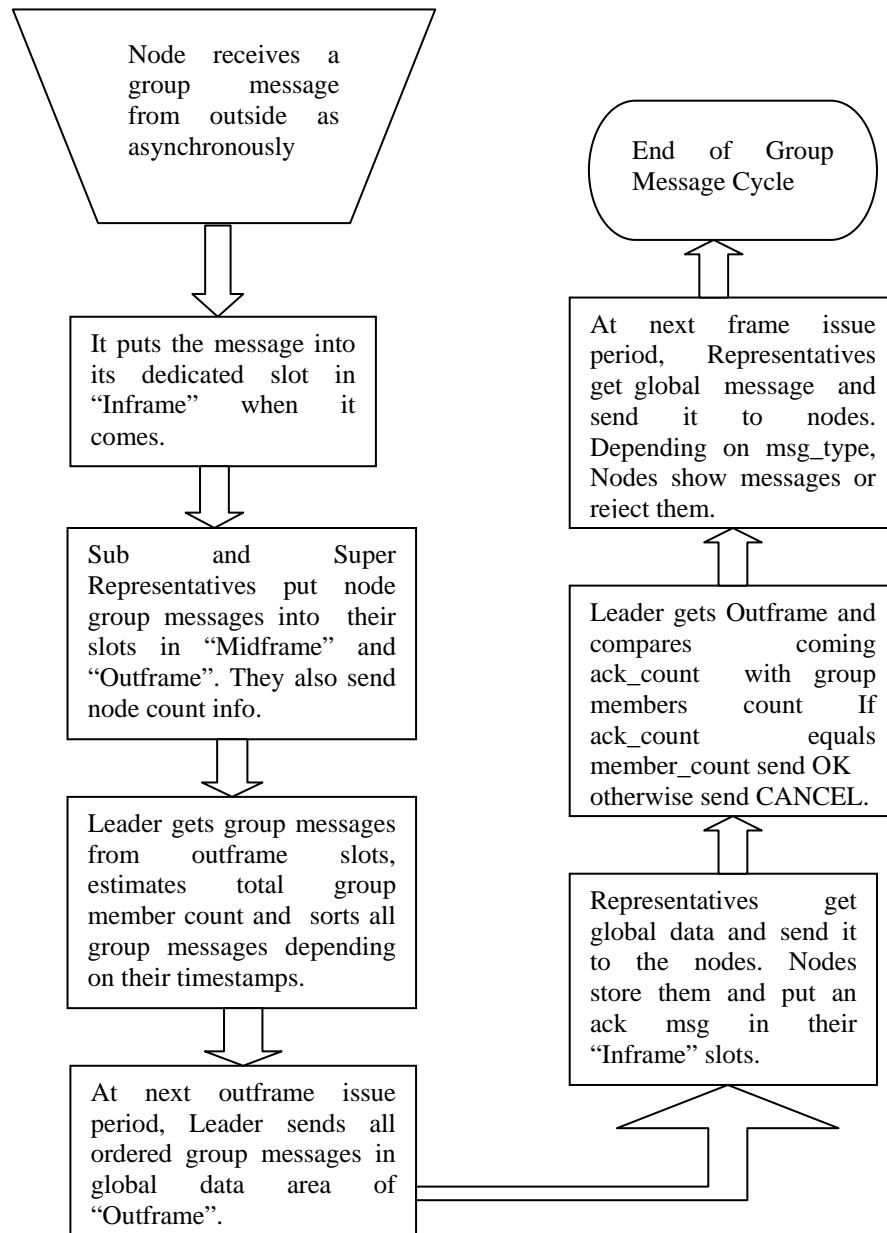


Figure-4: Group Communication Work Flow Diagram.

The general properties of Group Communication mechanism is as follows:

Leader is the central of the group. It represents all hierarchical structure with its nodes. It manages all group message flow in hierarchy with sorting, group view and atomicity properties.

Representatives have two important jobs during message broadcasting. They forward messages from nodes to Leader or vice versa. The second is Representatives estimate group member count depending on the active node counts on rings.. This information is used by Leader to sort messages and check atomicity.

Asynchronous message sending to nodes is done externally by using a console program. The event-driven received message is sent to Leader via Representatives at next "Inframe" receipt. If group messages will be sent directly to Leader, the same kind of console program can be used, only destination process type will be changed.

The sender of the group message does not receive its own message at the end of the cycle. It is implemented by inserting a source address field into group message structure. These messages are located into the data area of the distributed real-time system frames ("Inframe", "Midframe", "Outframe"). For the message deliverance and acceptance of it, the global read only data area of these frames is used.

At the same time, more than one message can be sent to a group. However, because of the group message sending overhead, the required time for concurrent message sends must not be greater than the frame issue periods. Otherwise, frame timeouts occur and the system loses its integrity. Because of this, as the node count increases, the maximum concurrent message send count will decrease.

Group Message Cycle is defined as atomic. It means that, a group message is received by none of members or all of members. There is no other option. The atomicity control is managed by Leader. It firstly gets group member count and then compares the coming ack message count from nodes. If they don't match, it represents that atomicity was broken.

In three-level design of the distributed system, Total group message transmit period is equal to $7T$ in worst-case situation. T represents frame period. The first T is waiting period if the inframe is just released before group message comes. The next T defines the access time of message to Leader. The third T defines sorting and waiting

next frame issue period in Leader. The fourth T represents the receiving of group message packets by nodes. The fifth T represents collection of ack messages toward Leader from Nodes. The sixth T defines the atomicity checking and waiting for the next outframe issue period. The seventh and last T defines the access of the group communication session result. Depending on the atomicity checking, it is OK or CANCEL.

If any crash or problem occurs during message send session, it will cause a failure for group communication, because the control is done using group message packet frames. In designed model, there is no timer defined for group communication. However, if it is required, a timer mechanism must be defined for senders of the group messages. Actually, it has only an information functionality for the time-being. If a fault-tolerant mechanism is established, some kind of buffering mechanisms will be needed.

During sorting of messages, a selection sort algorithmic data structure is used. The parameter for sorting is the timestamp of messages. This timestamp must be the value coming from Clock Synchronization entity. The time complexity of selection sort is $O(n^2)$ in terms of node counts.

ANALYSIS OF THE SYSTEM MODEL

In this section, the complexity analysis results of distributed system model will be given. As algorithmic approach, some details were taken from thesis of Erciyes, Akay, 2001. In addition, in regular work and in crash scenarios how does the system react, will be explained in complexity arguments.

The complexity analysis was done for three-level hierarchy of the model. Two complexity measures were thought. The first is Time Complexity, how much time passes during packet circulation and crash recoveries. The second is Message Complexity, how many messages are delivered at any regular work time and during crash recoveries. The complexity scenarios were selected depending on the worst-case situations.

In regular work, the passed time of packet circulation on rings in terms of node, represent counts equals $O(n^3)$. During this Big-Oh representation, the time from one member to another was thought as constant. For n members and three levels in hierarchy, this value is measured. During estimation, it was assumed all member

counts in clusters was equalized to a constant value. For message complexity of the system, it equals $O(n)$. Actually, it equals the number of total sub or super represent counts, cluster count. The reason is that at any time on all rings, at least one packet circulates. In worst-case, this packet counts equals the total number of clusters in system.

In case of crashes, depending on the represent and node counts, many scenarios can be generated. At this time, standard member count is assumed that (node counts and represent counts) greater than 2. The results in Table-1 and Table-2 are estimated from the base system model algorithms. T defines constant packet send period (it is assumed that for all packets, sending times are equal.), s defines the message. Like t , for all kind of messages, only s letter is used.

As it is seen from the table, the member counts affect the performance of the system. As the level of hierarchy is getting higher, the performance falls down.

Scenarios	Time Complexity Measures (for $n > 2$)	
Node Crash	$T(n) = (n+3) t$	$O(n)$
Sub-Represent Crash	$T(n) = (4n+7) t$	$O(n)$
Super-Represent Crash	$T(n) = (2n^2+6n+7) t$	$O(n^2)$
Leader Crash	$T(n) = (2n^3+4n^2+4n+5) t$	$O(n^3)$

Table-1: Time Complexity of the System for Crash Scenarios.

Scenarios	Message Complexity Measures (for $n > 2$)	
Node Crash	$T(n) = (5n+2) s$	$O(n)$
Sub-Represent Crash	$T(n) = (11n+9) s$	$O(n)$
Super-Represent Crash	$T(n) = (6n^2+23n+9) s$	$O(n^2)$
Leader Crash	$T(n) = (4n^3+12n^2+12n+1) s$	$O(n^3)$

Table-2: Message Complexity of the System for Crash Scenarios.

The assumption of equalization of the node and represent counts will be valid for great values of node and represent counts.

When it comes to the evaluation of the results, especially, when Super-Represent or Leader crash, all members in the system is affected. Because, the packet circulations under the crashed member and circulation with its neighbors stop after a while and they cause frame timeouts to occur.

The same situation is valid for message counts. As the number of member counts increase, and the member level of crash rises, the performance of the system for recovery falls down. It means the the number of messages increases. Actually a part of system is responsible for the recovery. However, low level members send each other alive messages because of the uncoming ring packets and frame timeouts.

IMPLEMENTATION

After finishing the design phase of the system and its generic applications, the implementation phase was done. The middle members, Representatives are coded in scope of this study. In addition, Clock Synchronization and Group Communication were implemented. Other parts are coded by another person (Erciyes, Akay, 2001).

Nodes, Represents and Leader are implemented as POSIX threads on Sun OS v5.7 UNIX operating system. All of them run on Alpha workstations. On each workstation, for simulative reasons, more than one node and represent can start. If required, they can be run on different workstations.

The addressing mechanisms are based on IPv4. Each member of the system has an IP address plus a counter value which states the member position in that workstation. During implementation, the following member position intervals are used:

1-216: Node members

217-234: Sub-Representatives

235-237: Super-Representatives

0: for Leader.

At once time, maximum 238 members can run. The member management (join, remove...) is done by console and main program threads. The communication between member threads are done by threads: sender, receiver. They use FIFO queues and UDP/IP sockets. The synchronization in FIFO queues is provided by special kind of

counter semaphores. The timing mechanisms were implemented by a special timer thread module. In addition to these, a logger thread was developed for debug checking and displaying result purposes.

The programming language tool was GNU C/C++ compiler. For network environments, the Alpha stations in Ege University were used. The backbone was 100Mbps FDDI.

After base system model was developed, the generic applications were coded. First Clock Synchronization, and then Group Communication module were implemented.

After all implementations completed, some measurements and estimations were done. In this measurements, packet circulation values and crash recovery results are given for some scenarios as follows:

Scenario	Inframe Circulation	MidFrame Circulation	Outframe Circulation
Sup#:3, Sub#:5, Node#:9	114ms	35ms	27ms
Sup#:3, Sub#:3, Node#:6	74ms	15ms	26ms

Table-3: Packet Circulations

During measurements, the packet issue period was set as 170ms. Actually, this value is only used by Leader. Represents don't use a timer period. Instead, As they get the upper level packet, they issue their low level packets. In Table-4, for two scenarios, the crash result values are given:

As it is seen from tables again, there is a reverse relationship with number of members and the performance of the system. After these estimations, some measurements were also done for generic applications. In Table-5, the clock drift between real clock and virtual estimated clock is given. During this estimation, the frame period was 200ms and the level count was 2 (There is only one Represent).

Scenario	Sup#:3, Sub#:5, Node#: 9	Sup#:3,Sub#:3, Node#:6
Node Crash	106ms	57ms
Sub-Represent Crash	228ms	106ms
Super-Represent Crash	232ms	167ms
Leader Crash	96ms	84ms

Table-4: Crash Recovery Results

Inframe Ring (1 member) in ms.	Inframe Ring (3 members) in ms.	Inframe Ring (4 members) in ms
Node-1: 0.503	Node-1: 0.502 Node-2: 0.503 Node-3: 0.503	Node-1: 0.502 Node-2: 0.502 Node-3: 0.502 Node-4: 0.503

Tablo-5: Clock Synchronization Drift Values After 30 seconds period.

For Group Communication application event, the following measurements were estimated. Depending on the number of concurrent messages, the total time cycle of group message is given. Frame period was set to 170ms, and level count was three.

	Node# 18	Node # 27	Node # 54	Node # 81	Node # 90	Node # 135
Concurrent Msg Cnt = 1	1095	1132	1165	1176	1241	1266
Concurrent Msg Cnt = 5	1113	1115	1198	1160	1190	1228
Concurrent Msg Cnt = 10	1130	1157	1154	1214	1296	1380

Table-6: Group Communication Estimations in ms.

As it is seen from Table-6, the values actually occur nearly at the same value points. The reason is that the mechanism depends on the frame issue period. It is mainly affected by the arrival of message to node. The number of member counts has smaller effect on communication throughput.

CONCLUSION

With this study, a synchronous, fault-tolerant, hierarchical ring clustered, distributed and real-time system model was designed and implemented. Moreover, two generic applications, Clock Synchronization and Group Communication, are developed and run on this base system model.

The model has three main modules: Leader, Representatives and Nodes. Nodes are real members of system. Representatives do interface management and Leader has a master role. Packet circulation continuously work. At each period, a frame is issued. When Node members receive the ring frame, they send their data and get the global data prepared by Leader.

Clock Synchronization keeps system in stable state. All members own a virtual clock value. This value can provide a reliable group communication.

Group Communication application works asynchronously. It is activated when a group message request comes to one or more nodes. The main properties of the module are totally ordering and atomicity. The system implements atomicity with group view approach during message sending. Group Members are nodes and the group manager is Leader of the base system. Representatives send their member counts to Leader during group messaging session and also they have a data bridge role between Nodes and Leader.

The result after all work is that the model could be successfully used by many real-time critical applications. The next step must be using the model in a real-life project.

REFERENCES

- [1] Moser, L.E., 1996, "Totem: A Fault-Tolerant Multicast Group Communication Systems"
- [2] Agraval, Deborah, "Totem: A Reliable Ordered Delivery Protocol for Inter-connected Local-Area Networks".
- [3] Tunali, Erciyes, Soysert, 1998, "A Hierarchical Fault-Tolerant Ring Protocol For a Distributed Real-Time System".