On Simulating Two Permission-based Distributed Mutual Exclusion Algorithms For Wireless Sensor Networks

Sherenaz Al-Haj Baddar, The Department of Computer Science, Kent State University

Abstract—Sensor networks are challenging non conventional distributed systems. They need robust Distributed Mutual Exclusion primitives to be implemented in order to operate properly. This paper describes two simulators that were developed to examine and evaluate two Permission-based Distributed Mutual Exclusion Algorithms (DMX): Ricart-Agrawala's and Maekawa's algorithms. Experiments were conducted to evaluate the two simulated algorithms in terms of the number of messages exchanged and the responsiveness time. The results showed that Ricart-Agrawala's algorithm induces larger messaging overhead. They also showed that using broadcasts for implementing resource requests in Ricart-Agrawala's simulator and unicasting in order to achieve the same goal in Maekawa's simulator causes Ricart-Agrawala's responsiveness delay to be relatively smaller which is counter-intuitive.

Index Terms—Wireless sensor networks, Distributed Mutual Exclusion, Simulation.

I. INTRODUCTION

A distributed system is a collection of independent processes that communicate with each other in order to accomplish certain tasks. Sensor networks are appealing distributed systems that have been growing gaining attention recently.

Sensors are small, low-cost, battery-dependent, processing entities that interact with their surrounding environment, and are capable of communicating with one another. These sensors are usually wireless, due to the fact that they are deployed in environments where human intervention is either undesired or unsafe. These distinctive distributed systems have been thoroughly investigated recently in order to address their relevant design and implementation problems. It is worth noting that the solutions applied for conventional distributed systems might, and usually do, perform poorly when applied to sensor networks due to their special characteristics. Such characteristics include, but are not limited to, sensors’ low power supply, the unreliable communication media (Radio) they rely on, and their tiny sizes which result in limited processing and storage capabilities. Therefore, sensor networks need highly custom-tailored, problem-specific solutions that take their dispositions into consideration.

In a distributed system, it is not uncommon to have several processes sharing one or more resource(s) in an exclusive fashion. This brings up one of the most fundamental distributed systems problems: the problem of Distributed Mutual Exclusion (DMX).

If whenever a process holding a given resource, prevents all other processes from accessing the same resource until it voluntarily releases it, then the resource sharing is said to be mutually exclusive. The problem of Mutual Exclusion (ME) has been addressed in centralized systems. A typical solution for this problem must satisfy two vital properties[4]:

1. Safety (exclusion): If one process is using the shared resource, no other process can do so, until the process holding that resource releases it.
2. Liveness (Freedom from Deadlocks): if a process requests a resource, then its request will be satisfied within a finite amount of time.

Several solutions for the ME problem in centralized systems were developed, including but not limited to: Semaphores, Monitors, and Conditional Variables. All of these solutions benefit from the presence of shared memory among the centralized system’s processes competing for resources. The DMX problem is a more complicated variant of the ME problem. The absence of shared memory in typical distributed environments necessitates the development of DMX solutions that take advantage of the communication mechanism provided in these systems, namely, message passing.

The solutions developed for the DMX problem fall into one of two general categories: Permission-based DMX algorithms, and Token-based DMX algorithms. Permission-based DMX algorithms operate by forcing each process that wants to access the resource to gain a permission message from some or all of the processes within the distributed system. This guarantees that exactly one process at most will be holding the resource at a time. On the other hand, Token-based algorithms implement DMX by providing one token message in the system and allowing the process that solely holds the token to access the shared resource. Again, this ensures that at most
one process will be accessing the designated resource at a time.

Two widely known DMX algorithms are Ricart-Agrawala’s (RA) DMX algorithm [7], and Maekawa’s (MK) DMX algorithm [5]. In RA’s DMX algorithm, the process that wants to hold the resource requests permission from all OTHER processes in the system. If a process P receives a request to hold a shared resource from process Q, while it doesn’t want to hold the resource itself or has a lower priority request, then P sends a permission message to Q. Otherwise, P queues Q’s request, and replies after it releases the shared resource. When all of the processes reply to process Q it gains access to the shared resource.

In Maekawa’s DMX algorithm, each process communicates with a set of adjacent processes called the “quorum”. When process P wants to access a shared resource it has to gain permission from all of its quorum members. A quorum member Q, has one permission to give, and once given, any incoming request for using the shared resource has to be blocked. After releasing the shared resource, resource holder P sends release messages to all of its quorum members, allowing each one of them to grant permission to the next blocked processes.

In order to evaluate the efficiency of a given DMX algorithm, two performance metrics can be considered, the first of which is the number of messages exchanged per access to the shared resource. The second is responsiveness which measures the amount of time that elapses between requesting the shared resource and gaining access to it. These metrics can be affected by the system load, that is, the number of processes allowed to request the shared resource at the same time. The two commonly considered loads are Low and High loads. Low load on one extreme represents the case when one process is allowed to request the resource at a time. High Load represents the other extreme in which all processes are allowed to request the shared resource in the same time.

This paper describes two simulators designed to examine and evaluate Ricart-Agrawala and Maekawa’s DMX algorithms, under Low Load. The simulated distributed system consists of a collection of autonomous sensors that compose a single hop wireless network running TinyOS [9]. The simulation was developed under TOSSIM [12] environment using nesC [8] programming language.

Section 2 in this paper highlights some related work that has been carried out so far. Section 3 describes the structure of the developed simulators. The setup of the experiments conducted to evaluate the relative performance of the simulated algorithms along with the discussion of the results obtained is demonstrated in Section 4. Section 5 highlights future work and Section 6 concludes the research conducted.

II. LITERATURE OVERVIEW

Walter, Welch, and Vaidya, proposed a Token-based DMX, Reverse Link (RL) DMX, for mobile ad-hoc networks [1]. The proposed algorithm limits the number of processes with which a process needs to communicate to a limited number of neighbors making it well-suited for ad-hoc networks. In order to evaluate the performance of their algorithm, they designed a simulation environment in which they compared RL with Raymond’s Token-based DMX [3]. The metrics they used were: the number of messages and the average waiting time per resource acquisition. The simulation results showed that the proposed algorithm had a better average waiting time at the expense of inducing higher messaging overhead.

Baldoni, Virgillito and Petrassi proposed a Token-based DMX algorithm for mobile ad-Hoc networks [11]. The proposed algorithm assumes that the processes form a dynamic logical ring. A simulator was developed in order to test the algorithm using the GloMoSim simulation environment. GloMoSim [13] is a library-based simulator for wireless and mobile network, developed in the UCLA Parallel Computing Laboratory. The simulation results they obtained showed that, in a mobile ad-hoc network, an effective reduction in the number of hops per application message can be achieved, using a specific policy to build on-the-fly the logical ring.

Vedantham, Zhuang, and Sivakumar [10], designed a centralized, greedy approach to address the DMX problem in terms of benefit function. They also proposed an distributed realization of the greedy approach.

III. THE STRUCTURE OF THE DEVELOPED SIMULATORS

This section describes the structure of the two simulation modules developed: the Ricart-Agrawala’s simulation module, and Maekawa’s simulation module.

A. Ricart-Agrawala’s algorithm simulator.

When a process initiates Ricart-Agrawala’s module, it initializes its local environment, sets its status to IDLE, and initializes requests and replies queues to empty values. When the starting component of RA’s DMX module is executed, it initiates a kickoff delay timer if the running process happened to be the next one to request access to the shared resource. The basic delay interval used throughout the simulations is 1024 (one TinyOS second). When the kickoff delay timer is fired, the designated process posts a request to access the shared resource. When the control is transformed to the posted request task, the requesting process increments its local clock and broadcasts a request for locking the resource timestamped with its current local clock. If the broadcast is successful, the requesting process sets its local status to “requesting” and listens to incoming messages. For the remaining processes in the environment, when a process receives a request to access a resource, it checks its local states. If the process is neither holding the resource nor having a pending request for the same resource with a higher priority, it sends a reply message to the requesting process. Otherwise, it queues the reply message in the pending queue for later consideration. Process’s P’s request to access a resource is considered to be of higher priority than process Q’s request, if P’s request has a smaller timestamp than Q’s
request, or if P and Q’s requests have the same timestamp but P’s process identifier is smaller than Q’s. When the requesting process receives a reply message from all of the other processes, it gains an exclusive access to the designated resource. To control the amount of time a given process spends holding the resource, a delay timer is started when the process gains permission to use the resource. When this timer is fired, the process releases the resource, and as it does so, it checks its pending queue in order to send reply messages to the processes that had requests with lower priorities.

As a typical process tries to send a request or a reply message, the sending operation might fail. In such a situation, the sending process queues the failed message in the proper queue and starts a particular resending delay timer. When the designated resend delay timer is fired, the process tries to send the message on top of the associated queue. This counts for and avoids messages losses which are quite common in real sensor network environments.

B. Maekawa’s algorithm simulator.

The main idea behind Maekawa’s DMX algorithm is to reduce the necessary message complexity via limiting the number of processes with which a certain process needs to communicate. To achieve this goal, Maekawa’s algorithm introduces the concept of quorums. The quorum of process P, denoted by Rp, is a subset of the system processes from which P requests permission to exclusively access the shared resource and to which it sends release messages upon releasing the resource being held, such that:

1. \( P \in R_p \)
2. \( \forall P \in Q : R_p \cap R_q \neq \emptyset \)

Maekawa[5], has shown that the minimum quorum size is \( \text{ceil}(\sqrt{N}) \). However, the quorum formation itself needs to be resolved. Several algorithms for building quorums have been proposed. One of which is Billiard quorum formation algorithm [2]. Using Billiard quorums, the processes form a logical grid. This simplifies calculating a process’s quorum. Let a process P be located in the logical cell \( R(i,j) \), then P’s quorum members are the processes located along \( R(i,j) \)’s north, south, east, and west neighboring lines. Namely: the processes along these lines constitute the quorum members:

- \( R(i-1,j+1) = R(i,j)-(q-1)/2 \)
- \( R(i+1,j+1) = R(i,j)+(q+1)/2 \)
- \( R(i+1,j-1) = R(i,j)+(q-1)/2 \)
- \( R(i-1,j-1) = R(i,j)-(q+1)/2 \)

Where q denotes the size of the quorum.

The simulation module applies the algorithm in [2] in order to establish the quorum. When a typical process boots, it executes the init() component in which the process’s local environment gets initialized, including system and quorum sizes. Then, the start component is executed, where the process’s quorums get defined utilizing process’s TOS_LOCAL_ADDRESS in order to figure out the proper quorum members. Afterwards, the kickoff delay timer is started by the process designated to start requesting the shared resource. When the kickoff delay timer is fired, the process requesting the resource, denoted by P, posts a task requesting the permission to do so. When the task gets executed, P starts communicating with quorum members one at a time. P starts sending a request to the first not-contacted-yet quorum member, denoted by Q. Each quorum member has one permission to be given out. Once given to some process, Q won’t be able to give it to any other process until the process to which it gave the permission sends a release message. Meanwhile, any requests that arrive at Q, after it has given its permission to another process, are queued locally. When Q receives the release message from the process currently holding the resource, it will send reply to the requesting process located at the top request in its pending replies queue.

Here again, reliable messaging is as an issue. Upon sending any message: request, send or reply, the send command’s return value is tested. If the message fails to send, then it gets queued properly and a delay timer for that particular message type is started. When that timer is fired, the message on top of the designated queue is resent, the same steps are repeated until all pending messages have been sent successfully.

IV. EXPERIMENTATIONS AND RESULTS

Two performance metrics were considered in order to evaluate the relative performance of the two simulated algorithms. The metrics investigated were:

1. The number of messages: via counting the number of messages exchanged per resource request/release.
2. Responsiveness: the amount of time that elapses between the instant of time at which process requests a shared resource, and the instant at which it gains access to the designated resource.

Two sets of experiments were carried out one for each metric. The simulators were instrumented with the parameters necessary for measuring the designated metrics. The experiments setup and results are described in details in this section.

A. The general setup.

The number of processes within the system (the i.e. system size) considered varied between 4 and 40 processes. In each experiment, the simulator ran for 30000 TinyOS clock ticks. This time interval was found to be long enough to capture the algorithms’ behavior. The contention level considered was low level, allowing one process to request the critical section at a time. Delay timers were used to control the frequency and duration of accessing shared resources. They were also used for controlling the intervals between successive attempts to
Resend failed messages. Each of these timers was fired after 1024 clock ticks from the instant of time at which it was started.

For Ricart-Agrawala DMX simulation, a fully connected network was assumed. For Maekawa’s DMX simulator, processes were forming a logical grid in order to apply Billiard algorithm for quorum construction. Table 1 shows the quorum size used together with the corresponding system sizes.

<table>
<thead>
<tr>
<th>Number of nodes</th>
<th>Quorum size</th>
</tr>
</thead>
<tbody>
<tr>
<td>4-6</td>
<td>3</td>
</tr>
<tr>
<td>7-12</td>
<td>5</td>
</tr>
<tr>
<td>13-28</td>
<td>7</td>
</tr>
<tr>
<td>29-40</td>
<td>9</td>
</tr>
</tbody>
</table>

**B. Measuring the number of messages.**

Ricart-Agrawala’s simulator was instrumented with counters that kept track of the number of requests sent before resource permission was granted. The broadcast that was issued for gaining access to the resource in a system of \( n \) processes was counted as sending \( n \) different messages, one to each destination. The number of replies received upon sending the request was added to the total message count. Upon releasing the resource, the number of reply messages that were queued in the pending replies queue was added and the total of these counts was used to represent the number of messages exchanged per one resource request/release.

In Maekawa’s simulation, the number of request messages issued to quorum members was counted. Then, the number of reply messages received from the quorum members giving permission to the requesting process was added. Upon releasing the critical section, the number of release messages sent to quorum members was added to the total that represents the number of messages exchanged per one resource request/release. The above scenarios were repeated varying the number of processes over the range 4-40. Figure 1 shows the total message counts obtained for each conducted experiment.

![Messages Per CS Entry](image)

Figure 1. Maekawa’s messaging overhead is less that Ricart-Agrawala’s.

Figure 1 shows that the number of messages exchanged in Maekawa’s algorithm exceeds the corresponding number generated by Ricart-Agrawala’s algorithm for system sizes <7. This observation agrees with the mathematical result obtained by solving \( 6 \sqrt{N} > 2(N-1) \), where \( N \) is a positive Natural number.

For larger system sizes, the total number of messages reported by Ricart-Agrawala’s simulator exceeds the number of messages exchanged by Maekawa’s algorithm significantly. However, more experiments are necessary in order to validate the growing gap in number of messages exchanged between the two algorithms as system size increases.

Another observation, is that Maekawa’s DMX simulator exhibits less sensitivity towards the increase of system size. The number of messages exchanged increases at the system sizes value at which the quorum size increases. However, it remains the same for the different system sizes that have the same quorum size. In Ricart-Agrawala’s simulator, on the other hand, each system size results in a different number of messages being exchanged.

**C. Measuring responsiveness.**

Responsiveness is defined to be the amount of time that elapses between the instant of time at which a process starts requesting a shared resource and the instant of time at which this request is satisfied. The SimpleTime interface has been used in the two simulation modules to help record the amount of time that elapsed between two events.

In order to measure the responsiveness of Ricart-Agrawala’s algorithm, the SimpleTime interface command getLow32() was used to record the current system time and save it in a local variable, timebefore. When the requesting process gains access to the resource, it records the current system time using getLow32(), and stores the time recorded in another local variable, timeafter. Then, the difference between the two recorded times is computed and deposited in a third local variable, respond.

Measuring responsiveness of Maekawa’s simulator proceeded in a similar fashion. Upon requesting a resource local system time is recorded. Subsequently, when permission is granted, the time is recorded and the time elapsed is calculated and deposited.

When comparing Ricart-Agrawala and Maekawa’s algorithms in terms of responsiveness one expects that Ricart-Agrawala’s responsiveness is going to be slower than Maekawa’s. More specifically, the time it takes a requesting process to gain access to the resource using Ricart-Agrawala’s is expected to be larger than the time it takes a process under Maekawa’s algorithm to accomplish a similar task. Naturally, the two situations compared have the same system size. Quite surprisingly, the results collected showed the exact opposite tendency. Figure 2 plots the relation between the system size and the responsiveness measured in TinsyOS clock ticks.
The simulation development environment justifies the unexpected outcomes. In Ricart-Agrawala’s simulator the requesting process issues a single broadcast message. It then waits for replies from the rest of the system processes. However, in Maekawa’s simulator, the requesting node unicasts a separate request to each quorum member in a sequential manner. It begins contacting the first quorum member, then, issues a new message to contact the second quorum member, and so on, until all quorum members are contacted. It is quite apparent, that Maekawa’s unicast-based communication induces delays that are much larger than Ricart-Agrawala’s broadcast-based communication delays for the same system size.

To overcome this anomaly different solutions can be deployed, a simple solution is to start recording the system time after issuing all requests necessary to access the resource. However, this solution alters the exact semantics of responsiveness. Another solution is to let the requesting process issue a broadcast instead of the unicast. Subsequently, the receivers that don’t belong to the requesting process’s quorum will ignore the received request. However, this solution eradicates Maekawa’s DMX solution advantage of having a smaller messaging overhead. A third alternative is to develop and use a multicasting communication primitive. This multicast primitive must be efficiently equivalent to the broadcast communication primitive, where the designated request gets flooded only to a subset of the system processes.

Another observation that can be noticed in Figure 2, is that Maekawa’s simulator responsiveness is much more sensitive to the increase in system size. The increase of responsiveness values for Ricart-Agrawala’s outcomes tends to be much slower than what is exhibited by Maekawa’s. Further experimentation with larger data sets is needed to validate and properly justify this observation. Moreover, applying one of the three solutions for measuring responsiveness properly might have a different impact on the simulators’ responsiveness sensitivity towards system size.

V. FUTURE WORK

This section introduces some possible elaborations on the simulators developed.

The simulators capabilities of tolerating message loss needs to be improved in order to be able to capture higher contention levels. Such simulators will be capable of locating and managing the deadlocks that may show up in Maekawa’s DMX.

Another interesting issue that is worth investigating is the possible correlation between the sparseness of quorums and the like hood of running into deadlocks in Maekawa’s DMX algorithm. In some experiments that were conducted on Maekawa’s simulator, several processes were picked at random and were allowed to contend for the critical section concurrently. A general trend that was observed is that selecting processes with more sparse quorums (fewer intersections within quorums), resulted in delaying running into deadlocks. Well designed experiments that deploy several quorum formation strategies can be used to investigate this potential possibility further.

The developed simulators can be further extended to consider multi-hop sensor networks, as well as more generalized DMX problems like the dining or drinking philosophers [6].

VI. THE CONCLUSION

Wireless sensor networks are challenging non conventional, distributed systems that have been gaining growing attention recently. These networks need highly custom-tailored, problem-specific solutions that take their distinctive characteristics into account.

This paper explores a fundamental requirement that is necessary for the robust and consistent operation of any distributed system, including sensor networks; namely, Distributed Mutual Exclusion (DMX). The absence of shared memory in typical sensor networks necessitates developing distributed solutions that take advantage of the sensor’s basic communication mechanism, that is, message passing.

Several solutions that try to solve DMX have been proposed so far. These solutions fall into one of two general categories: Permission-based DMX algorithms, and Token-based DMX algorithms. In Permission-based DMX algorithms, a process gains access to the shared resource when it receives a permission message from some or all of the system processes. In a Token-based DMX algorithm, a process accesses the shared resource when it obtains the only token in the system. When compared to Token-based DMX solutions, Permission-based DMX algorithms induce higher messaging overhead. However, they produce relatively lower synchronization delay and exhibit less sensitivity to system load, in general.

This paper discusses two simulations developed for examining and contrasting two Permission-based DMX algorithms: Ricart-Agrawala [7] and Maekawa’s [5] DMX algorithms. The simulated sensors run TinyOS [9]. The simulation was developed under TOSSIM [12] environment.
using nesC [8] programming language. Two performance metrics were considered in order to evaluate the relative performance of the two simulated algorithms. Firstly, the number of messages exchanged per resource request/release. Secondly, responsiveness, which is defined as the amount of time that elapses between the instant of time at which process requests a shared resource, until it gets access to that resource.

The conducted experiments have shown that Ricart-Agrawala DMX messaging overhead is higher than Maekawa’s for systems larger than 6 nodes. They also showed that the gap in this overhead gap between the two compared algorithms increases significantly as system size increases. The experiments contradicted the expectation that Maekawa’s DMX responsiveness will be better than Ricart-Agrawala’s DMX. This unexpected result was due to the fact that Maekawa’s simulator used unicasting between the requesting process and its quorum members. On the other hand, Ricart-Agrawala DMX simulator used broadcasting between the requesting process and the rest of the system processes. The time it takes a broadcasted message to reach its destination is much smaller than pulling each quorum member sequentially. Thus, Maekawa’s resource request time is much larger than the time it takes a Ricart-Agrawala’s requesting process to send a broadcast message to the rest of the network. To overcome this anomaly different solutions can be proposed, a simple solution is to start recording system time after request(s) to access the resource has(have) been issued successfully. Another solution is to use the broadcast command in Maekawa’s algorithm and let the receivers that are not members of the requester’s quorum ignore the message received. A third alternative is to develop and use a multicasting communication primitive that is as efficient as the broadcast communication primitive. The conducted research can be extended to investigate the effect of quorum sparseness on the likelihood of encountering deadlocks in Maekawa’s DMX. The simulators can be also extended to investigate multi-hop sensor networks as well as generalized instances of the DMX problems, like the dining (or the drinking) philosophers.

REFERENCES