IJISET - International Journal of Innovative Science, Engineering & Technology, Vol. 1 Issue 7, September 2014.
www.ijiset.com
ISSN 2348 – 7968

Structuring a Token Based Distributed Mutual Exclusion Algorithm for Mobile Hosts

[1]**Deepshikha Sharma,** [2]**Latika Sharma**

Computer Science and Engineering
Bharat Institute of Technology - Meerut, India

Abstract

Distributed Mutual Exclusion Algorithms have been designed for network with static host .A mobile host can connect to the network from different location at different times .In this paper we implement a distributed mutual exclusion algorithm which is based onto a dynamic logical ring and combines the best from the two families of token based algorithms (i.e., token-asking and circulating token) in order to get a number of messages exchanged per CS access (the main performance index for such algorithms) that tends to optimal values under heavy request load. In this paper we implement this algorithm on B.R .Badrinath operational system model. According to this approach we can achieve effective reduction in number of hops per application message by using a specific policy to build on-the-fly the logical ring and we reduced the energy consumption computing power as well as size of available memory.

I. Introduction

A mobile ad-hoc wireless network (MANET) is a collection of mobile nodes that communicate over paths composed of one or a sequence of wireless links.A wireless link is established only if two nodes are within a certain transmission radius. Moreover, the nodes mobility pattern creates unpredictable wireless links formation and removal; as a consequence a path between two nodes can change very frequently due to topology change.

In [1], Badrinath et al.show that also in MANET implementing distributed algorithms that ensures control and ordering is a critical point for many specific problems. Since these networks presents limited device power supply, the resource consumption required for the execution of distributed algorithms should be carefully controlled. In such setting previous specific performance metrics (e.g. power consumption) as well as communication issues (e.g. link failures and recovery) have to be taken into account.

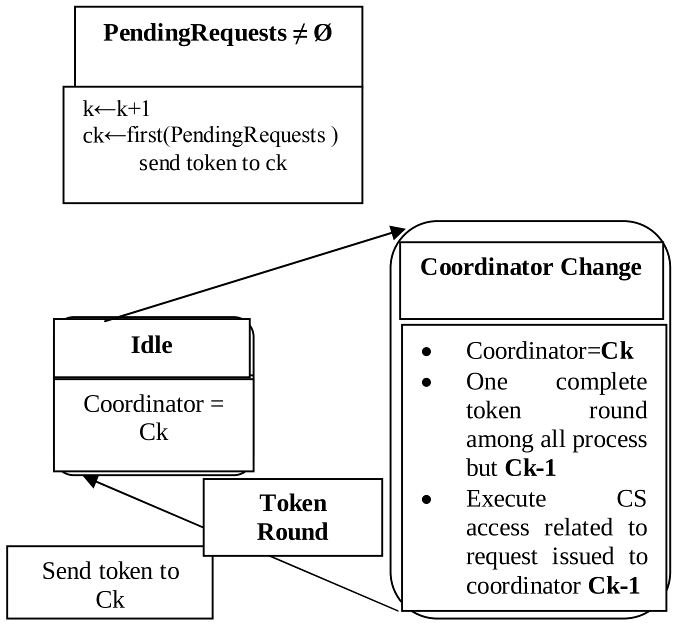
A general DMUTEX algorithm presenting features that allow it to be effectively used over a MANET. This algorithms aims at maintaining device power consumption as low as possible by reducing the number of hops traversed per CS execution and by not sending any control message when no processes request the critical section. The logical ring of processes is dynamic: each time a process receives the token from its predecessor, it decides on the fly which will be its successor between processes that did not yet receives the token during the token round. This decision is based on a given policy P. this policy can be specialized to adapt the algorithm to specific settings. In this paper we choose P to order processes according to their hops' distance. Each round of the ring has a coordinator which is the only process that can block the token to inform processes about coordinator changes. Experimental performance results, obtained by simulation, shows that the proposed DMUTEX algorithm running on a mobile network achieves best performance when request load is heavy.

In this paper we implement this alg6rithm on B.R.Badrinath operational system model .According to this we maintain a logical ring of MSS instead of individual processes so we reduced the time consumption on a particular round and achieve best performance when request load is heavy.

II. EXISTING SYSTEM MODEL AND THE DMUTEX ALGORITHM

The algorithm belongs to the token-asking family. However, it structures processes onto a logical ring by allowing CS access by means of a circulating token (as logical ring algorithms). This ensures that circulating-token optimal scenarios could happen.

The logical ring of processes is dynamic: each time a process receives the token from its predecessor, it decides on-the-fly which will be its successor between processes that did not yet receive the token during the token round. This decision is based on a given policy ***P.*** This policy can be specialized to adapt the algorithm to specific settings. In this we chose ***P*** to order processes according to their hop distance. Each round of the ring has a coordinator which is the only process that can block the token and enter an idle state. The aim of a round of the token is to inform processes about coordinator changes. Experimental performance results, obtained by simulation, shows that the proposed DMUTEX algorithm running on a mobile network achieves best performance when request load is heavy.


PendingRequests ≠ Ø
k←k+1
ck←first(PendingRequests)
send token to ck
Coordinator Change
Coordinator=Ck
One complete token round among all process but Ck-1
Execute CS access related to request issued to coordinator Ck-1
Idle
Coordinator = Ck
Token Round
Send token to Ck

3.1 The DMUTEX ALGORITHM

The algorithm, from an external viewpoint, continuously executes transitions between two states: *Idle* and *Coordinator-Change*. The algorithm evolves in a sequence of rounds during which the two states alternate. A new round starts when the state switches to Coordinator-Change.
For each round of the algorithm, one process is declared the *coordinator*. We will indicate with **CK** the coordinator for round k. The coordinator changes from Ck-1 to Ck. each time there is a transition between Idle and Coordinator-Change state and the process Pi that provokes the transition becomes the current coordinator **Ck.** The latter is the only process enabled to execute the next transition from Coordinator-Change to Idle.

a. Algorithm's State Diagram. A process that wants to access its critical section issues a request to the coordinator and waits for the token. The coordinator inserts the request in a set namely ***pending Request (P),*** ordered according to a certain serialization discipline ***P***. The serialization discipline ***P*** could be deterministic (e.g. the process in the set with the lowest identifier) or driven by some specific knowledge (e.g. FIFO, short-job-first, the closest process to ***Pi*** in terms of some metric such as the number of hops etc.). Initially the algorithm is in the Idle state and the coordinator ***Co*** corresponds to the process ***Pi*** and this information is known by all the processes.

b.Transitions from Idle state to Coordinator-Change state: it occurs when the coordinator ***Ck*** is in the Idle state and the set ***pending Request (P),*** is not empty. During the transition, the coordinator sends the token to the first process in the ordered set ***pending Request (P),***, say ***Pj***, that will become the coordinator ***Ck+1.***

Coordinator-Change state. This state consists of one round of the token on a logical ring. The ring starts from the current coordinator ***Ck*** and it is composed of ***n-1*** processes (the process corresponding to the previous coordinator ***Ck-1*** is excluded from the ring). This round has two targets: (i) informing all the processes about the identifier of the current coordinator ***Ck*** and (ii) allowing requesting processes to access their CS once they receive the token. These requests were issued to the previous coordinator ***Ck-1.***

Transition from Coordinator-Change state to Idle state.
It occurs when the current coordinator ***Ck*** receives back the token from its predecessor in the logical ring embedded in the Coordinator-Change state.

b. Logical Ring The main task of the Coordinator-Change state is to execute a round of a logical ring of ***n-1*** processes. The round starts from the coordinator ***Ck*** and does not include ***Ck-1***. As a consequence the structure of the logical ring has to be computed on-the-fly. i.e., the process ***Pi*** that receives the token has to relay it towards a process (***Pi*** successor) that (i) has not received the token yet in this round and that (ii) is distinct from ***Ck-1***. The last process in this logical ring relays the token to ***Ck*** provoking the transition of the algorithm to the Idle state. As a consequence the

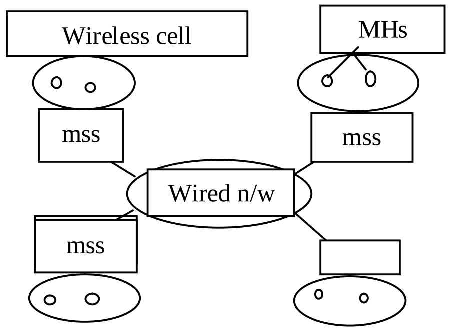
structure of the ring is dynamic and might be different in each round, adapting itself to the mobility of nodes. Operationally the processes passed by the token in a round of a logical ring are kept in a data structure, actually a vector of Boolean of size n, sent with the token, namely *received Token*. *Received Token*[*i*] is FALSE if ***Pi*** has not yet received the token in the current round. During the transition from the Idle state to the Coordinator- Change state, the coordinator ***Ck-1***, say process ***Pi***, sets each entry of ***receivedToken*** to FALSE except ***receivedToken*[i],** which is set to TRUE before sending out both the token and the data structure ***receivedToken*** to the new coordinator ***Ck.*** ***receivedToken*[i]** equal to TRUE avoids that the token during the round in the Coordinator-Change state will pass through ***Pi.*** .
When a process ***Pi*** receives the token along with the data structure ***receivedToken*** during the round coordinated by ***Ck,*** it first sets ***receivedToken*[i]** to TRUE, then defines a set of possible successors ***poss Succi*** in the following way:

Poss Succi = {Pl /*receivedToken* [L] = FALSE ^ (L =/ i)

If **Poss Succi** is empty then the token is sent to the coordinator **Ck** (i.e., the round is finished and the algorithm is going to enter Idle state). Otherwise the set is ordered according to a certain serialization discipline ***P2*** which provides a total order on **Poss Succi** , denoted **Poss Succi (Pi)** ,based on the distance in terms of number of hops between each process in the set and ***Pi***. This distance is calculated by **Pi** basing on information provided by the underlying routing protocol. If a node is temporarily disconnected, it is considered to be at infinite distance. If two processes are at the same distance, the one with the lowest identifier comes first.The first process in the ordered set **Poss Succi(P)**, say ***Pj*** , is selected and the token is sent to it. If **Poss Succi(P)** contains only processes at infinite distance, ***Pi*** waits and tries to retransmit after a certain amount of time. For the assumptions made will be eventually reachable and will receive the token.
In this way, the ring is built dynamically by always passing the token to the node that requires the lowest effort in terms of network resources and power supply. Of course this discipline does not ensure optimal behavior. However, it provides a good heuristic, actually a simple greedy policy that takes into account mobility with the only additional assumption of querying the local routing table. If a more sophisticated behavior is required, ***P*** can be easily substituted without impacting the rest of the algorithm.
Figure 2 shows an example of how the algorithm builds a logical ring. In the example, each process in the system is connected to any other. Processes in will be ordered according to the hop count between ***Pi*** and all the other processes, calculated by the routing layer at the time the token have to be sent. Then, when ***Pi*** sends the token it chooses as receiver the closest process among the ones that have not received the token yet. The Figure depicts the physical configuration of processes. A link (thin dotted line) between two processes indicates that they are in each other's transmission range. In this example, we assume for clarity that nodes do not move. The thick arrows highlight the actual logical ring formed by the sequence of token transmissions.

III. PROPOSED IDEA

If we implement this algorithm on the B.R.Badrinath operational system model which is given below then we can reduce the power consumption ,communicational cost , as well as time consume in each token round.



Now we present justifications for this choice:
In this algorithm, all the participants are logically arranged in a unidirectional ring .when a process that wants to enter the critical section, issued a request to the coordinator and wait for the token. The coordinator inserts the request in a set, namely ***pending request (P)***,

ordered according to a certain serialization discipline ***P.*** Each Participant executes as follows:

- Wait receipt for Token from its predecessor in the ring.
- Coordinator = Ck
- One compute token round among all processes but not Ck-1
- Execute critical section access related to requests issued to coordinator Ck-1.

All communication in this algorithm, thus occur only along the channels that define the logical ring.

Algorithm R1 Consider an execution of the algorithm directly on the MHs wherein the N MHs from the logical ring. Now, the sender and recipient of every message is a MH and incurs a cost ***2 Cwireless + Csearch*** . Thus, the total communication cost for the token to traverse the ring once, is ***N x (2 Cwireless+Csearch).*** Note that this cost is independent of ***K,*** the number of mutual exclusion requests satisfied.

Inefficiency in maintaining a logical structure amongst mobile hosts stems from the fact that, unlike fixed hosts, the physical connectivity amongst mobile hosts is redefined on every move: this manifests itself through an increase in the search component of the overall communication cost of the algorithm.

Algorithm R2 Algorithm R2 maintains a l6gical structure amongst the MSSs: a token circulates amongst the M MSSs logically arranged in a unidirectional ring. Each MSS maintains a request queue .A MH that needs to access the CS send it request to the local MSS, which then inserts it at the tail of its request queue. When the token arrives at a MSS, all pending requests from the request queue are moved to a grant queue. The MSS holding the token then sequentially services each entry in a grant queue: it deletes the request at the head of the queue, sends the token to the MH that made the request and awaits return of the token from the same MH. This is repeated till grant queue is empty. The MSS then transfer the token to the next MSS which is would be successor of current MSS according to policy ***P*** and follow this process until pending request is not empty and when pending request is empty then coordinator is in its idle state and after that when new request is generated then change the coordinator change state.

Algorithm **R2** illustrates an interesting interplay between mobility of hosts and the movement of the token amongst the static **MSSs:** after a **MH's** request is satisfied at the current MSS, it is possible that it moves to a new cell under a **MSS** which is the next recipient of the token in the logical ring. The **MH** may then submit a new request at this **MSS,** prior to the arrival of the token. **Thus,** multiple requests from the same **MH** may be satisfied (at different **MSSs)** in a single traversal of the ring, and the total number of requests that may be satisfied is thus ***N*** x ***M.***

To prevent a **MH** from accessing the token more than once in one traversal of the ring, the token is associated with a counter ***(token-Val)*** which is incremented every time it completes one traversal. Each **MH** maintains a local counter ***access-count*** whose current value is sent along with the **MH's** request for the token to the local MSS. A pending request is moved from the ***request queue*** to the ***grant queue*** at the **MSS** holding the token, only if the request's ***access-count*** is less than the token's current ***token-Val.*** When a **MH** receives the token, it assigns the current value of ***token-vu1*** to its copy of ***access-count.*** We will refer to this variation of R2 as R2'. The choice of using R2 or **R2'** is governed by the following trade-off: **R2** sacrifices "fairness" at the expense of satisfying more number of requests in one traversal of the ring, while R2' ensures at most one access to the token by a **MH;** both incur the same fixed cost to circulate the token amongst the **MSSs.**

Comparison of algorithms RI and R2

- **Search cost:** The total search overhead incurred by algorithm **RI** is ***proportional to N,*** the number of **MHs** constituting the ring and is independent of the number of mutual exclusion requests satisfied in one traversal of the ring. Algorithm R2 and its variation R2' incur a search overhead ***proportional to K,*** the number of mutual exclusion requests satisfied. Both **R2** and **R2'**incur a fixed cost for circulating the token amongst the ***M*** MSSs.

- **Disconnection and doze mode** Algorithm **R1** is vulnerable to disconnection of ***any*** MH and requires the logical ring to be re-established amongst the remaining **MHs** when one or more **MHs** disconnect. However, with R2, disconnection of a **MH** that has not submitted a request for mutual exclusion, does not affect the rest of the system at all. In addition, it is also easy to handle disconnection of a **MH** with a pending request in **R2:** when the token is received by the local **MSS** of such a **MH** (after searching for it). it observes that a "disconnected" flag is set for the particular **MH** and returns the token back to the sending MSS. **R1** also interrupts a **MH** operating in doze mode if it happens to be the next recipient of the token in the logical ring, irrespective of whether it made a request for mutual exclusion or not. **In** contrast, **R2** interrupt a **MH** in doze mode only to satisfy a prior request from that **MH.**

- **Battery consumption** Algorithm **R1** consumes battery power of every MH to first receive the token from its predecessor in the ring, and then transmit it to the successor .as we know R2 avoids consuming battery power at ***every*** **MH:** only those **MHs** that request the token expend battery power to access the wireless network thrice, i.e. to transmit the request, receive the token and then return it back.

- ***Variations.*** In R2', it is possible that a "malicious" **MH** could present a ***access-count*** lower than its me value. The following variation eliminates such a possibility:

The token now contains a ***token-list*** of ***44, h***> pairs, where **M** is the **MSS** where the **MH** ***h*** last accessed the token during the token's current traversal of the ring.
On arrival of the token, ***M*** deletes all pairs from ***token-list*** whose first element is ***M.*** Then, a pending request from a **MH** ***h*** in the ***request queue*** is moved to the ***grant queue*** only if ***h*** is not present as the second element of any pair in teh updated ***token-list.*** After h's request is serviced, ***M*** adds ***44,*** *h*> to ***token-list.***
The above scheme ensures that if *h* receives the token from ***M,*** then a subsequent request from h will be serviced only after the token visits every **MSS** in the logical ring.

I. CORRECTNESS :

This idea ensures that if the timestamp of a request R1 is less than that of another request R2 , then R1 will be satisfied before R2.In our model a request from a MH is timestamped when the init() message is received by its local MSS i.e. through MHs do not maintain logical clocks ,the timestamp assigned to request(h1) by m1 can be considered as the timestamp of h1's request for mutual exclusion. Since the Coordinator execute the concept of Mutual exclusion and a grant_request message will be sent to h1 before another MH h2 if the timestamp assigned to request (h1) is less then that of request (h2).

II. CONCLUSIONS :

The design algorithms for distributed system and their communication costs have been based on the assumption that the location of the hosts in the network do not change and the connectivity amongst the host is static in the absence of failures .However,with the emergence of mobile computing these assumptions are no longer valid .Additionally , mobile hosts have severe constraints on energy consumption , computing power and size of available memory ,compared to fixed hosts.
This paper presented a new algorithm which is implemented on operational system model for the mobile computing environment and describes a simple and useful principle for structuring distributed mutual exclusion algorithm.

VI. REFERENCES

[1] Arup Acharya and B. R. Badrinath. Delivering multicast messages in networks with mobile hosts. In *Proc. of the 13th IntL Con$ on DistributedComputing Systems,* May 1993.
[2] Andrew Athas and Dan Duchamp. Agent-mediated message passing for constrained environments. In *USENIX Symposium on Mobile and Location-Independent Computing,* Aug. 1993.
[3] Baruch Awerbuch and David Peleg. Concurrent online tracking of mobile users. In *Proc. ACM SIGCOMM Symposium on Communication., Architectures and Protocols,* September 1991.
[4] B. R. Badrinath, Amp Acharya, and Tomasz Imielinski. Impact of mobility on distributed

computations. *ACM Operating Systems Review*, 27(2), April ’93.
[5] B. R. Badrinath and T. Imielinski. Replication and mobility. In *Proc. of the* 2”” *workshop on the management of replicated data*, pages 9-12, 1992.
[6] P. Bhagwat and Charles E. Perkins. A mobile netwcrking system based on internet protocol (ip). In *USENIX Symposium on Mobile and Location-Independent Computing*, Aug. 1993.
[7] Micheal Bender et. al. Unix for nomads: Making unix support mobile computing. In *USENIX Symposium on Mobile and Location-Independent Computing*, Aug. 1993.
[SI T. Imielinski and B. R Badrinath. Querying in highly mobile distributed environments. In *lgh Intl. Conference on Very Large Databases*, pages 41-52, 1992.
[9] T. Imielinski, S. Viswanathan, and B. R. Badrinath. Power efficient filtering of data on the air. In *EDBT ’94*, 1994.
[10] J. Ioannidis, D. Duchamp, and G. Q. Maguire. Ipbased protocols for mobile internetworking. In *Proc.* of *ACM SIGCOMM Symposium on Communication*, *Architectures* and *Protocols*, pages 235-245, September 199 1.
[11] L. Lamport. Time, clocks and the ordering of events in a distributed system. *Comm. ACM*, 21(7):558- 565, 1978.
[12] G. Le Lann. Distributed systems, towards a formal approach. *IFIP Congress*, *Toronto*, pages 155-1 *60*, 1977.
[13] Fumio Teraoka, Yasuhiko Yokote, and Mario Tokoro. A network architecture providing host migration transparency. *Proc. of ACM SIGCOMM’91*, September, 1991.
[14] Hiromi Wada, Takashi Yozawa, Tatsuya Ohnishi, and Yasunori Tanaka. Mobile computing environment based on internet packet forwarding. In *1992 Winter Usenix*, Jan. 1993.
[15] T. Watson and B. N. Bershad. Local area mobile computing on stock hardware and mostly stock software. In *USENIX Symposium on Mobile and Location-Independent Computing*, Aug. 1993.
.