K-Mutual Exclusion in Decentralized Systems

*Cairo University, Faculty of Engineering Advanced Operating Systems, Computer Department 2020.

Ahmed Mahmoud Abd El-Monaem
Faculty of Engineering, Computer Department
Cairo University
Cairo, Egypt
Ahmed.Afifi98@eng-st.cu.edu.eg

Shady Fahmy Abd El-Hafez
Faculty of Engineering, Computer Department
Cairo University
Cairo, Egypt
shadyfahmy67@gmail.com

Abdulrahman Khalid Hassan
Faculty of Engineering, Computer Department
Cairo University
Cairo, Egypt
abdulrahman.elshafei98@gmail.com

Hossam Ahmed Mahmoud
Faculty of Engineering, Computer Department
Cairo University
Cairo, Egypt
hossamahmed201515@gmail.com

***Abstract*—In this paper we propose a distributed algorithm that achieves mutual exclusion among number of machines in a computer network. this network is decentralized which means that there's no controller or moderator for the whole system. Machines can only communicate by sending and recieving messages. The proposed algorithm is a token-based algorithm.**

***Index Terms*—K-mutex, mutual exclusion, operating systems, distributed systems, token-based**

I. Introduction

Mutual Exclusion is used to avoid race conditions between two or more concurrent processes. This means that a process can't enter its critical section while another process is executing its critical section i.e. only one process can execute the critical section at any given time instance.

In a single computer system, processes share the same memory and share the same other resources. This makes the mutual exclusion problem easily solved by using shared variables i.e. semaphores or monitors. In distributed systems, processes run on different machines that don't share the same memory, so the mutual exclusion problem can't be solved using semaphores. Solutions for mutual exclusion problem in distributed systems mainly use message passing.

This problem can be solved by different approaches. The solution can be token-based where a unique token is shared among nodes. A node can enter its critical section only if posses the token. Non-token based approach where each node requests from all other nodes in the system to enter its critical section. This approach requires an exchange of two or more successive rounds of messages between system nodes. The last approach is a quorum-based approach where each node requests permission from only a subset of nodes called a quorum. Any two quorumscontain a common node that is responsible to ensure mutual exclusion in the system.

Mutual exclusion algorithms should satisfy some properties and requirements.

- **Safety**: This means that any instance only one process can execute its critical section.
- **Liveness**: Which means the absence of deadlock and starvation. the message that the process is waiting for to enter its critical section will never arrive.
- **Fairness**: This means that each process should get a fair chance to enter the critical section depending on the order of the request.
- **Fault tolerance**: This means that in case of failure system should detect this failure to continue its normal function.

On implementing mutual exclusion algorithms, after satisfying the basic requirements, performance should be taken into consideration. Performance is measured by four basic metrics:

- **Message Complexity**: Number of messages required by a node to enter its critical section.
- **Synchronization Delay**: Time required by the next node to enter the critical section after the current node leaves its critical section.
- **Response Time**: Time interval between a request and completing execution of the requested critical section.
- **System Throughput**: The rate at which the system executes requests for a critical section.

The distributed k-mutual exclusion problem is the problem of permitting k nodes to enter the critical section where $1 \leq k \leq n$ and n is the total number of nodes. More than one process can enter the critical section simultaneously due to the nature of the resource or the resource it has more than one entity that can be accessed at the same time. For example, a local network having many computers that execute different applications, the k-mutual exclusion algorithm is used to allocate bandwidth to different processes. The bandwidth is the shared resource in this example. A distributed database that allows more than one process to access data from the same section is another example of an application that uses k-mutual exclusion.

In this paper, we modified an existed algorithm that satisfies the basic requirements of mutual exclusion and works on improving fairness and performance metrics of other k-mutual exclusion algorithms.

II. Related Work

Several k-mutual exclusion algorithms have been developed over the past 20 years. they can be divided into two main categories, token-based algorithms, and permission-based algorithms.

Lamport [11] proposed the first mutual exclusion algorithm for distributed systems. In Lamports algorithm, when a node i wants to enter its critical section it must send a request message with a time stamp to each node of all the other $n-1$ nodes in the network. When it receives a permission message from each node of the other $n-1$ nodes then node i can then enter its critical section. When node i exits the critical section, a release message is sent to all other nodes to indicate that request has been granted. This algorithm requires $3(n-1)$ messages for a system of n nodes for a single critical section request.

Carvalho and Roucairo [14]proposed an algorithm which achieves the same requirements as Ricart and Agrawala's algorithm but this algorithm requires between 0 and $2(n-1)$ messages. When a node j receives permission message from a node i, this permission is considered a valid permission till node j receives another request from node i to enter its critical section . When node j wants to enter its critical section, it sends request messages to only nodes for which node j does not have valid permissions.

Banerjee and Chrysanthis [13] proposed a token based algorithm for mutual exclusion in distributed systems for a fully connected network. A node k is selected such that all requests from nodes that want to enter the critical section are sent to this node (request collector). We select any node to be a request collector at initialization. When node k receives a request, request queue is copied to the token which is held by node k and is sent to the first node in the request queue. Node k broadcasts a message to all the nodes in the network to inform them that there is a new request collector which is the node at the rear of the token queue. The token keeps moving from one node to another according to the order in the token queue until the last node is reached. If a node had sent a request to node k before it received the broadcast message but after the token was sent, then node k passes the token to the new request collector.

Raymond proposed a generalization to k-mutual exclusion but his algorithm is permission-based which has a very high message complexity. [4]. Srimani and Reddy [5] extended the ideas of Suzuki and Kasami [6], they proposed a token-based algorithm which involves too many requests and that makes the algorithm poorly suitable for distributed systems. Also, it is not fault-tolerant and does not consider fairness.

Ricart and Agrawala proposed an algorithm [7] which uses a single token and a counter to keep track of the number of processes that are executing the critical section currently. It is shown that this algorithm behaves like a single token algorithm under high load. Also, this algorithm requires all nodes to communicate with each other which involves many messages making it message inefficient for distributed systems. Also, it is not fault-tolerant and does not consider fairness.

Bulgannawar and Vaidya [8] proposed an algorithm which is a token-based and uses k separate tokens. The algorithm uses k instantiations of a dynamic forest tree structure. The algorithm uses heuristics for choosing a tree among k, to send the request. This algorithm is not fair, as there is no load balancing of requests. Also, the algorithm is not fault-tolerant.

Neilsen and Mizuno [9] tried to reduce the k-mutex problem to k one-mutex subproblems but in their algorithm each node has exactly k tokens or permissions which may result that number of nodes greater than k enter their critical section at the same time.

Lang and Mao [10] arrange system nodes in a rectangle form called torus such that there are k torus and each torus has exactly one node can enter its critical section at any give time after taking permissions from all other nodes in the torus so the message complexity depends on torus size which is equal to $\frac{n}{k}$.

Chaudhuri and Thomas Edward algorithm [1] proposed a token based algorithm for mutual exclusion in distributes systems. The algorithm divides a network of n nodes to $\sqrt{n}$ groups of $\sqrt{n}$ nodes each. Each group has a request collector that collects the requests from the local nodes of its group. The network has one global request collector which is one of the local request collectors. Global request collector receives requests from the request collector of each group. The global request collector holds K tokens and it passes them according to requesting nodes. This algorithm has a worst case message complexity of $O(\sqrt{n})$.

III. Methodology

In this paper, we propose a new algorithm for the k-mutual exclusion problem. Our algorithm is based on Pranay Chaudhuri and Thomas Edward algorithm [2]. We have added many modifications to their algorithm to improve fairness and response time.

A. The network.

The network structure consists of n nodes. Nodes in the network are divided into $\sqrt{n}$ local groups (LG). Each local group consists of $\sqrt{n}$ nodes. Each local group has a node that is called a local request collector (LRC). The LRC collects requests from nodes in its local group that requires to enter the critical section. The group of LRC nodes is called global group (GG) From these LRC nodes a node is chosen to be the global request collector (GRC) which holds k tokens and receives requests from the LRC nodes then it is responsible for token distribution among the requesting local nodes. Fig.15 illustrates the network structure consists of 16 nodes.

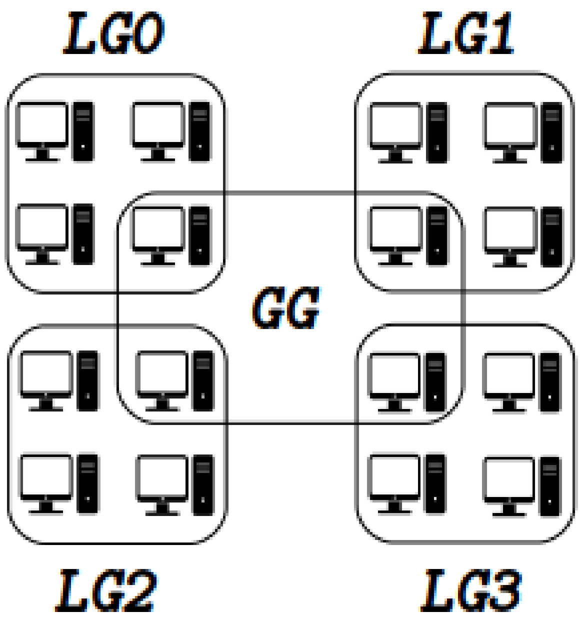

Fig. 1. The network.

B. Description of the algorithm.

When a node requests a token to enter its critical section it sends a request to LRC of its group and this node is enqueued in the Local Request Queue (LRQ). Then LRC sends a request to GRC and enqueued in the Global Request Queue (GRQ). When the GRC has all tokens, It divides them among all LRCs that their LG has at least one node requests to enter its critical section (LRCs in the GRQ) and then dequeue them all.

GRC starts to respond to the requesting nodes by passing tokens to them. This is called the token forwarding phase.

Let p equals the total number of requests contained in LRQ, let q be the total number of requests contained in GRQ and let k equal the total number of tokens. Token forwarding from GRC to LRCs has several cases according to p and q.

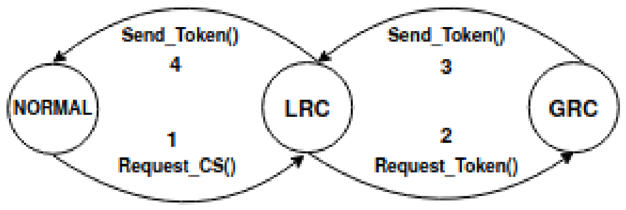

Fig. 2. Requesting Token Process.

- **1.** If $p > 0$ and $q = 0$:
  This means that nodes in the local group of GRC request access to critical sections while nodes in other groups do not.
  - **A.** If $p \leq k$:
    nodes requesting tokens are less than or equal the total number of tokens, so create p tokens and send a token to each requesting node.
  - **B.** If $p > k$:
    nodes requesting tokens are more than the total number of tokens and it means that requests can not be satisfied at once. So, we send all the available tokens. We loop on all the tokens and for each token queue add a requesting node from LRQ and dequeue. Do this until LRQ is empty then these tokens are forwarded to nodes in their queue.

- **2.** If $q > 0$ and $p = 0$:
  This means that nodes contained in the local group of the GRC does not request tokens and nodes from outer groups require tokens.
  - **A.** If $q \geq k$:
    in this case we initiate k tokens and we loop on tokens and for each token queue add a requesting node from the GRQ. Repeat until GRQ is empty.
  - **B.** If $q < k$:
    in this case we divide tokens equally among requesting LRCs such that each LRC gets k/q tokens.

- **3.** If $p > 0$ and $q > 0$:
  This means that nodes in the local group of GRC and nodes from other local groups request tokens.
  - **A.** If $q \geq (k - 1)$:
    in this case $(k - 1)$ are forwarded to requesting LRCs. We loop on all the tokens and for each token queue add a requesting node from GRC and dequeue. Do this until GRQ is empty then these tokens are forwarded to nodes in their queue.
  - **B.** If $q < (k - 1)$:
    in this case we divide tokens equally among the local groups such that each requesting LRC receives $k/(q + 1)$ tokens. And $k/(q + 1)$ tokens are kept

for the nodes in the local group of the GRC.

In the original algorithm [2] It sends only one token to each LRC. Each LRC receives the tokens and starts to distribute these tokens on the nodes in LRQ. Each token has its own queue so LRC distributes those tokens among those queues. the token with its queue are passed to the first node in this token queue and When it exits its critical section, it dequeues it from the token queue then pass the token with its queue to the next node in the queue and so on. If the token queue is empty, the token and its queue returned to the LRC of that local group. When LRC receives the tokens back, it sends them back directly to the GRC.

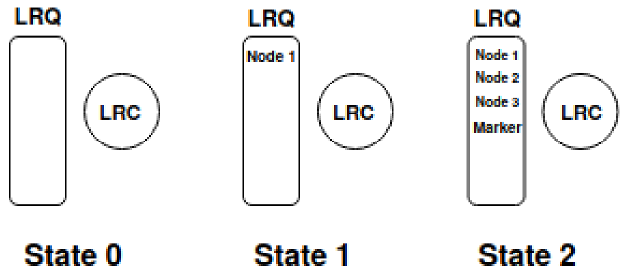

Fig. 3. LRC Phases.

The GRC won't wait to collect all tokens to start sending them again to LRCs but once it receives the tokens from one of the LRCs it will directly forwarding them to another LRC that can help it to finish its requests much faster.

The dynamic distribution of tokens in that way improves the fairness and response time of our algorithm. In Pranay Chaudhuri and Thomas Edward algorithm [2], only one token was sent to each LG and other tokens were circulating in GRC local group, which may cause a local group to have a lot of requesting nodes while GRC local group does not contain any requesting nodes or only one requesting node. In our algorithm every LRC has equal number of tokens and after GRC receiving some tokens back from the LRCs with empty LRQs, it forwards these tokens back to an LRC which needs them so resources are used as much as possible and unused tokens are minimized as much as possible and more tokens are available for this LRC which got the forwarded tokens so its nodes enter the critical section much faster as more shorted queues due to the larger number of tokens, it has.

C. Correctness Proofs.

- **Starvation free:** Any node wants to enter its critical section will be enqueued into the LRQ at the LRC node of its local group (assuming reliable network and message system and there's no message loss). So, after receiving the token from GRC node and the requested nodes are enqueued in this token queue. All nodes into that queue will enter its critical section when it goes to the front of the queue, as nodes are dequeued after exiting their critical section. Similarly this will happen at the GRQ.
- **Mutual Exclusion:** The main idea of the algorithm is to make a shared resource available for only k nodes at any given time. This is guaranteed by the idea of tokens which are non-sharable and indivisible variables and no node can enter its critical section without acquiring one of them.
- **No node can hold a token forever:** Using reliable network and message system and assuming that there's no token loss, we can easily prove that there's no node can hold any token indefinitely.

  All nodes in the system rather than LRCs and GRC node can hold the token only if it's in its critical section then it will directly send it to the next node in token queue or send it back to its LRC after exiting its critical section. Regarding LRC nodes, after receiving a token from the GRC node they use it to enter their critical section or send it directly to other nodes who wants to enter their critical section in their local group, and after receiving the token back from nodes and their local groups, they send it directly back to GRC node. Therefore, no node can hold a token forever.
- **Deadlock free:** Since circular wait is one of the conditions that must be achieved for causing deadlock, and we already prove that there's no node can hold a token forever and the token must be returned to the GRC so we can prove that the circular waiting condition can't happen here. Hence, the algorithm is deadlock free.

D. Complexity Analysis.

- **Message complexity:** Assuming that there are only one node wants to enter its critical section. The best case scenario is that the GRC is this only node that wants to enter its critical section, therefore, It can enter its critical section with zero delay and no messages. The worst case scenario is that this node which wants to enter its critical section is in some local group that doesn't have the token yet, therefore, the first message will be sent from this node to the LRC of its local group, then this LRC will send another message to the GRC asking it for a token. there are two more messages that will be sent in the opposite direction from GRC to LRC and from LRC to this node so that it can enter its critical section, and after exiting the critical section there are two more messages to return the token back to the GRC. 6 messages in total for this node to enter its critical section. And we can see that 6 messages are nothing compared to n messages in the naive permission-based algorithms.

Assuming that all nodes in the network wants to enter their critical section at the same time. At first we'll have n messages from all nodes in the network to the LRCs in each local group, then we'll have $\sqrt{n}\ -1$ messages from all LRCs to the GRC asking for tokens. then the GRC will generate K tokens and send them to all LRCs such that each LRC has the same number of tokens which require another $\sqrt{n}\ -1$ messages. Thats brings us to a total of $2*(\sqrt{n}-1)$ messages, tokens are circulated to the $\sqrt{n}$ nodes of that group. After the local group does not need its tokens any more it sends it back to the GRC that is another $\sqrt{n}-1$ and the GRC forwards them to another LRC that is $\sqrt{n}-2$ excluding the first LRC which sends the tokens back thats bring us with a total of $4*(\sqrt{n}-1)-1$, Hence the algorithm has a worst-case message complexity of $O(\sqrt{n})$.

- **Space complexity:** Nothing require much space rather than queues, and in our algorithm any queue can have at most $\sqrt{n}$ entery at a time and each node can have only one qeueu at a time which is token queue so our space complexity is $O(\sqrt{n})$.

IV. Experimental Reuslts:

let's define some variables:

- M is average time to send a message between two nodes.
- E is average time needed to execute the critical section.
- K is the number of tokens for a specific resource.
- Q is the number of groups that have at least one node wants to enter its critical section.
- N_i is the number of nodes which want to enter their critical section in group i.

- **Case 1:** All groups have the same number of nodes that want to enter their critical section and the total number of tokens for some resource is greater than the number of groups that have at least one node request the critical section. $[K > Q\ and\ N_i = N]$.

  The system will follow the following scenario:

  1) N nodes request critical section in each group i from the Q groups.
  2) GRC sends $S = \dfrac{K}{Q+1}$ tokens equally to every group's LRC which are stored in the GRQ
  3) Every LRC receives S tokens and pushes their N requested groups nodes stored in its LRQ in S token queue, every token queue will have $T = N/S$ requested nodes
  4) Time in every group to finish all its nodes requests $G = T*(M+E)$
  5) Time to receive tokens back to the GRC would be $B = G + M$

Considering all 5 steps, we can calculate total time to execute all critical sections and return tokens back to GRC and we can also calculate the total number of messages by the system until returning tokens back to GRC using the following equations.

Time = $3M + \dfrac{N}{S}(M+E) + M$
Where $3M$ is the time to send message from the node requesting the critical section to the LRC and message from LRC to GRC and then the message holding the token sent from GRC to LRC. and $\dfrac{N}{S}(M+E)$ is the time that will be taken by the token to circulate in that group until it returns to the LRC of that group. here we divided by S as now we have more than one token in each group. And finally M is the time taken to return the token from the LRC to the GRC.

Number of Messages = $NQ + 2Q + \dfrac{N}{S}Q + 2Q$
Where NQ is the number of messages for all nodes to send their requests to their LRC in their group. And then $2Q$ is the messages required for all LRCs to ask the GRC for the token and for the GRC to send the token to all LRCs. And another $2Q$ to return the token back to the GRC. And finally $\dfrac{N}{S}Q$ is the messages required for the token to circulate in any group.

Using the same Assumptions we can calculate the same parameters for Chaudhuri and Edward Algorithm [2]

Time = $3M + N(M+E) + M$
Where $3M$ is the time to send message from the node requesting the critical section to the LRC and message from LRC to GRC and then the message holding the token sent from GRC to LRC. and $N(M+E)$ is the time that will be taken by the token to circulate in that group until it returns to the LRC of that group. And finally M is the time taken to return the token from the LRC to the GRC.

Number of Messages = $NQ + 2Q + NQ + 2Q$
Where NQ is the number of messages for all nodes to send their requests to their LRC in their group. And then $2Q$ is the messages required for all LRCs to ask the GRC for the token and for the GRC to send the token to all LRCs. And then the exactly same number of messages to return the token back to the GRC.

- **Case 2:** Groups have different number of nodes that want to enter their critical section such that group i have i nodes requesting critical section. the total number of tokens for some resource is greater than the number of groups that have at least one node request the critical section. $[K > Q \ and \ N_i = N_{i+1}/2 \ ]$.

The system will follow the following scenario:

1) N_i nodes request critical section in the group i of the Q requested groups

2) GRC sends $S = \dfrac{K}{Q+1}$ tokens equally to every group's LRC which are stored in the GRQ

3) Every LRC receives S tokens and pushes their N requested groups nodes stored in its LRQ in S token queue, each token queue will have $T_i = \dfrac{N_i}{S}$ requested nodes

4) Time in every group to finish all its nodes requests $G_i = T_i(M + E)$

5) Every finished Group will give tokens back to the GRC.

6) The GRC forwards the token to next LRC_{i+1} stored in GRQ COPY.

7) LRC_{i+1} divides the remaining requested nodes on the tokens they have so the messages now decreased by $\dfrac{1}{4}$ as the group $i+1$ has already finished $\dfrac{1}{2}$ the requests of critical sections.

Considering all 7 steps, we can calculate total time to execute all critical sections and return tokens back to GRC and we can also calculate the total number of messages by the system until returning tokens back to GRC using the following equations.

Time = $4M + G_f$

$$G_f = \frac{N_f(Q+1)}{2K}(M+E) - G_{f-1} + 2M$$

Number of Messages =

$$3Q + \sum_{i=0}^{Q-1} N_i + (N_0\frac{Q+1}{K}) + (3\frac{Q+1}{4})\sum_{i=1}^{Q-1} N_i$$

Using the same Assumptions we can calculate the same parameters for Chaudhuri and Edward Algorithm [2]

Time = $3M + N_{Max}(M + E) + M$
Where $3M$ is the time to send message from the node requesting the critical section to the LRC and message from LRC to GRC and then the message holding the token sent from GRC to LRC. and $N_{Max}(M+E)$ is the time that will be taken by the token to circulate in that group until it returns to the LRC of that group,N_{Max} here as all groups are working in parallel so the group that has maximum nodes will be the last one to return the token to the LRC. And finally M is the time taken to return the token from the LRC to the GRC.

Number of Messages = $4 \times Q + (2\times \sum_{i=0}^{Q} N_i)$
Where $4Q$ is the number of messages required to send the tokens from the GRC to all LRCs and to return the token back from all LRCs to the GRC. the process of requesting the critical section and circulating the token inside the group depends on N_i for each Group.

V. Conclusion

In this paper, we have proposed an algorithm for the k-mutual exclusion problem. We have presented the general idea of the algorithm followed by a detailed description. We have proved the validity of the algorithm as we proved that it is deadlock free and starvation free. We also proved safety as no more than k nodes can access the critical section at the same time.There are k tokens in the system.The network contains n nodes. The logical structure for the network differentiates between local nodes and link nodes. We could achieve message complexity of $O(\sqrt{n})$ in the worst case. For space complexity nothing requires much space rather than queues, and in our algorithm any queue can have at most $\sqrt{n}$ entry at a time and each node can have only one queue at a time which is the token queue so our space complexity is $O(\sqrt{n})$.

References

[1] P. Chaudhuri, T. Edward, An O(n) distributed mutual exclusion algorithm using queue migration, Journal of Universal Computer Science 12, 2006.

[2] P. Chaudhuri, T. Edward, An algorithm for k-mutual exclusion in decentralized systems.

[3] Ismail M. Nur, Armin Lawi, A formal model of quorum based k-mutex algorithm using input/output automata.

[4] K. Raymond. Multiple entries with ricart and agrawalas distributed mutual exclusion algorithm. Technical Report 78, University of Queensland, 1987.

[5] P. Srimani and R. Reddy. Another distributed algorithm for multiple entries to a critical section. IPL, 1992.

[6] I. Suzuki and T. Kasami. A distributed mutual exclusion algorithm. ACM TOCS, 3(4):344349, 1985.

[7] G. Ricart and A. Agrawala. An optimal algorithm for mutual exclusion in computer networks. Communications of the ACM, 1981.

[8] S. Bulgannawar and N. Vaidya. A distributed k-mutual exclusion algorithm. Proc . ICDCS, pages 153160, 1995.

[9] M.L. Neilsen, M. Mizuno, Nondominated k-coteries for multiple mutual exclusion, Information Processing Letters 50 (1994)

[10] S.D. Lang, L.J. Mao, A comparison of two torus-based k-coteries, in: Proceeding of the 1998 International Conference on Parallel and Distributed Systems, 1998.

[11] Lamport, Time, clocks and the ordering of events in a distributed system, Communications of the ACM 21, 1978, 558-565..

[12] G. Ricart and A. K. Agrawala, Authors response to On mutual exclusion in computer networks by Carvalho and Roucairol, Communications of the ACM 26. 1983, 147 - 148.

[13] S. Banerjee and P. K. Chrysanthis, A new token passing distributed mutual Exclusion algorithm, Proceeding of Intl. Conf. on Distributed Computing Systems (ICDCS), Hong Kong, 1996, 717 - 724.

[14] O.S.F. Carvalho, C. Roucairo, On the mutual exclusion in computer networks, Communications of the ACM 26 (2) (1983)

VI. Appendix

A. Code Snippests.

. **Init:** Initialize the node and determine its type.

```
FUNCTION Init(i, j, ip)
    id <- i, j
    myIp <- ip
    requested <- False
    mutex <- False
    requestSent <- False
    tokenQueue <- None
    tokens <- None
    IF(id[1] = 1)
        type <- LRC
        totalTokenSent <- 0
        IF(id = (0, 1)):
                type <- GRC
            numOfTokens <- TOTAL_TOKENS_NUM
        ELSE:
            type <- NORMAL
            numOfTokens <- 0
        ENDIF
      ENDIF
    ConfigPubSubSockets()
ENDFUNCTION
```

Request Critical Section: Function called by any Node to declare that it wants to enter its critical section.

```
FUNCTION Request_CS()
    requested <- True
    IF (type = NORMAL)
        Msg <- {"MsgID": NODE_LRC_REQ_CS,
                "ID": id[1]}
    ELSE:
        LRQ.enqueue(id)
        IF (numOfTokens > 0)
            numOfTokens -= 1
            run_CS()
            release_CS()
            numOfTokens += 1
        ELSEIF (type = LRC)
            Msg <- {"MsgID": LRC_GRC_REQ_CS,
                    "GID": id[0]}
            requestSent <- True
    ENDIF
        ENDIF
```

. **GRC Changer:** Function used by the old GRC to declare the new one

```
FUNCTION Broadcast_New_GRC(newGRCId)
    Msg <- {"MsgID": MsgDetails.NEW_GRC,
            "GRCId": newGRCId, "GRQ": NodeNew.
                GRQ}
    IF(type = NodeType.GRC)
        type <- NodeType.LRC
        NodeNew.GRQ <- Queue()
```

. **Receive Request:** Receive request to enter the critical section.

```
FUNCTION Recieve_request(fromNode)
    IF (type = LRC):
        LRQ.enqueue(fromNode)
        IF (not requestSent)
            Msg <- {"MsgID": LRC_GRC_REQ_CS,
                    "GID": id[0]}
            requestSent <- True
        ENDIF
    ELSEIF (type = GRC):
        IF (fromNode[1] = 1)
            GRQ.enqueue(id)
        ELSE:
            LRQ.enqueue(fromNode)
        ENDIF
        send_token()
    ENDIF
ENDFUNCTION
```

. **Receive Token:** Receive Token from LRC or GRC.

```
FUNCTION Recieve_Token(tokenQueue)
    tokenQueue <- tokenQueue.copy()
    IF (type = NORMAL):
        IF(requested):
            mutex <- True
            tokenQueue.dequeue()
            run_CS()
            release_CS()
        ELSE:
            send_token()
        ENDIF
    ENDIF
    IF(type = LRC):
        numOfTokens += 1
        IF (not sameLocalGroup):
            LRQ.enqueue(MARKER)
            requestSent <- False
            IF (requested AND LRQ.front == id)
                mutex <- True
                LRQ.dequeue()
                run_CS()
                release_CS()
            ELSE:
                send_token()
            ENDIF
        ELSE:
             send_token()
        ENDIF
    ELSE:
        numOfTokens += 1
        totalTokenSent -= 1
        send_token(tokenQueue)
    ENDIF
ENDFUNCTION
```

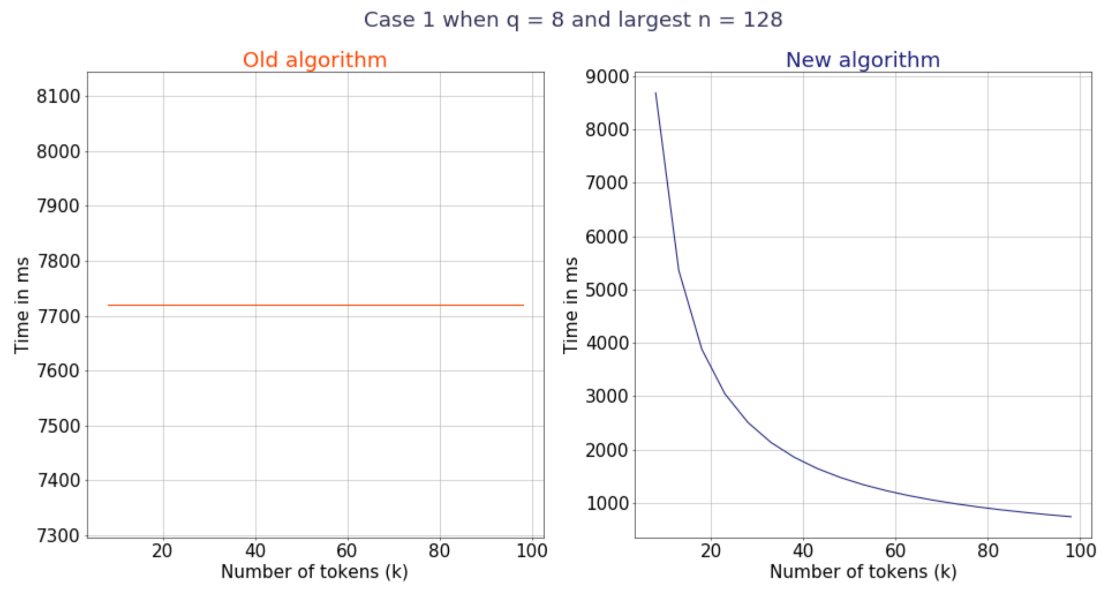

Fig. 4. [Case 1] Number of Tokens vs Time.

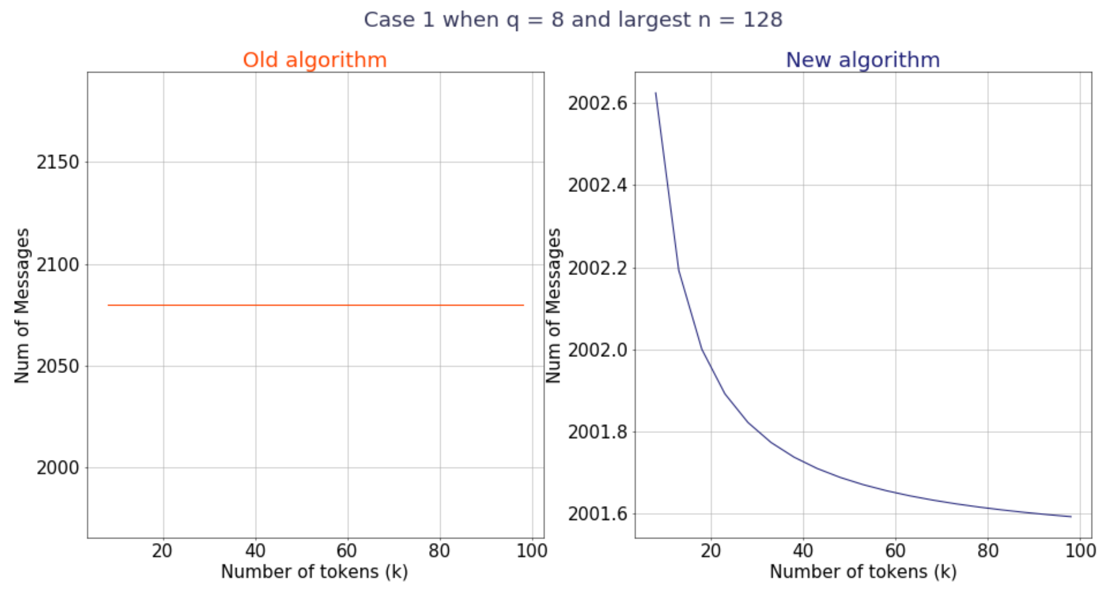

Fig. 5. [Case 1] Number of Tokens vs Number of Messages.

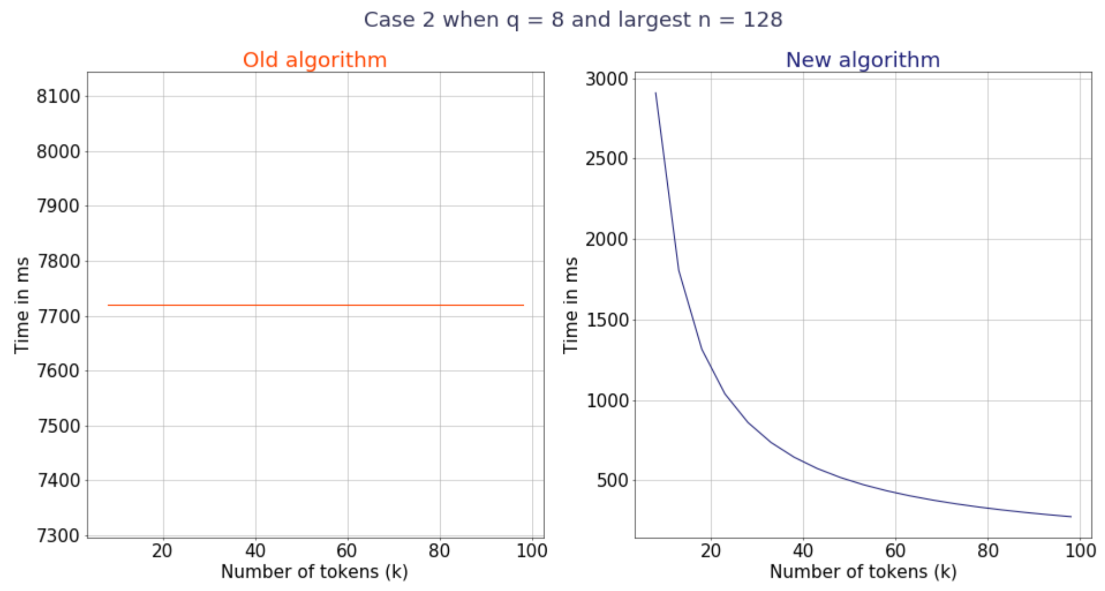

Fig. 6. [Case 2] Number of Tokens vs Time.

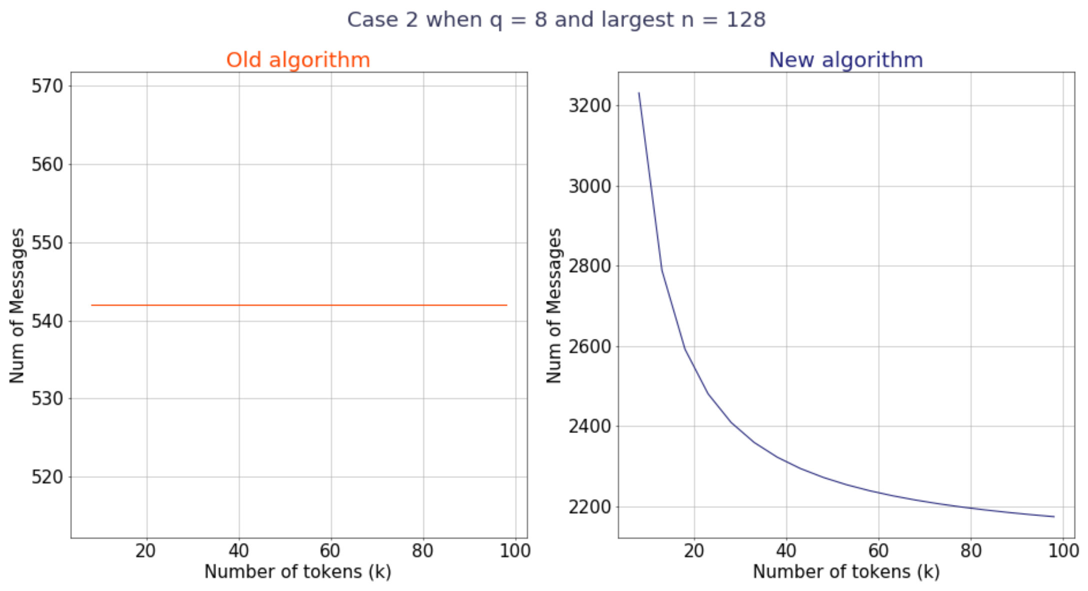

Fig. 7. [Case 2] Number of Tokens vs Number of Messages.

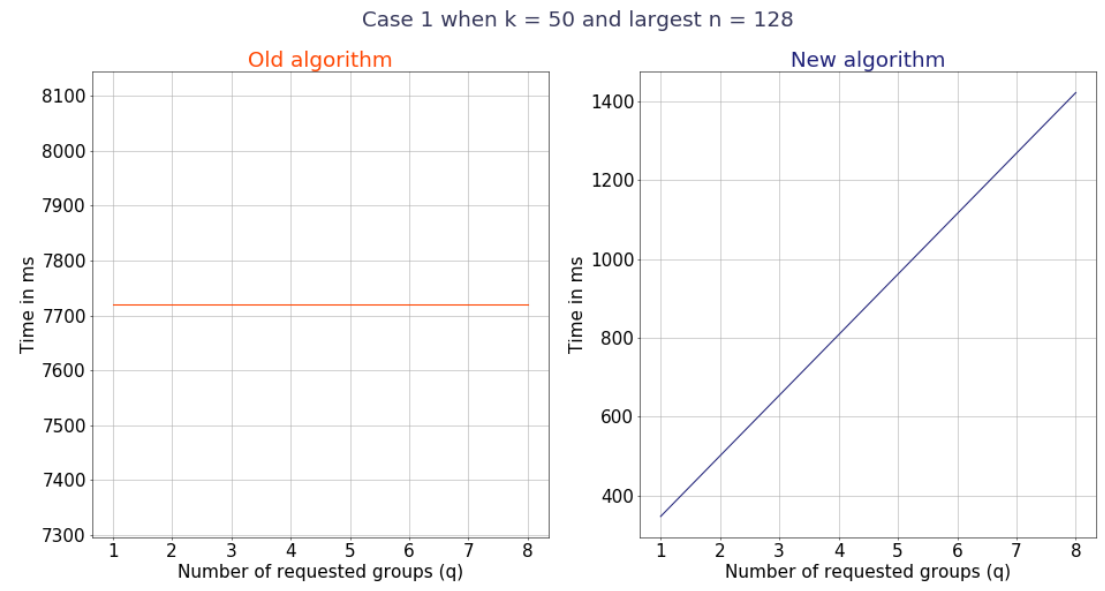

Fig. 8. [Case 1] Number of Requested Groups vs Time.

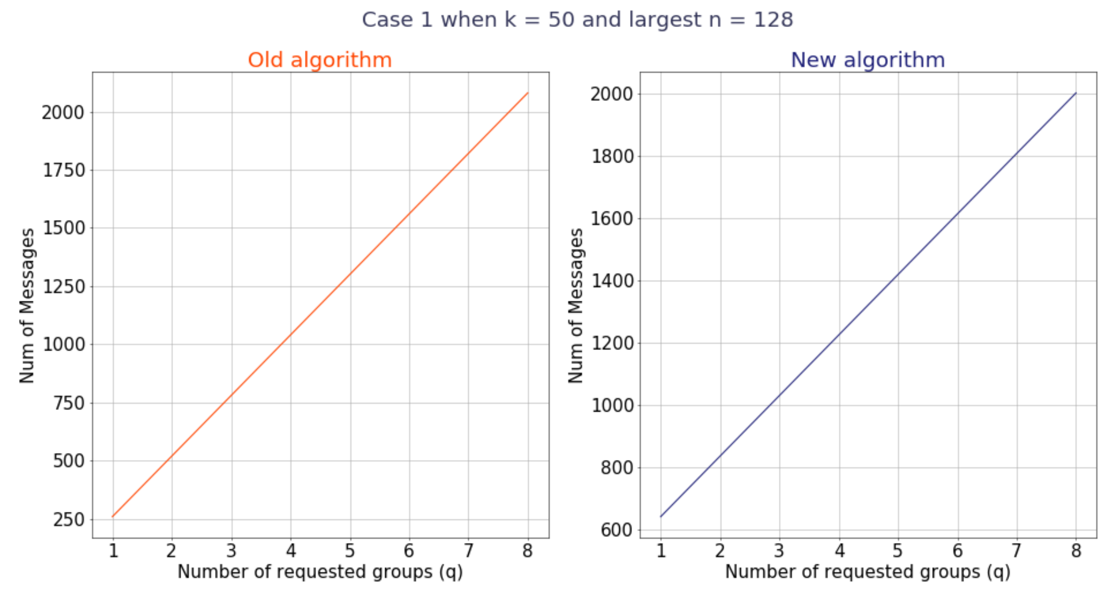

Fig. 9. [Case 1] Number of Requested Groups vs Number of Messages.

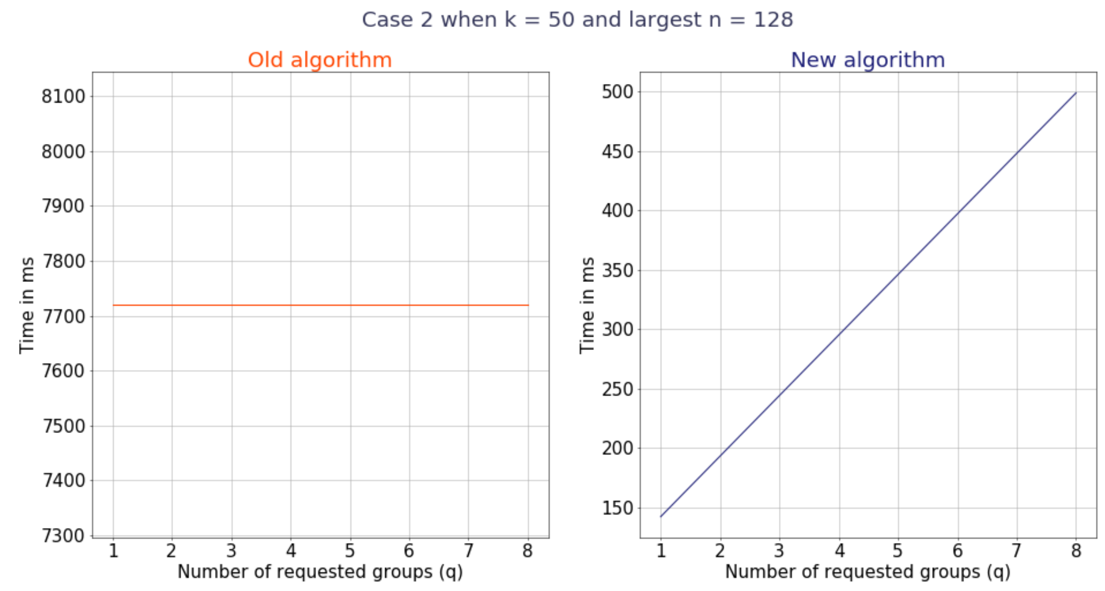

Fig. 10. [Case 2] Number of Requested Groups vs Time.

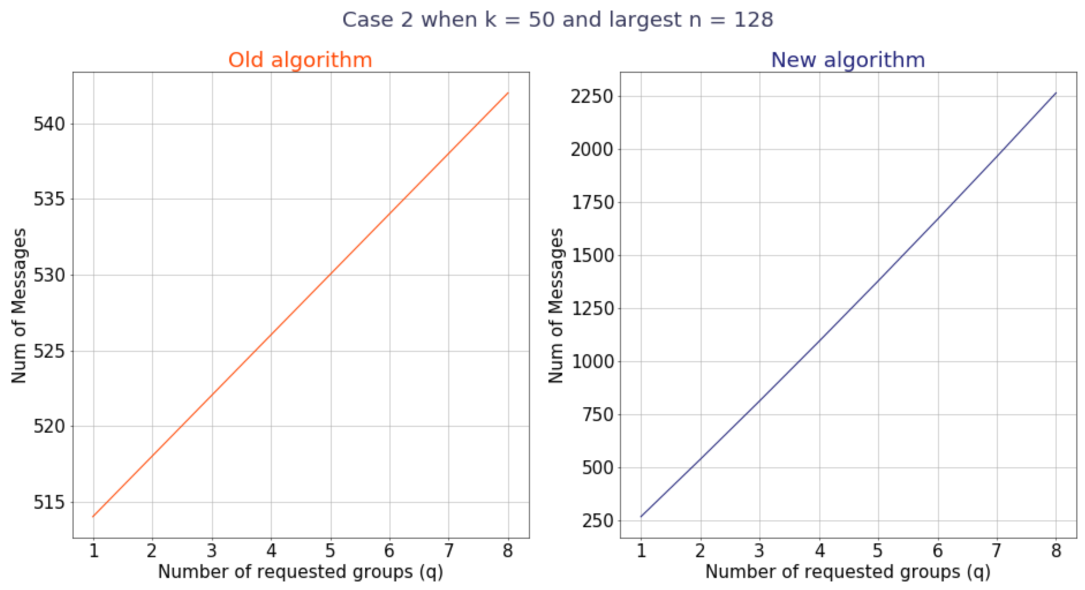

Fig. 11. [Case 2] Number of Requested Groups vs Number of Messages.

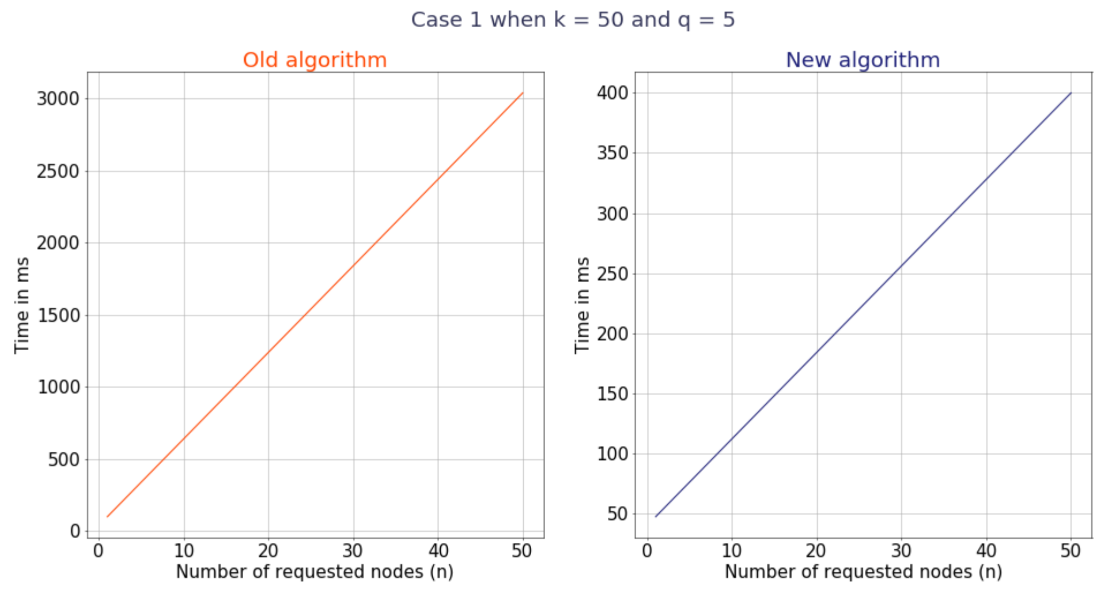

Fig. 12. [Case 1] Number of Requested Nodes vs Time.

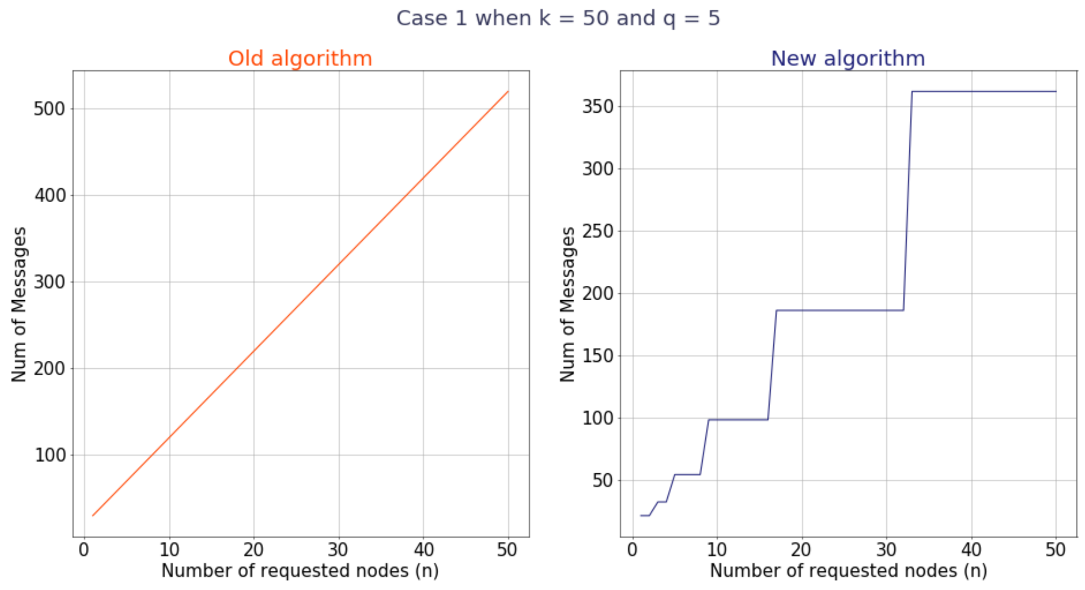

Fig. 13. [Case 1] Number of Requested Nodes vs Number of Messages.

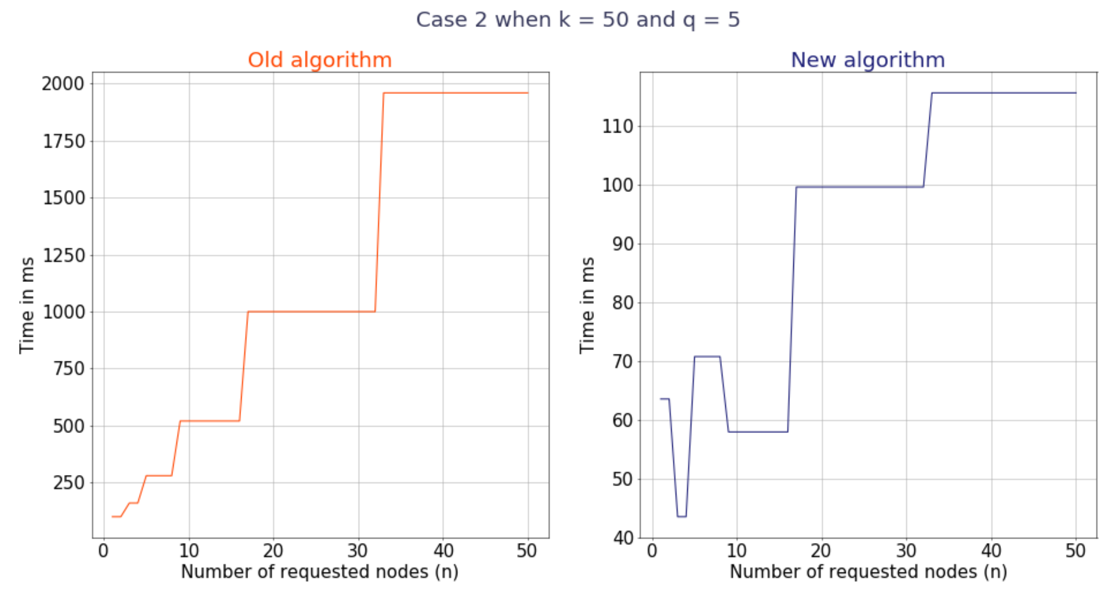

Fig. 14. [Case 2] Number of Requested Nodes vs Time.

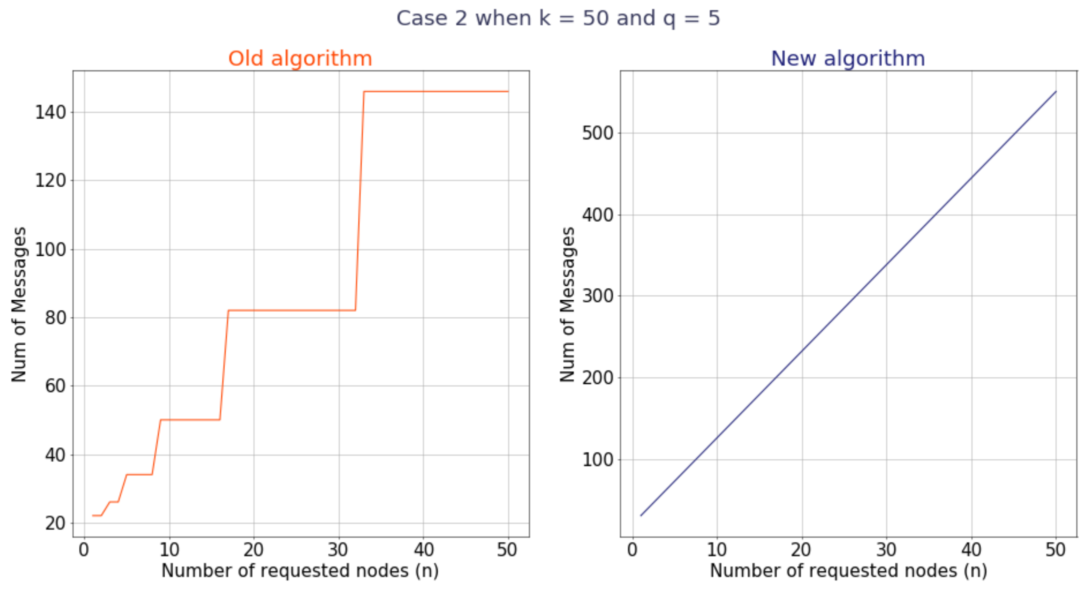

Fig. 15. [Case 2] Number of Requested Nodes vs Number of Messages.