Deadlock-Free Distributed Relaxed Mutual-Exclusion Without Revoke-Messages

Sukhamay Kundu

Computer Science Dept, Louisiana State University
Baton Rouge, LA 70803, USA
kundu@csc.lsu.edu

Abstract. The revoke mechanism in generalized relaxed distributed mutual exclusion algorithm GRME [1] for eliminating a potential deadlock can cause extensive unnecessary revoke actions by the nodes which fail to receive the required granted-replies to their resource requests within a certain predefined time limit. It may happen that there is actually no deadlock present or that only a few nodes need to revoke some of their resource requests to eliminate the deadlock. We show that if the interference graph G is triangle-free (no three nodes are mutually adjacent), then we can choose the request-sets R_i in GRME in such a way that deadlocks are prevented altogether and there is no need to use revoke-messages, while keeping the resource-use decisions fully distributed and allowing non-interfering nodes to use the resource simultaneously.

1. Introduction

The distributed Generalized Relaxed Mutual Exclusion algorithm (GRME) was introduced in [1] to address the dynamic frequency allocation problem to the base stations in a mobile telephone system. Here, each frequency is a separate shared resource for which the base stations compete with each other, and thus there is one mutual exclusion problem for each frequency. A frequency can be shared by two non-interfering base stations, i.e., when they are physically separated by a certain minimum distance. Since there is a limited number of frequencies available to a given wireless service provider and the number of customers far exceeds the number of those frequencies, the sharing of frequencies among the base stations (for their customers) plays a key role in improving the service-performance by reducing the call blocking, which occurs when a base station cannot be assigned a suitable frequency in response to one of its customer's attempt to place a call. (No two customers within a base station cell can share a frequency at any given moment.) The term "relaxed" in GRME refers to the fact that several mutually non-interfering nodes (base stations) can simultaneously use a resource, and the term "generalized" refers to the use of the abstract information structure approach introduced in [6].

In the simplest distributed mutual exclusion algorithm by Ricart and Agrawala [2], a node sends a request-message to all other nodes for permission to use the resource and enters the critical section when it receives all $N-1$ permissions (N being the number of nodes in the network). It uses only $2(N-1)$ messages per resource-access compared to $3(N-1)$ in Lamport's algorithm [3]. The algorithm by Suzuki and Kasami [4] uses $N-1$ request-messages and one granted-message (which is called privilege-message in [4]) per resource access, but the granted-message has N pieces of information (one per node). Here, the node which is currently using the resource determines single-handedly the next node to get the resource and sends that node the granted-message. The granted-message contains for each node i the most recent resource-access number r_i, which is increased to $r_i + 1$ when node i completes a resource-use before it sends granted-message to the next node to access the resource. Maekawa [5] uses a new approach, where a node sends request-message only to a subset of nodes and it accesses the resource when granted permission by each of those nodes; it uses only $O(\sqrt{N})$ messages. In [6], Sanders generalizes Maekawa's idea to an information-structure-based approach. In each of [2-6], at most one node can use the resource at a time, whereas in GRME several non-interfering nodes can access the resource simultaneously.

The resource (frequency) sharing problem is modeled in [1] by an *interference* graph $G = (V(G), E(G))$ whose nodes are the processes (base stations) and whose links (i, j) indicate that process i and process j interfere, i.e., they cannot access the resource at the same time (mutual exclusion). Two nodes i and j can use the resource simultaneously if and only if $(i, j) \notin G$. In GRME algorithm [1], which generalizes Sanders' algorithm [6], each node i sends a request-message to a subset of nodes R_i, and when each node $j \in R_i$ grants the request to i, node i can access the resource. Likewise, when node i is done with resource, it informs a set of nodes $I_i \subseteq R_i$. The behavior of GRME is completely determined by the choice of the sets R_i and I_i, except for the variation of message arrivals due to arbitrary transmission delays. We first review GRME algorithm and show that we must have $I_i = R_i$ for all i in order for GRME to be a meaningful solution to the mutual exclusion problem.

The initial GRME algorithm, which can easily give rise to deadlocks, is modified further in [1] using an abort mechanism that allows a node to withdraw all its pending resource requests using a revoke-message if it does not get the resource within a predetermined time. The later is simply interpreted as a deadlock (there being no explicit test for deadlock detection) although there may not be an actual deadlock. If there is an actual deadlock, it may happen that all nodes involved in the deadlock will exercise the abort mechanism although it might be possible to eliminate the deadlock by aborting only some of the resource requests of one or a few deadlocked nodes. The approach used in [1] has therefore the potential of a significant and unnecessary increase in message load, and indeed the system remains vulnerable to the same deadlock once again. We address the deadlock avoidance problems by focusing on the appropriate selection of the sets R_i so that no deadlock can occur and hence no need to use the revoke-messages. We show this can be done for triangle-free interference graphs G, while keeping the workload fairly distributed among all nodes.

2. Review of GRME Algorithm

Fig. 1 shows a finite-state model (FSM) of the algorithm GRME in [1], where the algorithm was presented simply as a set of rules. The finite-state model shows more clearly the actions available at a node i at various stages. Each transition in Fig. 1 is labeled with a "condition/action" pair. If node i is currently in state σ, then each transition in state σ whose condition-part is satisfied becomes *enabled* and one of the enabled transitions is selected at random for execution. The action-part of the selected transition is then carried out and the head of the transition becomes the new state of node i. The empty-condition, denoted by "−", means that it is always satisfied; the empty-action, also denoted by "−", means no action is performed. Each node i in the interference graph G acts according to this FSM. Table 1 shows the three kinds of messages (M1)-(M3) used in GRME for coordinating the resource allocation to achieve mutual exclusion. The list L_i at node i contains the messages at i that are yet to be processed (or need further processing). The FSM in Fig. 1 is non-deterministic because at each state $\sigma \neq$ start-state one may have several transitions enabled at a given moment. We also do not specify any particular order in which messages are to be selected from L_i for processing. We write $R'_i = \{j\colon i \in R_j\}$, the nodes which send request-message to node i.

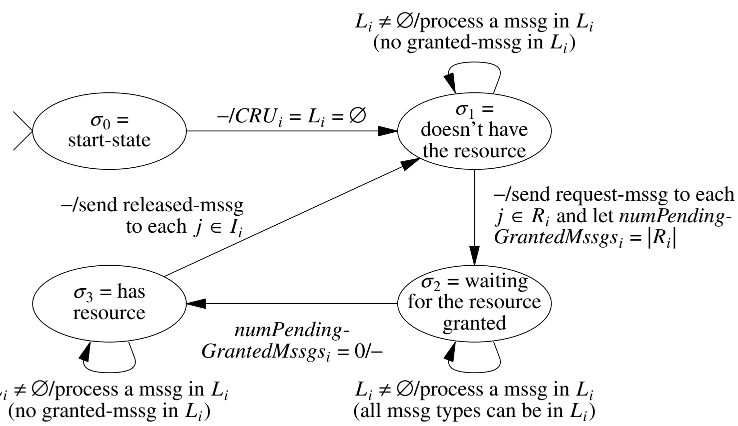


Figure 1. The finite-state machine for node i in algorithm GRME.

Table 1. Messages use in GRME algorithm.

(M1)	Resource *request* message: A node i sends request(i, j) message to a each node j in a fixed set R_i = request-set(i) when it currently does not have the resource and wishes to acquire it.
(M2)	Resource *granted* message: A node $j \in R_i$ sends granted(j, i) message to node i as a response to a request(i, j) message from i if certain conditions hold.
(M3)	Resource *released* message: A node i sends released(i, j) message to each node j in a fixed set I_i = inform-set(i) when it is done with the current use of the resource. The released-messages to nodes in I_i as a whole can be thought of as the response of node i to the set of granted-messages it received (from nodes in R_i) as a whole.

It is convenient to define the digraph G_R which has the same nodes as G and the arcs $\{(i, j): j \in R_i, 1 \le i \le |V(G)|\}$. Thus, R_i is the set of nodes that are *adjacent from* node i and R'_i is the set of nodes that are *adjacent to* node i. As will be seen, the behavior of GRME in regard to mutual exclusion and deadlock is completely determined by G_R (in spite of its non-deterministic nature) when $R_i = I_i$ for all i, which is the case of primary interest.

2.1. Message Processing

We denote a message from i to j by mssg(i, j), where mssg = request, granted, or released. Each node i maintains, in addition to its message-list L_i, the set CRU_i of nodes j to which it has sent the message granted(i, j) and from which it has not received the corresponding message released(j, i). Thus, $j \in CRU_i$ means that node j is either currently holding the resource or it is waiting to receive granted-message from some other nodes or it may have released the resource but i does not know about it (i.e., the released(j, i)-message has not been processed at i); it may even be the case that i will never know about it because $i \notin I_j$. In particular, $CRU_i \subseteq R'_i$.

(P1) Processing of request(i, j) at j: If $i \notin CRU_j$ and there is no node k such that $(i, k) \in G$ and $k \in CRU_j$, then add i to CRU_j, send granted(j, i) to i, and remove request(i, j) from L_j. (Otherwise, the message request(i, j) is put back in L_j.)

(P2) Processing of released(i, j) at j: $CRU_j = CRU_j - \{i\}$ and remove released(i, j) from L_j. (If we were to keep the request(i, j) that was postponed in (P1) in a separate queue D_j at node j, then one could go through them and see if any of them can be granted now and in that case we would remove that request from D_j and add i to CRU_j.)

(P3) Processing of granted(i, j) at j: Decrement $numPendingGrantedMssgs_j$ (which is initialized to $|R_j$ in the transition from σ_1 to σ_2 in Fig. 1) by 1 and remove granted(i, j) from L_j.

The states σ_1 and σ_3 in Fig. 1 differ only in that node i sends out different messages (to nodes in R_i or I_i, respectively) in the transitions (σ_1, σ_2) and (σ_3, σ_1); none of these transitions changes L_i. The processing of a particular kind of message does not depend on the state, but the content of L_i can depend on the state.

Example 1. Consider the interference graph G in Fig. 2 and let $R_i = \{4\}$ and $I_i = \{2\}$ for each i. On completion of processing the messages in the sequence ⟨request(1,4), request(3,4), granted(4,1), granted(4,3)⟩, nodes 1 and 3 will have the resource simultaneously and we have $CRU_4 = \{1, 3\}$, $L_1 = L_3 = L_4 = \emptyset$, and $numPendingGrantedMssgs_1 = numPendingGrantedMssgs_3 = 0$. (In contrast, for any message-sequence we have $CRU_1 = CRU_2 = CRU_3 = \emptyset$, with L_2 having zero or more messages.) Node 4 cannot now grant future request for resource from nodes 1 and 3 and thus nodes 1 and 3 cannot access the resource again. Also, a request from node 2 or from node 4 for the resource will not be granted by node 4, even after both nodes 1 and 3 have sent released-message to node 2. However, had we chosen $I_1 = \{2, 4\} \supseteq R_1$, say, only node 1 can access the resource again and again, and the same is true for node 4, although both nodes 1 and 4 cannot have the resource at the same time. Note that we can remove 2 from an I_i without having any impact on the accessibility of the resource for that node or any other node.

Figure 2. A simple interference graph G (shown on the left) and its G_R (shown on the right) where $R_i = \{4\}$ for each i.

2.2. Analysis

A node i cannot send its $(n+1)$th request-message, $n > 0$, until its nth request was granted by every one in R_i and node i has subsequently sent released-message to every node in I_i when it finished the nth use of the resource. It follows that L_j can contain at most one copy of the message request(i, j) because the previous copies of request(i, j) were removed from L_j when node j issued the corresponding granted(j, i). Likewise, the list L_i can contain at most one copy of granted(j, i) for a given j. In regard to release-message, if $j \in I_i - R_i$, then i never enters the set CRU_j and L_j may contain multiple copies of released(i, j). Other than the multiple copies of released(i, j) in L_j, the removal of j from I_i does not prevent or provide access to the resource for node i or any other node for any message sequence. Thus, we can assume (as in [1]) that $I_i \subseteq R_i$ for all i. In that case, there can be at most one released(i, j) in L_j for $j \in I_i$ because for the case $I_i \subset R_i$ node i cannot even get access to the resource more than once and for the case $I_i = R_i$ node i cannot send $(n+1)$th released-message until its nth released-message was processed by each $j \in I_i = R_i$. Also, both request(i, j) and released(i, j) cannot occur simultaneously in L_j because that would require $j \in R_i \cap I_i$, but then request(i, j) would have been deleted from L_j as node j sent granted(j, i) before node i could send released(i, j). This suggests the following lemma.

Lemma 1. If $I_i \subseteq R_i$ for all i, then $|L_i| \leq |R'_i| + numPendingGrantedMssgs_i$ ($\leq |R'_i| + |R_i|$).

Proof. Suppose $R'_i = \{j_1, j_2, \cdots, j_n\}$ and currently $CRU_i = \{j_1, j_2, \cdots, j_m\}$, where $0 <= m \leq n$ ($m = 0$ means $CRU_i = \varnothing$). Then, L_i can contain at one most one request(j, i) for $j = j_k$, $m+1 \leq k \leq n$. Let $I'_i = \{j\colon i \in I_j\} \subseteq R'_i$; these are the nodes j who can send released-message to node i. Since only the nodes that have received granted-message can send released-message, there is at most one released(j, i) for $j = j_{k'}$, $1 \leq k' \leq m$. The first term in the bound for $|L_i|$ then gives the number of request(j, i) and released(j, i) messages in L_i. The second term gives the bound for the number of granted(j, i) messages in L_i, with $j \in R_i$. Since R_i and R'_i may be disjoint, the lemma is proved. □

2.3. Two Simplifications

First, we argue that we can assume $(i, i) \notin G$ for each i (in [1], it is assumed that $(i, i) \in G$ for each i). The only role (i, i) can play is in processing request(i, j) at a node $j \in R_i$ if $i \in CRU_j$. This condition can arise only after a request for resource from node i was granted by each node in R_i and i has already released that instance of the resource (and informed the nodes in R_i), but $j \notin I_i$ and node i making the next request for resource. However, by removing (i, i) from G we can only allow more access to resource at node i, without violating mutual exclusion. Note that if $I_i = R_i$, then we cannot have request$(i, j) \in L_j$ and $i \in CRU_j$.

Next, we argue that we can assume $I_i = R_i$ for each i (Theorem 1 in [1], which we will show to be wrong, assumes that $I_i \subseteq R_i$ for each i). We begin with the simple observation that removing a node $j \in I_i - R_i$ does not affect the resource allocation to node i or any other node; this can be seen from Example 1 with $j = 2$, which does not belong to any R_i. Now, suppose $j \in R_i - I_i$ for some i. Then, following the processing of the first request(i, j) at j, we have $i \in CRU_j$ permanently. There are two cases to consider. If we did not eliminate $(i, i) \in G$, then node i will not ever receive granted(j, i) from j and hence will not receive the resource more than once, leading to starvation. On the other hand, if we have eliminated (i, i) from G, then other problems can arise when there is a node k such that $(i, k) \in G$ and $j \in R_k$. In that case, node k cannot receive the granted(j, k) message and hence cannot get access to the resource following the first access to resource by node i, which is again a case of starvation. On the other hand, if there is no such k, then removing j from R_i will not lead to any new cases of failure of mutual exclusion that were not there before. This shows that we can assume $I_i = R_i$.

Lemma 2. A necessary and sufficient condition to assure that each node i can access the resource infinitely often, assuming a first-come-first-serve processing of the messages from each L_i, is $I_i = R_i \neq \emptyset$. □

Henceforth, we assume that $I_i = R_i \neq \emptyset$ for each node i.

2.4. Corrected Necessary and Sufficient Condition

The purpose of Theorem 1 in [1] is only to assure the relaxed mutual exclusion, i.e., two nodes i and j can get simultaneous access to the resource if and only if $(i, j) \notin G$. It is not intended to prevent deadlock or starvation. We can achieve the relaxed mutual exclusion property by simply taking each $R_i = \{1\}$, say, and each $I_i = \emptyset$. If we do not want empty sets for I_i, then we can take each $R_i = V(G)$ and let each I_i be arbitrarily assigned to $\{1\}$ or $\{2\}$, say. But this shows that the condition "$[(i \in I_i \cap R_j) \wedge (j \in I_j \cap R_i)] \vee [I_i \cap I_j \neq \emptyset]$ for each $(i, j) \in G$" in [1]] is not necessary. The corrected necessary and sufficient condition for RME is then given by the following Theorem. The condition given in [1] reduces to this if we assume that $I_i = R_i$ for each node i.

Theorem 1. If $I_i = R_i$ for each node i, then a necessary and sufficient condition for GRME to provide mutual exclusion is $R_i \cap R_j \neq \emptyset$ for $(i, j) \in G$. □

To prevent starvation of node i, node j should not repeatedly process request(i', j), $i' \neq i$, and send granted(j, i') message while there is a pending request(i, j) $\in L_j$. To see this, consider G with three nodes and the links (1, 2) and (2, 3) and $R_i = I_i = \{j\}$ for some j. It may happen that first node 1 is granted the resource, then request(2, j) is not granted because of (1, 2) $\in G$ and subsequently request(3, j) is granted. Now before node 3 releases the resource it may happen that node 1 has released the resource and has its next request(1, j) granted while node 2 is still waiting with its request(2, j) $\in L_j$ pending. In this fashion, as long as one of nodes 1 and 3 has the resource, node 2's request is not granted, leading to its starvation. This can be easily prevented by each node j keeping a separate local count(i) = the number of requests granted to other nodes while a previous request(i, j) $\in L_j$ for each $i \in R'_j$; initially, each count(i) = 0. When a count(i) exceeds a predetermined maximum limit $M_j = |R'_j| - 1$, say, node j does not grant a request to any node that interferes with node i. Moreover, when node j processes a released(i', j)-message, count(i) for each node i which interferes with node i' is increased by one, indicating its higher preference for granting resource by node j. In other words, if there are several request(i, j) messages in L_j, node j will select a random i with the largest count values > M_j (if any, and hence currently having no interfering node granted its request by node j) for granted(j, i) message.

Definition 1. We say that a link $(i, j) \in G$ is *supervised* by a node k if $k \in R_i \cap R_j$, i.e., $\{i, j\} \subseteq R'_k$. The node k then prevents nodes i and j getting access to the resource at the same time. (A given link (i, j) may be supervised by more than one node.)

It is clear that for one successful access-use-release cycle to the resource for node i, it sends $|R_i|$ request-messages, receives that many granted-messages, and finally sends that many released-messages, giving a total of $3|R_i|$ message load. Thus it is important to keep the sets R_i as small as possible satisfying Theorem 1.

Lemma 3. If the sets R_i minimize the total message load, then $R_i \subseteq \bigcup\{R_j: (i, j) \in G\}$ for each i.

Proof. If $k \in R_i - \bigcup\{R_j: (i, j) \in G\}$, then removal of k from R_i cannot destroy the property "$R_i \cap R_j \neq \emptyset$ for $(i, j) \in G$" needed for the mutual exclusion. Thus, we can replace each R_i by $R_i - \bigcup\{R_j: (i, j) \in G\}$, say in the order $i = 1, 2, \cdots, N = |V(G)|$ and repeat the process till none of them changes in a cycle. The finiteness of the various sets implies that the process will terminate (in no more than N^2 cycles). The final values of R_i satisfy the lemma and the intersection property "$R_i \cap R_j \neq \emptyset$ for $(i, j) \in G$" in Theorem 1. □

3. Deadlock Prevention

It is clear that if we select a granted or a released message from L_i at node i for processing, then it can always be processed successfully. Till now we have not imposed any restriction like first-in-first-out for selection of messages in L_i for processing, and this worked fine because the only messages from a node j to i for which node i can influence node j's behavior in terms of node j's access to the resource (or other nodes' access to the resource who depend on granted-message from j) are request(j, i) and released(j, i) and only one of them can be in L_i at any time. We will continue to assume that any message can be selected from L_i for processing at a node i, except for "busy-looping" situation where node i attempts unsuccessfully to process the same request(j, i) message again and again without being able to send granted(i, j) to j, or cycling in that fashion through a set of request-messages for different j.

A deadlock in the present context means that there is a non-empty subset of nodes $D \subseteq V(G)$ which are unable to receive additional granted-message and progress further towards having access to the resource (after having received some granted-messages). The nodes in D may, however, continue to process other messages sent to them, including sending granted-messages to nodes not in D. A minimal message-scenario in a deadlock is therefore one where the only messages present in the system are the unfulfilled (and unprocessable) request-messages from nodes in D. The following definition is a refinement of an observation in [1].

Definition 2. An *alternating cycle* in G_R is a cycle $C = \langle s_1, t_1, s_2, t_2, \cdots, t_m, s_1 \rangle$, where the conditions (1)-(3) below hold. In particular, C has an even number of arcs, which are traversed alternately in the forward direction and the backward direction.

(1) The nodes s_i's are distinct from each other and so are the arcs (s_i, t_i) and (s_{i+1}, t_i); in contrast, the nodes t_j's need not be distinct and, moreover, we may have $s_i = t_j$ for some i and j, including the cases $j = i$ or $i + 1$.

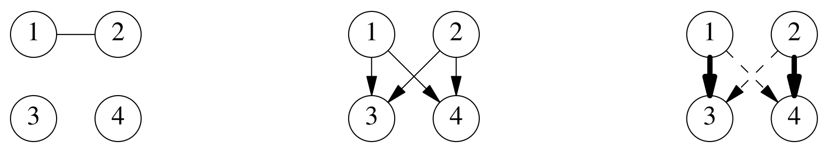
(2) Each arc $(s_i, t_i) \in G_R$ corresponds to a granted request, and

(3) Each arc $(s_{i+1}, t_i) \in G_R$ is a blocked request due to the granted request (s_i, t_i) and $(s_i, s_{i+1}) \in G$ (with $s_{m+1} = s_1$).

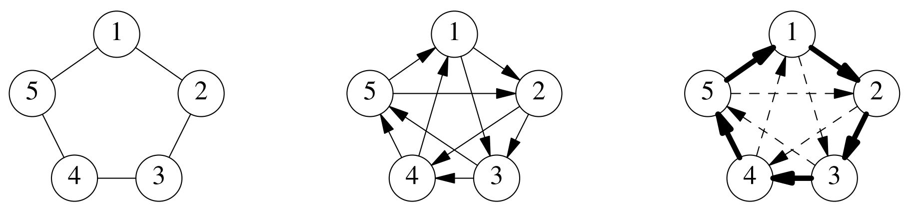
Note that the arcs (s_i, t_i) and (s_{i+1}, t_i) form an Eulerian graph on the nodes in an alternating cycle C in the sense that each node in C has an even $(2p \geq 0)$ number of arcs to it and an even $(2q \geq 0)$ number of arcs from it, with $p + q > 0$ for each node s_i and t_j. This gives a simple method of showing the absence of a possible alternating cycle in G_R without being concerned with which arcs represent granted requests and which arcs represent blocked requests. Each alternating cycle $C = \langle s_1, t_1, s_2, t_2, \cdots, t_m, s_1 \rangle$ in G_R has associated with it the cycle $\langle s_1, s_2, \cdots, s_m, s_1 \rangle$ in G and thus the existence of C depends on both G and the sets R_i, i.e., G_R.

Example 2. Figs. 3(i)-(iii) show an interference graph G, a possible graph G_R, and a deadlock state. Here, node 3 has sent granted(3, 1) to node 1 and node 4 has sent granted(4, 2) to node 2, which now prevent node 3 to send granted(3, 2) and also prevent node 4 to send granted(4, 1). Since neither of nodes 1 and 2 can proceed further, we have a deadlock with $D = \{1, 2\}$. In this case, the deadlock is caused due to the fact that the interference (1, 2) is being supervised by both nodes 3 and 4. A deadlock can occur even if each interference is supervised by exactly one node. This is shown in Figs. 3(iv)-(vi). In each case, we have an alternating cycle C; $C = \langle 1, 3, 2, 4, 1 \rangle$ in Fig. 3(iii) and $C = \langle 1, 2, 5, 1, 4, 5, 3, 4, 2, 3, 1 \rangle$ in Fig. 3(iv). Note that if we remove (5, 1) from G in Fig. 3(iv) and remove the arcs (1, 2) and (5, 2) from G_R

in Fig. 3(v) because node 2 is the supervisor of (5, 1) $\in G$, then (1, 3) $\in G_R$ is not part of an alternating cycle since this is the only arc from node 1 and this in turn means (2, 3) $\in G_R$ is not in part of alternating cycle since this is now the only arc to node 3, and so on. It follows that the reduced $G-(5, 1)$ and its associated reduced $G_R - \{(1, 2), (5, 2)\}$ is deadlock-free. (If we apply Theorem 3 to $G-(5, 1)$, then we get a different G_R which is also deadlock-free; we can also get a deadlock-free family of R_i for the interfernce graph G in Fig. 3(iv) using Theroem 3.)



(i) An interference graph G. (ii) A digraph G_R that can create a deadlock. (iii) A deadlock state.



(iv) A more complex G. (v) G_R; each $(i, j) \in G$ has a unique supervisor. (vi) A deadlock state.

Figure 3. Illustration of deadlock state; solid arcs show requests granted and dashed arcs show requests blocked.

Lemma 4. A given family of sets R_i (= I_i) allows a deadlock in GRME algorithm [1] if and only if there is an alternating cycle in G_R.

Proof. We prove the "only if" part, the "if-part" being trivial. Consider a minimal set of nodes D involved in a deadlock. Each node $s_i \in D$ has therefore at least one of its requests granted that causes at least one request in some other node in D to be blocked, because otherwise we could remove the requests from s_i (i.e., pretend they did not take place), including those that have been granted to s_i, and $D - \{s_i\}$ would still be deadlocked, contradicting the minimality of D. Let t_i be a node that granted a request to s_i and is blocked from granting a request to $s_{i+1} \in D$, $s_{i+1} \neq s_i$. Starting with a node $s_1 \in D$, we can successively traverse the links (s_1, t_1), (t_1, s_2), (s_2, t_2), etc as in an alternating cycle till we have either (t_m, s_{m+1}) where $s_{m+1} = s_i$, $i \leq m$, or we have (s_m, t_m) where $t_m = t_i$ for $i < m$. In either case, we get an alternating cycle involving s_j and t_j, for $i \leq j \leq m$. By minimality of D, it follows that $i = 1$ and $D = \{s_1, s_2, \cdots, s_m\}$. □

Theorem 2. If the sets R_i give a deadlock-free mutual exclusion with GRME and $\pi(x)$ is a permutation of the nodes $V(G)$, then the sets $R_i^{\pi} = \{\pi(j)\colon j \in R_i\}$ also give a deadlock-free mutual exclusion with GRME.

Proof. Immediate from Theorem 1 and Lemma 4. □

Example 3. Figs. 4(i)-(ii) illustrate a simple method for choosing the sets R_i so that the resulting GRME provides mutual exclusion property in a deadlock-free fashion. For each link $(i,\ j) \in G$, we simply let the larger node $\max\{i,\ j\}$ be a supervisor for the link. Note that if G does not contain three nodes which are mutually adjacent to each other, i.e., G is triangle-free, then this gives a unique supervisor for each $(i,\ j) \in G$. For a cycle on the nodes $i < j < k$ in G, the above scheme makes both j and k a supervisor of the link $(i,\ j)$ in G and hence it can create a deadlock due to the alternating cycle $C = \langle i,\ k,\ j,\ j,\ i\rangle$, where $s_1 = i$, $t_1 = k$, and $s_2 = j = t_2$ (cf. Example 2). We argue that no deadlock can occur in Fig. 4(ii). Since (4, 4) is the only arc leaving node 4, the Eulerian property of an alternating cycle C implies that the arc (4, 4) cannot be part of an alternating cycle. Thus, the only way node 4 can be in an alternating cycle C is that it acts in the role of a t_i node implying that the arcs (2, 4) and (3, 4) are in C, but then (2, 3) must be of the form $(s_i,\ s_{i+1})$ and belong to G, which is not the case. It follows that node 4 is not in C, but then clearly $G_R - \{4\}$ has no nonempty Eulerian subgraph and hence there is no alternating cycle in G_R. This example suggests the next definition.

(i) An interference graph G with no 3-cycle.

(ii) G_R; each $(i,\ j) \in G$ has a unique supervisor.

Figure 4. Illustration of a deadlock-free scheme R_i when there are no triangles in G.

Definition 3. Let $R_i = \{j\colon (i,\ j) \in G \text{ and } i < j\}$ if there is no link $(i',\ i) \in G$ for $i' < i$; otherwise, let $R_i = \{j\colon (i,\ j) \in G \text{ and } i < j\} \cup \{i\}$. In particular, $R_i = \varnothing$ if and only if i is an isolated node (is not adjacent to any node). We call this the *max-scheme* for defining the sets R_i or equivalently the digraph G_R, whose arcs are now given by $\{(i,\ j), (j,\ j)\colon (i,\ j) \in G \text{ and } i < j\}$.

Lemma 5. If G is triangle-free and the sets R_i are defined by the max-scheme, then they satisfy the property in Theorem 1 and the property in Lemma 3.

Proof. First, if $(i,\ j) \in G$ and $i < j$, then $j \in R_i \cap R_j$ and hence the sets R_i satisfy Theorem 1. Now we show that the sets R_i satisfy Lemma 3. For an isolated node i, $R_i = \varnothing$ and the union $\bigcup\{R_j\colon (i,\ j) \in G\}$ in Lemma 3 is also $\varnothing$. Now consider a node i which is not isolated and is not adjacent to any node $i' < i$; let the nodes adjacent to i in G be $A(i) = \{j_1,\ j_2,\ \cdots,\ j_m\}$, where $m \geq 1$ and each $j_k > i$. Then, $R_i = \{j_1,\ j_2,\ \cdots,\ j_m\}$ and each $R_{j_k} \supseteq \{j_k\}$, which shows $R_i \subseteq \bigcup\{R_j\colon (i,\ j) \in G\}$. For any other node i, there is at least one node $i' < i$, $i' \in A(i)$; there may be other nodes smaller than i which are in $A(i)$. Let $\{j_1, j_2, \cdots, j_m\}$, $m \geq 0$, be the nodes in $A(i)$ that are larger than i. Then, $R_i = \{i, j_1, j_2, \cdots, j_m\}$ and once again $R_i \subseteq \bigcup\{R_j: (i, j) \in G\}$ because $R_{i'} \supseteq \{i\}$ and $R_{i'}$ is included in the union and each $R_{j_k} \supseteq \{j_k\}$. □

Theorem 3. For each triangle-free interference graph G, the request-sets R_i defined by the max-scheme gives a deadlock-free GRME and degree(i) $\leq |R_i| + |R'_i| \leq 2 +$ degree(i), where degree(i) = the number of links at i in G.

Proof. We may assume that G is a connected graph because otherwise G_R can be constructed separately for each component, with each R_i contained in the component that contains node i. Suppose G has N nodes $\{1, 2, \cdots, N\}$, $N \geq 2$. From Def. 3, it follows that $R'_1 = \emptyset$, $R_N = \{N\}$, $|R_1| + |R'_1| =$ degree(1), and degree(i) $\leq |R_i| + |R'_i| =$ 2 + degree(i) for $i > 1$.

The proof for the deadlock-freeness follows a similar argument as in Example 3 together with the induction on the number of nodes N in G. We first show that node N does not participate in an alternating cycle C. Since there is only one arc (N, N) leaving node N in G_R, this node can participate in C only as a t_i-node. But then the corresponding $(s_i, s_{i+1}) \in G$ and also (s_i, t_i) and $(s_{i+1}, t_i) \in G_R$ because of the max-scheme. Since G is triangle-free, this means $s_i = N = t_i$ and s_{i+1} is a node adjacent to N in G, say, the node k. It follows that there is no t_{i-1} with $(s_i, t_{i-1}) \in G_R$ so that it can be traversed backwards in C and it is distinct from (s_{i+1}, t_i). This shows that N is not part of an alternating cycle. By induction $(G - \{N\})_R = G_R - \{N\}$ has no alternating cycle and hence G_R has no alternating cycle. □

The bounds degree(i) $\leq |R_i| + |R'_i| \leq 2 +$ degree(i) in Theorem 3 shows that the message-load $2(|R_i| + |R'_i|)$ = #(messages sent and received by node i for one access to the resource by each node) for node i is proportional to the number of nodes with which it interferes. In this sense, the scheme used in Theorem 3 is fair. Of course, the choice $R_i = \{N\}$ for each i is also deadlock-free, but it has the disadvantage that node N has a disproportionate message-load compared to the other nodes. If G is a tree (and hence triangle-free), with each node $i \neq N$ adjacent to node N, then the max-scheme gives each $R_i = \{N\}$. In this case, we can reduce $\max(|R_i| + |R'_i|)$ from N+2 to 3 if we interchange the node labels 1 and N. Note that we can combine Theorem 3 with Theorem 2 and get many alternative family of sets $\{R_i: i \in V(G)\}$ to provide a deadlock-free GRME for a triangle-free interference graph G.

4. Conclusion

We have provided a simple sufficient condition "triangle-free property of the interference graph G" and a method for selecting the request-sets R_i to achieve distributed generalized relaxed mutual exclusion, where no deadlock can happen and where the processing load is fairly distributed among the nodes.

References

1. J.J. Jiang, T.-H. Lai, N. Soundarajan, On distributed dynamic channel allocation in mobile cellular networks, IEEE Trans. Parallel and Distributed Systems, **13**

(2002) 1024-1037.
2. G. Ricart and A.K. Agrawala, An optimal algorithm for mutual exclusion in computer networks, Communications of ACM, **24** (1981) 9-17.
3. L. Lamport, Time, clocks, and the ordering of events in a distributed system, Communications of ACM, **7** (1978) 558-565.
4. I. Suzuki and T. Kasami, A distributed mutual exclusion algorithm, ACM Trans. Computer Systems, **3** (1985) 344-349.
5. M. Maekawa, A $\sqrt{N}$ algorithm for mutual exclusion, ACM Trans. on Computer Systems, **3** (1985) 145-159.
6. B.A. Sanders, The information structure of distributed mutual exclusion algorithms, ACM Trans. on Computer Systems, **5** (1987) 284-299.
7. M. Singhal, A taxonomy of distributed mutual exclusion, J. Parallel and distributed Computing, **18** (1993) 94-101.