Condições de conectividade de algoritmos de exclusão mútua em redes dinâmicas

Paulo H. Floriano[1], **Luciana Arantes**[2], **Alfredo Goldman**[1]

[1]Instituto de Matemática e Estatística – Universidade de São Paulo (USP)
R. do Matão, 1010 – São Paulo, SP – Brazil

[2]LIP6 – Université de Paris 6 - CNRS - INRIA
4, Place Jussieu 75005, Paris - França

{floriano,gold}@ime.usp.br, luciana.arantes@lip6.fr

Abstract. *Countless distributed mutual exclusion algorithms exist to control access to a shared resource among network nodes. These algorithms can be divided in two classes: permission and token. However, most of them do not consider network dynamics, which occurs, for example, in MANETs, DTNs and opportunistic networks. In this paper, we wish to determine the necessary and sufficient connectivity conditions in a dynamic network which allow the correct execution of mutual exclusion algorithms in both classes. To that end, we use an approach proposed by Casteigts et al., which explores evolving graphs and graph relabellings in order to study the algorithms created by Ricart and Agrawala (permission) and Helary et al. (token).*

Resumo. *Existem inúmeros algoritmos de exclusão mútua distribuída para controlar o acesso a um recurso compartilhado entre diversos nós. Estes algoritmos podem ser classificados em dois tipos: permissão e token. Entretanto, a maioria destes não considera dinamicidade na rede, como ocorre, por exemplo, em MANETs, DTNs e redes oportunistas. Neste artigo, queremos determinar as condições necessárias e suficientes de conectividade de uma rede dinâmica que permitem a execução correta de algoritmos de exclusão mútua nas duas classes. Para tal, utilizamos uma abordagem proposta por Casteigts et al. que explora os grafos evolutivos e as renomeações de rótulos para estudar os algoritmos de Ricart e Agrawala (permissão) e de Helary et al. (token).*

1. Introdução

A dinamicidade da topologia de redes como MANETs (*Mobile Ad-Hoc Networks*), DTNs (*Delay and Disruption Tolerant Networks*) e Redes Oportunistas (*Opportunistic Networks*) não pode ser representada por meio de grafos estáticos, pois devido às desconexões e à mobilidade dos nós, estas redes são extremamente dinâmicas e as conexões entre os nós variam ao longo do tempo. De fato, nestas redes o caminho entre dois nós deve ser construído ao longo do tempo. Um desafio é então encontrar soluções para lidar com a dinamicidade da rede em algoritmos distribuídos.

Vários modelos foram propostos na literatura para redes dinâmicas [Anta et al. 2010, Mostefaoui et al. 2005, Baldoni et al. 2007, Arantes et al. 2010] com o intuito de caracterizar diferentes aspectos do dinamismo destas redes tais como, por exemplo, a variação constante na topologia da rede e conectividade entre nós, mobilidade dos nós, a falta de visão global que um nó tem da rede, entre outros.

Neste contexto, um arcabouço foi proposto para servir como uma base teórica para o estudo de propriedades fundamentais de algoritmos distribuídos em redes dinâmicas [Casteigts et al. 2010]. Este permite a caracterização das condições necessárias e/ou suficientes que um algoritmo requer da conectividade da rede, explorando os conceitos de grafos evolutivos [Ferreira 2002] e renomeação de rótulos [Litovsky and Sopena 1999].

Um problema muito conhecido na área de algoritmos distribuídos é o da exclusão mútua. Ele consiste em um conjunto de processos que necessita de acesso exclusivo a um recurso compartilhado. Algoritmos para este problema podem ser divididos em duas classes: *permissão* e *token*. Os algoritmos da primeira classe [Lamport 1978, Ricart and Agrawala 1981] adotam o princípio de que um nó tem acesso ao recurso compartilhado somente após ter recebido a permissão de todos os outros nós. Na segunda classe [Suzuki and Kasami 1985, Raymond 1989, Helary et al. 1988], um *token* único circula entre os nós e, ao possuí-lo, um nó tem o direito exclusivo de ter acesso ao recurso. A grande maioria destes algoritmos foram definidos para redes estáticas e apenas alguns deles foram adaptados para suportar dinamicidade.

Neste trabalho, explorando o arcabouço citado, estudaremos as condições necessárias e suficientes para que os algoritmos [Ricart and Agrawala 1981] (*permissão*) e [Helary et al. 1988] (*token*) possam ser executados corretamente em uma rede dinâmica. Mostramos uma formalização dos algoritmos e definimos e demonstramos as condições necessárias e suficientes de conectividade para garantir seu funcionamento correto. Uma primeira versão da formalização do algoritmo de Ricart e Agrawala foi publicada em [Floriano et al. 2011].

Vale ressaltar que vários algoritmos de exclusão mútua distribuída para redes dinâmicas foram propostos em alguns trabalhos anteriormente, especialmente MANETs (*Mobile Ad-hoc Networks*) [Baldoni et al. 2002], [Walter et al. 2001] e [Badrinath et al. 1994]. Entretanto, estes algoritmos consideram a rede como um grafo fortemente conexo no qual os caminhos são construídos estaticamente. O objetivo destes trabalhos difere dos nossos, visto que eles adaptam algoritmos de exclusão mútua já existentes para a dinamicidade da rede, enquanto nós formalizamos as condições de conectividade que asseguram o funcionamento correto dos algoritmos existentes em redes dinâmicas.

O restante deste texto está organizado da seguinte maneira: A Seção 2 apresenta o modelo de grafos evolutivos e explica o arcabouço utilizado na formalização de algoritmos distribuídos e a Seção 3 detalha a formalização dos algoritmos de Ricart e Agrawala e de Helary et al. A Seção 4 mostra as conclusões. Por falta de espaço, não apresentaremos neste artigo uma revisão bibliográfica, mas que pode ser encontrada em [Floriano et al. 2012].

2. Modelo

Consideramos um sistema distribuído dinâmico composto de um conjunto finito de N nós móveis, conhecido por todos os nós e que não há falhas nem perda de mensagens. Toda vez que a rede é inicializada com um algoritmo, assumimos que ela roda por tempo infinito. Além disso, assumimos também que cada nó v tem um identificador único $id(v)$ e conhece os identificadores de todos os outros nós. Consideramos também que todo nó possui um relógio lógico [Lamport 1978], um mecanismo que se utiliza de um contador

em cada processo para determinar uma ordem parcial dos eventos na rede. Como também existem identificadores únicos nos processos, é possível criar uma ordem total entre os eventos.

2.1. Grafos Evolutivos

Modelaremos a dinamicidade das redes por meio de um grafo evolutivo [Ferreira 2002] $\mathcal{G} = (G_0, G_1, \ldots, G_T)$, em que T é um número natural e $G_i = (V(\mathcal{G}), E(G_i))$ é um grafo com conjunto de nós $V(\mathcal{G})$ (comum a todos os G_i) e conjunto de arestas $E(G_i)$, para todo i entre 0 e T. O número T representa o maior instante de tempo em que a rede está modelada. Dizemos que um instante de tempo t está no alcance de $\mathcal{G}$ se $0 \leq t \leq T$. Para representar uma rede até o infinito, simplesmente dizemos que $T = \infty$ e, neste caso, todo número natural é um instante de tempo no alcance de $\mathcal{G}$. Dizemos que G_i é o grafo que modela a rede no instante i. Desta forma, a mobilidade dos nós e dinamicidade das conexões é completamente modelada, pois se dois nós u e v estão em distância de comunicação no instante i, existe uma aresta (u, v) em $E(G_i)$. Neste caso, dizemos que u e v são vizinhos no instante i.

Um conceito importante é a jornada entre dois nós u e v ($\mathcal{J}_{u,v}$), que representa um caminho ao longo do tempo de u até v no grafo evolutivo $\mathcal{G}$. Definimos uma jornada como $\mathcal{J} = (v_0 \overset{t_1}{\rightarrow} v_1 \overset{t_2}{\rightarrow} v_2 \ldots v_{r-1} \overset{t_r}{\rightarrow} v_r)$, em que r é um número natural, $v_0, v_1, \ldots, v_r$ são vértices distintos em $V(\mathcal{G})$, $t_1, t_2, \ldots, t_r$ são instantes de tempo no alcance de $\mathcal{G}$ tal que $t_1 \leq t_2 \leq \ldots \leq t_r$ e (v_i, v_{i+1}) é uma aresta em $E(G_i)$ para todo i entre 0 e $r-1$. Denotamos por $\mathcal{J}_{u,v,t}$ uma jornada de u para v tal que o instante de percurso da primeira aresta t_1 é igual a t. Se, para dois nós $u, v \in V(\mathcal{G})$ e para todo instante de tempo t no alcance de $\mathcal{G}$, existe um segundo instante $t' > t$ tal que existe uma jornada $\mathcal{J}_{u,v,t'}$, dizemos que u e v estão *conectados a termo*. Definimos uma componente conexa no instante i como o conjunto $W \in V(\mathcal{G})$ tal que para cada par de nós u e v que pertencem a W, existe um caminho estático entre u e v em G_i.

2.2. Arcabouço para Formalização de Algoritmos Distribuídos

Existem diversos modelos de comunicação que podem ser utilizados para expressar algoritmos distribuídos, como, por exemplo, troca de mensagens ou memória compartilhada. Entretanto, ao escrever um algoritmo com um paradigma específico, estamos limitando seu escopo de funcionamento. Com este problema em vista, [Casteigts et al. 2010] propuseram um arcabouço que utiliza renomeações de rótulo e grafos evolutivos para caracterizar, respectivamente, a comunicação entre os nós e a dinamicidade da topologia da rede.

Neste modelo, os algoritmos distribuídos são descritos a partir de um conjunto de interações locais simples que é independente da maneira com que os nós efetivamente se comunicam. Cada nó e aresta possui um estado local, chamado de rótulo, que representa o estado algorítmico da rede e só pode ser alterado por meio de uma interação local, denominada regra de renomeação.

Uma regra de renomeação é composta por *pré-condições*, que determinam o estado local da rede (referente a rótulos de nós e arestas) necessário para aquela regra, e *ações*, que determinam o estado local da rede após a execução da regra. Dividimos as regras de renomeação em *regras de comunicação*, que dependem de dois ou mais nós, e

regras monádicas, executadas em apenas um nó quando seu rótulo corresponde às pré-condições. Portanto, a ação de uma regra de comunicação em um nó só pode ser executada em um certo instante de tempo se este nó estiver conectado a um ou mais nós neste instante (o número e a topologia das conexões necessárias depende das pré-condições da regra), o que não é necessário no caso de uma regra monádica.

Considere que o sistema é representado pelo grafo evolutivo $\mathcal{G} = (G_0, G_1, \ldots, G_T)$. No instante de tempo i, a rede é representada pelo grafo $G_i = (V(\mathcal{G}), E(G_i))$. Seja Σ um alfabeto e seja $\lambda_i : V(\mathcal{G}) \cup E(G_i) \rightarrow \Sigma^*$ uma função que associa cada nó e aresta do grafo a uma palavra do alfabeto Σ, denominada rótulo. Então, no instante i, o estado do vértice v de $V(\mathcal{G})$ é representado por $\lambda_i(v)$ e o estado da aresta e de $E(G_i)$ é dado por $\lambda_i(e)$. O grafo rotulado no instante i é dado pelo par (G_i, λ_i). Um algoritmo sobre uma rede é representado por uma tripla $(\Sigma, \mathcal{I}, Z)$, em que $\mathcal{I}$ é o conjunto de estados iniciais e Z é um conjunto de regras de renomeação.

Num grafo evolutivo, as conexões entre nós são alteradas a cada instante de tempo. Chamamos essas alterações de eventos topológicos e chamaremos de $\mathcal{T}_i$ o evento topológico que ocorre entre os instantes de tempo $i-1$ e i. Logo, o grafo G_i representa a rede logo após o evento $\mathcal{T}_i$. Um algoritmo pode realizar diversas regras de renomeação em um dado instante de tempo. Seja $\mathcal{A}$ um algoritmo, chamamos de $\mathcal{A}_i$ a sequência de renomeações de rótulo induzida por $\mathcal{A}$ na rede no instante i. Se (G_i, λ_i) é o grafo rotulado no instante i antes de $\mathcal{A}_i$, então, chamamos de (G_i, λ_i') o mesmo grafo após a sequência de renomeações $\mathcal{A}_i$. Logo, a sequência de eventos em uma rede dinâmica pode ser vista da seguinte maneira: $(G_0, \lambda_0) \xrightarrow{\mathcal{A}_0} (G_0, \lambda_0') \xrightarrow{\mathcal{T}_1} (G_1, \lambda_1) \xrightarrow{\mathcal{A}_1} (G_1, \lambda_1') \xrightarrow{\mathcal{T}_2} (G_2, \lambda_2) \ldots$

O formalismo acima oferece um meio de representar tanto a dinamicidade da topologia da rede quanto as interações entre nós e mudanças de estado. Portanto, podemos utilizá-lo para buscar as condições necessárias e suficientes referentes à dinâmica da rede para o funcionamento dos algoritmos. Para isto, definimos o objetivo de um algoritmo $\mathcal{O}_\mathcal{A}$ como o estado $\mathcal{P}$ da rede que deve ser alcançado ou mantido a partir de um certo instante de tempo. Se um estado deve ser alcançado, o objetivo é que exista um instante de tempo i tal que a propriedade desejada valha em (G_i, λ_i). Se o estado deve ser mantido em todos os momentos, o objetivo é que, para todo instante de tempo i, a propriedade valha em (G_i, λ_i). Portanto, denotamos:

- $\mathcal{O}_\mathcal{A} = \exists i, \mathcal{P}(G_i, \lambda_i)$: quando um estado $\mathcal{P}$ deve ser alcançado;
- $\mathcal{O}_\mathcal{A} = \forall i, \mathcal{P}(G_i, \lambda_i)$: quando um estado $\mathcal{P}$ deve ser mantido todo o tempo.

Dizemos que uma condição C (sobre a topologia da rede) é necessária para o algoritmo $\mathcal{A}$ se e somente se para todo grafo evolutivo $\mathcal{G}$, se C não vale em $\mathcal{G}$, então não é possível alcançar o objetivo $\mathcal{O}_\mathcal{A}$. Simetricamente, a condição C é suficiente para o algoritmo $\mathcal{A}$ se e somente se para todo grafo evolutivo $\mathcal{G}$, se C vale em $\mathcal{G}$, então o objetivo $\mathcal{O}_\mathcal{A}$ sempre será alcançado. É importante ressaltar que, de acordo com [Casteigts et al. 2010], a prova de uma condição suficiente só é possível se fizermos alguma suposição que garanta que as regras de renomeação são executadas corretamente.

3. Algoritmos para o problema da Exclusão Mútua Distribuída

Algoritmos para exclusão mútua distribuída devem garantir as duas seguintes propriedades:

- Segurança (*safety*): a todo instante, no máximo um nó está na seção crítica;

- Vivacidade (*liveness*): um processo que requisita acesso à seção crítica o obtém em tempo finito.

Em geral, algoritmos para exclusão mútua distribuída são divididos em duas classes: *permissão* e *token*. Em algoritmos baseados em permissão, para conseguir o recurso, um nó precisa da permissão de todos os outros nós da rede. Em algoritmos baseados em *token*, uma mensagem especial circula pela rede e o nó que a detém possui acesso exclusivo ao recurso.

Em ambos os algoritmos estudados, todos os nós inicialmente estão no estado *ocioso* e, quando precisam entrar na seção crítica, executam o protocolo de entrada. No algoritmo de Helary et al., um dos nós começa com o *token*. No protocolo de entrada, o nó envia um pedido, marcado com o tempo do relógio do nó, para cada um dos outros nós no sistema. Neste caso, denotamos que o nó se encontra no estado *requisitando*. Para entrar na seção crítica e passar ao estado *utilizando*, os nós devem devem aguardar uma determinada condição. No algoritmo de Ricart e Agrawala, o nó espera uma resposta de cada um dos outros nós, enquanto que no de Helary et al., o nó espera a mensagem de *token*.

Todo processo possui um relógio lógico [Lamport 1978] e toda mensagem de requisição é marcada com o tempo do relógio do nó que a enviou. Antes de enviar uma mensagem, o processo incrementa seu relógio lógico e, ao receber uma mensagem, o atualiza com o máximo entre o tempo da mensagem recebida e o tempo de seu próprio relógio mais 1. Logo, é possível estabelecer uma ordem total entre as requisições, baseada no valor de suas marcas de tempo e, em caso de empate, no identificador único de cada processo. A prioridade entre as mensagens é definida com base nesta ordem total. No caso do algoritmo de Helary et al., as mensagens de *token* também têm uma marca de tempo baseada no relógio lógico, uma vez que cada nó só guarda a mais atual que recebeu.

No algoritmo de Ricart e Agrawala, quando um processo recebe um pedido, ele envia sua permissão se não estiver interessado no recurso (isto é, se o processo está no estado *ocioso*) ou se está requisitando, mas o pedido recebido tem precedência sobre seu próprio; caso contrário, os pedidos recebidos são respondidos durante a execução do protocolo de saída (após o final da seção crítica). Neste momento, o processo também muda para o estado ocioso. Já no algoritmo de Helary et al., os processos que não detém o *token* apenas retransmitem os pedidos e respostas. No protocolo de saída, o nó que estava utilizando o recurso volta a ficar ocioso e marca o tempo lógico do seu último pedido atendido. Então, o nó verifica se há algum pedido pendente, escolhe o de maior prioridade e envia uma resposta com o *token* para o nó que emitiu este pedido.

Ambos os algoritmos já tiveram sua correção provada por seus respectivos autores em seus trabalhos [Ricart and Agrawala 1981, Helary et al. 1988]. Nas seções abaixo, mostramos a formalização dos algoritmos e a definição e demonstração de suas condições necessárias e suficientes de conectividade. Para isso, assumimos que a aplicação que roda nos nós executa o algoritmo de exclusão mútua corretamente, ou seja, primeiro executa o protocolo de entrada, depois a seção crítica e, por fim, o protocolo de saída.

Apresentaremos a seguir as condições necessárias e/ou suficientes para que os dois algoritmos estudados requer da conectividade da rede utilizando o arcabouço foi proposto por [Casteigts et al. 2010]. Por falta de espaço, não apresentaremos neste artigo os requisitos computacionais de ambas as soluções, mas que podem ser encontrada em [].

3.1. Algoritmo de Ricart e Agrawala

Para evitar a necessidade de conexões diretas, cada nó deve guardar e encaminhar todos as requisições e respostas que ele receber, mesmo que não sejam para ele. Deste modo, o rótulo de cada nó incluirá os seguintes conjuntos:

- P: conjunto de requisições recebidas cujos elementos tem a estrutura: (processo que requisitou, marca de tempo da requisição);
- Q: conjunto de pedidos recebidos cujos elementos tem a estrutura: (processo que requisitou, processo que respondeu, marca de tempo da requisição).

Estes conjuntos são disseminados de forma epidêmica pela rede: em qualquer instante de tempo i, os nós de cada componente conexa de G_i atualiza seus conjuntos, fazendo a união deles. Existem diversas maneiras de implementar esta atualização (por exemplo, cada nó trocar informações com todos os outros nós na componente) mas não vamos detalhá-las no algoritmo. Um exemplo de como as componentes são atualizadas pode ser visto na Figura 1. Inicialmente, as listas dos nós possuem conteúdos diferentes, mas, após a sincronização, toda a componente recebe todos os pedidos e respostas.

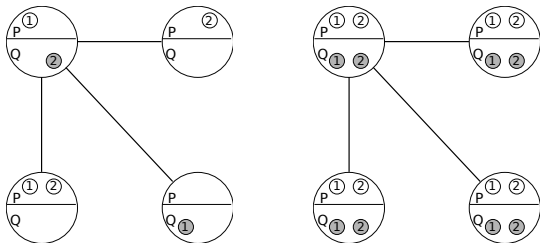

Figura 1. Um exemplo de sincronização de uma componente.

Ao receber uma nova mensagem de requisição, o nó deve decidir se envia ou não uma resposta. Para cada resposta recebida, o processo deve registrar o nó que a enviou para garantir que cada resposta é contada apenas uma vez. Quando o processo termina sua seção crítica, ele executa o protocolo de saída, que consiste em voltar ao estado ocioso e enviar respostas a todas as requisições pendentes.

- Alfabeto: (I, P, Q) (ocioso), (U, P, Q) (utilizando), (R, n, t, A, P, Q) (requisitando, o pedido foi enviado com tempo lógico t, n respostas foram obtidas e todos os nós no conjunto A responderam ao pedido). A notação t_l sempre irá se referir ao tempo lógico do processo no instante em que ele executa a regra em questão. P e Q são, respectivamente, a lista de pedidos e respostas recebidas, como definido anteriormente. Usaremos $*$ para indicar que uma determinada componente do rótulo pode ter qualquer valor válido. Por exemplo, denotamos por $(*, P, Q)$ o rótulo cujo estado pode ser I, R ou U. O tamanho das estruturas que compõem um rótulo pode ser ilimitado, mas, podem ser implementadas de forma a limitar seu tamanho [Floriano et al. 2011].
- Estado inicial: Todo nó começa no estado $(I, \emptyset, \emptyset)$.
- Objetivo: A meta do algoritmo é garantir as propriedades de segurança e vivacidade. Os objetivos sobre o grafo evolutivo rotulado $(\mathcal{G}, \lambda)$ são:
    1. A cada instante de tempo i, não existem dois nós u e v em $V(\mathcal{G})$ tal que $\lambda_i(u) = \lambda_i(v) = (U, P, Q)$ (Segurança).

2. A cada instante de tempo i e para todo nó u em $V(\mathcal{G})$, se $\lambda_i(u) = (R, n, t_l, A, P, Q)$, então existe um instante de tempo $j > i$ tal que $\lambda_j(u) = (U, P, Q)$ (Vivacidade).

- Protocolo de entrada: (regras monádicas)
  1. Pré-condições: $\lambda_i(v) = (I, P, Q)$
     Ações: $\lambda_i(v) = (R, 0, t_l, \emptyset, P \cup \{(v, t_l)\}, Q)$
     (*O nó v muda seu estado para R e adiciona seu próprio pedido a P*).
  2. Pré-condições: $\lambda_i(v) = (R, N-1, t, A, P, Q)$
     Ações: $\lambda_i(v) := (U, P, Q)$
     (*O nó v, que está requisitando, obtém $N-1$ respostas e pode usar o recurso*).
- Protocolo de saída: (regra monádica)
  Pré-condições: $\lambda_i(v) = (U, P, Q)$
  Ações: $\lambda_i(v) = (I, P, Q \cup \{(p, v, t) | (p, t) \in P\})$
  (*O nó v muda seu estado para I e registra em Q que enviou respostas para todos os pedidos pendentes*).
- Regra de comunicação:
  Requisitos topológicos: Uma componente conexa $C \in V(\mathcal{G}_t)$ tal que $\lambda_i(u) = (*, P_u, Q_u)$ para todo $u \in C$
  Ações: Para cada $u \in C$, $\lambda_i(u) = (*, \bigcup_{v \in C} P_v, \bigcup_{v \in C} Q_v)$
  (*Todo nó de toda componente conexa no instante t sincroniza suas listas de pedidos e respostas, não importando em que estado estão*).
- Regras monádicas de atualização de conjuntos:
  1. Pré-condições: $\lambda_i(v) = (I, P, Q) \wedge \exists (p, t) \in P, (p, v, t) \notin Q$
     Ações: $\lambda_i(v) := (I, P, Q \cup (p, v, t))$
     (*O nó v está ocioso e tem uma requisição pendente para a qual ainda não enviou resposta*).
  2. Pré-condições: $\lambda_i(v) = (R, n, t_1, A, P, Q) \wedge \exists (p, t_2) \in P, (p, v, t_2) \notin Q \wedge (t_1 > t_2 \vee (t_1 = t_2 \wedge id(p) < id(v)))$
     Ações: $\lambda_i(v) := (R, n, t_1, A, P, Q \cup (p, v, t_2))$
     (*O pedido de p tem precedência sobre o de v e v ainda não enviou resposta para p*).
  3. Pré-condições: $\lambda_i(v) = (R, n, t, A, P, Q) \wedge \exists p, (v, p, t) \in Q \wedge p \notin A$.
     Ações: $\lambda_i(v) := ((R, n+1, t, A \cup \{p\}), P, Q)$
     (*O nó v recebeu uma resposta de p para seu pedido; v deve registrar que a resposta de p apenas se esta resposta ainda não foi registrada*).

A Figura 2 mostra uma simulação da execução do algoritmo. Os tempos lógicos associados aos pedidos foram omitidos para facilitar a compreensão. Inicialmente, em 2(a) os nós v_1 e v_3 pedem o recurso no instante 0 (a cor cinza clara indica o estado R e os círculos pretos menores representam o pedido). Em 2(b), os nós v_1 e v_2 se comunicam, v_2 recebe o pedido de v_1 e envia uma resposta (círculo pequeno cinza). Em 2(c) v_2 conecta-se com v_3 e o envia uma resposta. v_3 também recebe o pedido de v_1 e envia sua resposta (círculo pequeno branco), pois o pedido de v_1 tem precedência sobre o seu. Em 2(d), v_1 recebe a resposta de v_3 e pode usar o recurso (o estado U é representado pela cor cinza escuro). v_1 também recebe o pedido de v_3, mas ainda não o responde. Em 2(e) v_1 termina de usar o recurso, volta ao estado ocioso e envia sua resposta a v_3, que chega em seu

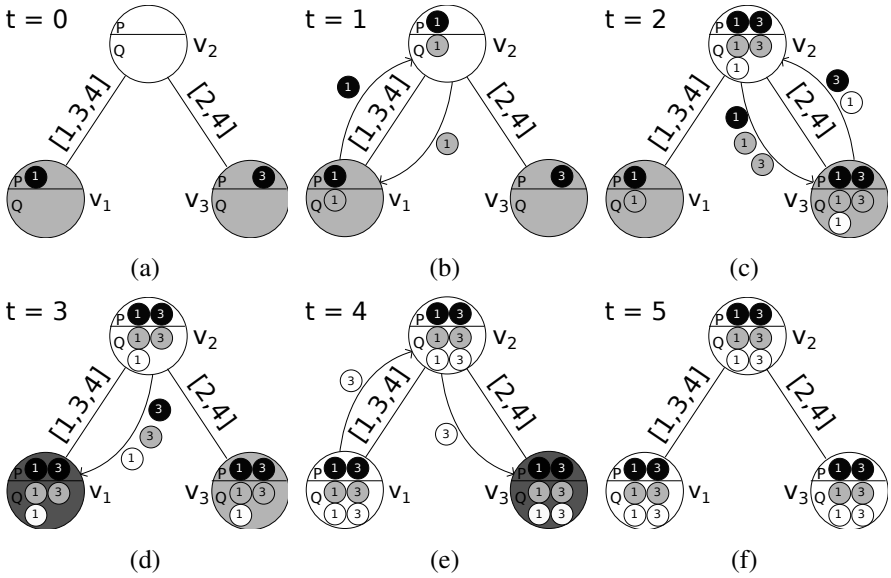

Figura 2. Simulação do algoritmo de Ricart e Agrawala com três nós.

destino no mesmo instante. v_3 agora pode usar o recurso. Finalmente, em 2(f), v_3 libera o recurso e todos os nós estão ociosos novamente.

Condição necessária

Para estabelecer uma condição necessária para este algoritmo, devemos assumir que pelo menos uma requisição é feita na rede ao longo de sua execução. Perceba que sem esta suposição, em casos em que não há pedidos, qualquer topologia de rede garante que os objetivos do algoritmo serão alcançados. Logo, qualquer condição que digamos ser necessária viola a propriedade mencionada anteriormente que diz que uma condição só é necessária se o algoritmo falha para toda topologia em que não vale esta condição.

Proposição 1. *A condição $C_1(\mathcal{G})$ = para todo par de vértices u e v em $V(\mathcal{G})$, existe uma jornada $\mathcal{J}_{u,v}$ e uma jornada $\mathcal{J}_{v,u}$ é uma condição necessária para o algoritmo dado.*

Prova: Vamos provar que para todo grafo evolutivo $\mathcal{G}$, se C_1 não vale em $\mathcal{G}$, então existe um instante de tempo i e um vértice u em $V(\mathcal{G})$ tal que $\lambda_i(u) = (R, n, t, A, P, Q)$ e, para todo instante de tempo $j > i$, $\lambda_j(u) \neq (U, P, Q)$. Ou seja, provaremos que, se existe algum par de vértices tal que não existem jornadas entre estes vértices, a vivacidade do algoritmo é violada nesta rede.

Suponha que existe um nó u em $V(\mathcal{G})$ e um instante de tempo i tal que $\lambda_i(u) = (R, 0, t_l, \emptyset, P, Q)$ e seja v um vértice em $V(\mathcal{G})$. Se não existe $J_{u,v}$, por mais que diversas regras de comunicação sejam executadas ao longo da rede, o pedido de u nunca chegará até v. Analogamente, se não existe $J_{v,u}$, o pedido de u pode até chegar a v, mas a resposta nunca chegará a u. Logo, u nunca conseguirá utilizar o recurso, ou seja, não existe um instante de tempo $j > i$ tal que $\lambda_j(u) = (U, P, Q)$. Portanto, C_1 é uma condição necessária para o algoritmo. ■

Condição suficiente

Hipótese forte da progressão: Neste algoritmo, devemos assumir que, em cada instante, primeiramente, todos os nós realizam uma regra de comunicação com sua componente conexa. Em seguida, os nós realizam todas as regras monádicas possíveis e, de-

pois, a regra de comunicação é realizada novamente. Desta forma garantimos que todas as conexões são consideradas pelo algoritmo.

Proposição 2. *A condição C_2 = para todo par de nós u e v em $V(\mathcal{G})$, u e v estão conectados a termo, acrescida da hipótese forte da progressão, é uma condição suficiente para o algoritmo dado.*

Prova: 1) Segurança é garantida: Vamos provar que em cada instante de tempo, pode haver no máximo um vértice utilizando o recurso. Suponha que, em um dado momento, existem dois nós u e v em $V(\mathcal{G})$ executando a seção crítica (ou seja, no estado (U, P, Q)). Isso significa que ambos os nós receberam pelo menos $N - 1$ respostas, que os permitiu executar a segunda regra do protocolo de entrada. Considere os pedidos enviados por estes dois nós para obter o recurso. Devido a ordem total imposta pelas marcas de tempo dos pedidos e pelos identificadores dos nós, um dos pedidos tem precedência sobre o outro. Suponha que u fez o pedido de menor prioridade. Como u está no estado (U, P, Q), isso quer dizer que este nó recebeu $N - 1$ respostas, incluindo uma de v. Entretanto, o pedido de v tem precedência sobre o de u, ou seja, v não pode ter enviado uma resposta para u, uma vez que um nó só envia uma resposta se está ocioso (primeira regra de atualização de conjunto) ou se o pedido recebido tem maior prioridade (segunda regra de atualização de conjunto). Portanto, chegamos a uma contradição. Logo, não pode haver mais de um nó no estado (U, P, Q) em um dado instante, então, a segurança é garantida.

2) Vivacidade é garantida: Vamos provar que, se a condição C_2 vale na rede $\mathcal{G}$, então todo nó que realiza o protocolo de entrada, em algum momento consegue acesso ao recurso. Suponha que, num dado instante, existe um conjunto $S_R \in V(\mathcal{G})$ de nós requisitando o recurso (no estado (R, n, t_l, A, P, Q)). Devido à condição C_2, existe uma jornada futura de cada nó de S_R para todo outro nó em $V(\mathcal{G})$. Sabemos também que os pedidos enviados pelos nós em S_R são totalmente ordenados. Considere o pedido de maior prioridade segundo esta ordem total. Devido à condição C_2 e à hipótese forte da progressão, o pedido enviado por este nó alcançará todos os outros nós da rede. Como este pedido tem a maior prioridade entre os nós em S_R, todo nó que o recebe envia sua resposta imediatamente, executando a segunda regra de atualização de conjunto se estiver em S_R ou a primeira, se não estiver. Novamente, devido à condição C_2 e à hipótese forte da progressão, as respostas enviadas alcançarão o nó que enviou o pedido de maior prioridade, que, para cada resposta, executa a terceira regra de atualização de conjunto. Portanto, este nó receberá $N - 1$ respostas, então, será capaz de executar a segunda regra do protocolo de entrada e acessar o recurso. Consequentemente, se em um dado instante existe um conjunto de nós requisitando o recurso, aquele que tiver a maior prioridade conseguirá, em algum momento, acessá-lo. Além disso, como existe uma ordem total nos pedidos e, após liberar o recurso, o pedido satisfeito deixa de ser o de maior prioridade, os pedidos de cada nó de S_R irão, em algum momento, passar a ser o pedido de maior prioridade. Portanto, como cada processo que envia um pedido obtém o recurso em algum momento, a propriedade da vivacidade é garantida. ■

3.2. Algoritmo de Helary et al.

Assim como no algoritmo anterior, todos os nós devem guardar uma lista de pedidos. A lista de respostas não é necessária, dado que no máximo uma resposta estará ativa na rede em qualquer momento. Além disso, é preciso guardar os instantes de tempo lógico do último pedido atendido de cada nó, para que se possa determinar quais pedidos ainda não foram atendidos. Logo, o rótulo de cada nó incluirá as seguintes estruturas, que

serão disseminadas pela rede por meio da regra de comunicação da mesma forma que no algoritmo anterior:

- P conjunto de requisições recebidas cujos elementos tem a estrutura: (processo que requisitou, marca de tempo da requisição);
- Q mensagem de *token* com a estrutura: (nó destino, marca de tempo da requisição, marca de tempo do envio da mensagem);
- S vetor indexado pelo identificador dos processos, guardando, para cada nó, o instante de tempo lógico em que seu último pedido foi atendido ou -1 se isso ainda não ocorreu.

- Alfabeto: O estado do nó pode ser (U, T, P, Q, S) (usando e com o *token*), $(R, T|N, t, P, Q, S)$ (requisitando, pode estar com (T) ou sem (N) o *token* e com o pedido enviado no tempo lógico t), $(I, T|N, P, Q, S)$ (ocioso, pode estar com (T) ou sem (N) o *token*). A notação t_l sempre irá se referir ao tempo lógico do processo no instante em que ele executa a regra em questão. Cada nó guarda os conjuntos P, Q e S, como definido anteriormente.
- Definições
    1. Seja u o nó que detém o *token* no instante i. Definimos $X_i = \{(orig, t) \in P_u$ tal que $S_u[orig] < t\}$ o conjunto de pedidos na rede no instante i que ainda não foram satisfeitos. Note que somente os pedidos que chegaram em u estarão em P_u e S_u contém a informação atualizada de quais pedidos já foram atendidos, já que u detém o *token*. Deste modo, só faz sentido computar o conjunto X_i para o nó que detém o *token* no instante i.
    2. Definimos como $maxPri(X_i)$ o pedido de maior prioridade que está em X_i, ou seja, o pedido não atendido que possui menor tempo lógico, desempatando, se necessário, pelo menor identificador de nó.
    3. Definimos $maxTmsg(Q_1, Q_2)$ a mensagem de *token*, dentre Q_1 e Q_2 cujo tempo lógico da resposta é maior.
    4. A união entre dois arrays S_u e S_v $S_u \cup S_v$ é definida como o array S_m, em que $S_m[i] := max(S_u[i], S_v[i])$, para toda posição i.
- Estado inicial: Todos os nós iniciam no estado $(I, N, \emptyset, \emptyset, S[1 \ldots N] = -1)$, exceto o nó (que pode ser qualquer um da rede) que possui o *token* e, portanto, começa no estado $(I, T, \emptyset, \emptyset, S[1 \ldots N] = -1)$.
- Objetivo: A meta do algoritmo é garantir as propriedades de segurança e vivacidade. Os objetivos sobre o grafo evolutivo rotulado $(\mathcal{G}, \lambda)$ são:
    1. A cada instante de tempo i, não existem dois nós u e v em $V(\mathcal{G})$ tal que $\lambda_i(u) = \lambda_i(v) = (U, T, P, Q, S)$ (Segurança).
    2. A cada instante de tempo i e para todo nó u em $V(\mathcal{G})$, se $\lambda_i(u) = (R, N, t, P, Q, S)$, então existe um instante de tempo $j > i$ tal que $\lambda_j(u) = (U, T, P, Q, S)$ (Vivacidade).
- Protocolo de entrada:
    1. pré-condição: $\lambda_i(v) = (I, *, P, Q, S)$
       ação: $\lambda_i(v) := (R, *, t_l, P \cup \{v, t_l\}, Q, S)$
       (O nó muda seu estado para R e emite um pedido)
    2. Pré-condições: $\lambda_i(v) = (R, *, t_p, P, Q = (v, t_p, t_r), S)$
       Ações: $\lambda_i(v) := (U, T, P, Q, S)$
       (O nó recebe o *token* e entra na seção crítica)

- Protocolo de saída: pré-condição: $\lambda_i(v) = (U, T, P, Q = (v, t_p, t_r), S)$
  ação: $\lambda_i(v) := (I, T, P, Q, S[v] := t_p)$
  (O nó sai da seção crítica, volta ao estado I e marca o tempo do seu último pedido em S)
- Regra de comunicação:
  1. Requisitos topológicos: Uma componente conexa W no instante i tal que $\lambda_i(u) = (*, *, P_u, Q_u, S_u)$ para todo $u \in W$
     Ações: Para cada $u \in W$, $\lambda_i(u) = (*, *, \bigcup_{v \in W} P_v, maxTmsg(Q_u, Q_v), \bigcup_{v \in W} S_v)$
     (*Todo nó de toda componente conexa no instante i sincroniza suas listas de pedidos e respostas e seus vetores S, não importando em que estado estão*).
- Regras Monádicas:
  1. Pré-condições: $\lambda_i(v) = (I|R, T, P, Q, S) \wedge X_i \neq \emptyset \wedge (j, t) = maxPri(X_i)$
     Ações: $\lambda_i(v) := (I|R, N, P, Q := (j, t, t_l), S)$
     (O nó que tem o *token* determina qual o pedido de maior prioridade na rede calculando o conjunto X e envia o *token* para o nó que emitiu este pedido)

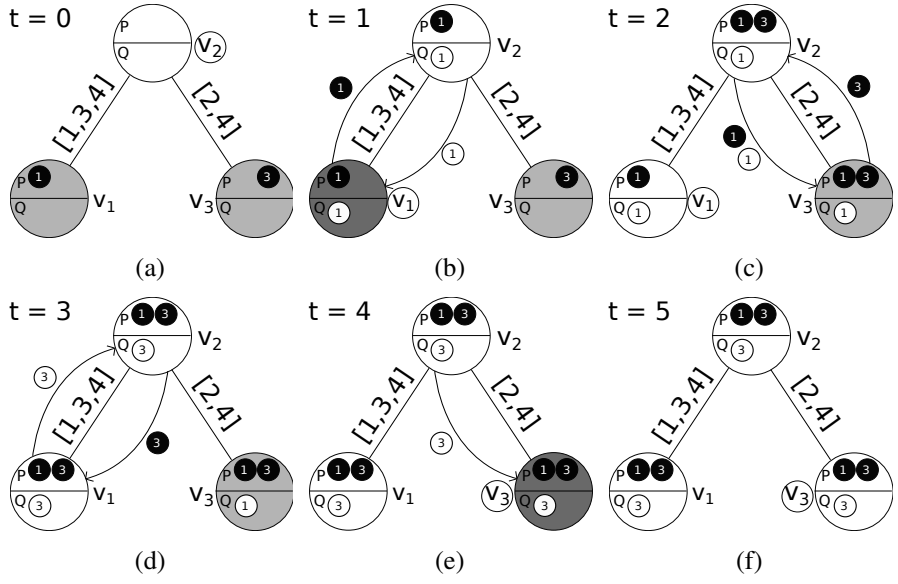

Figura 3. Simulação do algoritmo de Helary et al. com três nós.

A Figura 4 mostra uma simulação do algoritmo de Helary et al. na mesma situação da simulação anterior. Inicialmente, em 3(a), os nós v_1 e v_3 pedem o recurso e v_2 detém o *token* no instante 0. Em 3(b), v_1 e v_2 se comunicam e v_1 recebe uma resposta com o *token* e entra na seção crítica. Em 3(c), v_1 termina de utilizar o recurso e v_2 recebe o pedido de v_3 e envia o pedido e resposta de v_1. Em 3(d), v_1 recebe o pedido de v_3 e envia uma resposta com o *token*. Em 3(e), v_3 recebe a resposta e entra na seção crítica.

Condição Necessária

Não é possível descrever uma condição necessária para este algoritmo. Uma condição necessária é uma condição tal que, se ela não valer em uma determinada rede, nenhuma execução do algoritmo será possível. Mas, para qualquer condição que fixarmos, existe uma execução que funcionará, qualquer que seja a rede, que é o caso em que

apenas o nó que detém o *token* inicialmente faz pedidos para entrar na seção crítica. Por este motivo, vamos adicionar a suposição de que pelo menos dois nós diferentes realizam o protocolo de entrada.

Proposição 3. *Seja* $\mathcal{R}(\mathcal{G}) = \{u \in V(\mathcal{G}) \mid \exists i\ \lambda_i(u) = (R, *, t, P, Q, S)\}$ *o conjunto de todos os nós que requisitam o recurso em algum momento* mais *o nó que detém o* token *inicialmente. Considere a condição* $C_3(\mathcal{G})$ = *Para cada par de nós* u *e* v *em* $\mathcal{R}(\mathcal{G})$*, existe pelo menos uma jornada de* u *para* v *ou uma de* v *para* u*.*

Primeiramente, uma consideração sobre esta condição. Se existir $J_{u,v}$ apenas, existem casos de execução em que o algoritmo funciona. Considere que $\mathcal{R}(\mathcal{G}) = \{u, v, w\}$ e que o *token* começa com o nó u. Este requisita o recurso e o utiliza. O pedido de w chega em u e o *token* é transmitido para w. Não existe mais jornada entre u e w. Em seguida, o pedido de v chega em w e existe uma jornada entre w e v, logo, v também consegue o *token*, mesmo sem existir $J_{v,u}$. As propriedades de segurança e vivacidade foram preservadas.

Prova: Vamos provar que para todo grafo evolutivo $\mathcal{G}$, se C_3 não vale em $\mathcal{G}$, então existe um instante de tempo i e um vértice u em $V(\mathcal{G})$ tal que $\lambda_i(u) = (R, N, t, P, Q, S)$ e, para todo instante de tempo $j > i$, $\lambda_j(u) \neq (U, T, P, Q, S)$. Ou seja, provaremos que se existe algum par de nós em $\mathcal{R}(\mathcal{G})$ que não se conecta por ao menos uma jornada, a propriedade da vivacidade é violada.

Sejam u e v dois nós em $\mathcal{R}(\mathcal{G})$ tal que não existe $J_{u,v}$ nem $J_{v,u}$. Como tanto u quanto v estão em $\mathcal{R}(\mathcal{G})$, ambos requisitam o recurso em algum momento ao longo da execução da rede. Se nenhum dos nós receber o *token* em nenhum momento, então eles não conseguirão entrar na seção crítica e logo, a vivacidade é violada. Considere então, sem perda de generalidade, que o nó u recebe o *token* em algum momento. Se v receber o *token* em algum momento após u ter recebido, isso quer dizer que existe uma jornada de u para v, o que é uma contradição. Portanto, se u entrar na seção crítica, então v nunca entrará. Logo, a vivacidade é violada e, portanto, C_3 é uma condição necessária para o algoritmo. ■

Condição Suficiente

Proposição 4. *Considere a condição* $C_4(\mathcal{G})$ = *Todo par de nós* u *e* v *em* $\mathcal{R}(\mathcal{G})$ *está conectado a termo.* C_4*, acrescida da hipótese forte da progressão, é uma condição suficiente para o algoritmo dado.*

Prova: 1) *Segurança* é garantida: Vamos provar que, a todo momento, ou existe um único nó com o *token* (T) ou existe uma resposta circulando pela rede que ainda não foi recebida por seu destino final.

Inicialmente, um nó está no estado I e com o *token* (T), então a condição vale no início. Ao enviar o *token* em uma mensagem, este nó executa a regra monádica e fica sem o *token* (N). Esta mensagem só pode ser recebida por um único nó e apenas uma vez, pois a regra monádica restringe o nó que recebe a mensagem e o tempo lógico do pedido que será atendido. Ou seja, se um nó voltar a requisitar o recurso, seu tempo lógico será maior e, logo, a mesma mensagem de *token* não fará com que este nó entre na seção crítica. Ao receber uma mensagem de *token*, o nó passa ao estado U e, por fim, I novamente, portanto, a propriedade vale em todo este ciclo. Como uma nova resposta só será emitida quando a anterior tiver sido satisfeita, apenas uma resposta ativa pode circular pela rede. Logo, a propriedade vale em todo instante. Então, como existe no máximo um nó no

estado U, a segurança é preservada. ■

2) *Vivacidade* é garantida: Vamos provar que, se a condição C_4 vale em $\mathcal{G}$, então todo nó que realiza a primeira regra do protocolo de entrada em algum momento, executará a segunda regra do protocolo de entrada.

Como a condição C_4 e a hipótese forte da progressão valem na rede $\mathcal{G}$, então, todo nó que faz um pedido está no conjunto $\mathcal{R}(\mathcal{G})$ e, portanto, seus pedidos em algum momento chegam em um nó que detém o *token*. Analogamente, toda mensagem enviada com o *token* também sempre chega a seu destino.

Vale notar que, uma vez que um pedido chega ao nó que tem o *token*, este não sairá do conjunto X até que seja atendido, pois quando a mensagem de *token* é enviada para outro nó, os pedidos também são, bem como o vetor S. Logo, o conjunto X e o vetor S sempre estão atualizados no nó que está com o *token* e, portanto, a regra monádica determina corretamente o conjunto de pedidos não atendidos dentre aqueles conhecidos pelo nó que detém o *token* para calcular o de maior prioridade (a regra considera todos os pedidos em P cujo nó correspondente possui valor de S menor do que o tempo lógico do pedido). Deste modo, basta mostrar que todo pedido que está no conjunto X será atendido.

Considere que o nó u está com o *token* no instante i. Se u desejar entrar na seção crítica, deve emitir um pedido, que entrará no conjunto X_i. Se existe algum pedido neste conjunto, o nó deverá enviar uma mensagem de *token* para aquele de maior prioridade. Os pedidos em X_i são totalmente ordenados, então, u, enviará esta mensagem para o primeiro pedido nesta ordenação utilizando a regra monádica. Ao receber a resposta, o nó entra na seção crítica e, ao realizar o protocolo de saída, marca o tempo lógico do seu pedido realizado no vetor S, o que faz com que seu pedido saia do conjunto X e deixe de ser o de maior prioridade. Se este nó emitir outro pedido, a prioridade dele será menor, já que seu relógio lógico é incrementado a cada mensagem. Portanto, como os pedidos estão totalmente ordenados, o pedido de maior prioridade sempre é atendido e todo pedido atendido deixa de ser o de maior prioridade, todos os pedidos no conjunto X de um nó que detém o *token* serão atendidos em algum momento. E, como todo pedido em algum momento chega ao nó com o *token*, todo pedido feito é atendido. Portanto, a vivacidade é preservada. ■

Podemos notar que a condição C_4 leva em conta conhecimento sobre quais nós da rede vão fazer pedidos. Se este conhecimento estiver disponível, então, o algoritmo de Helary et. al requer uma condição de conectividade mais fraca do que o de Ricart e Agrawala. Do contrário, a condição C_4 considerará que o conjunto $\mathcal{R}(\mathcal{G})$ de nós que fazem pedidos é igual ao conjunto de nós $V(\mathcal{G})$ da rede e, portanto, obtemos a condição C_2, a mesma do algoritmo de Ricart e Agrawala.

4. Conclusões

Neste trabalho, apresentamos uma extensão de um trabalho anterior [Floriano et al. 2011] estudando uma formalização para os algoritmos de exclusão mútua de Ricart e Agrawala e de Helary et al. Este estudo nos permitiu observar que o algoritmo baseado em *token* possui uma condição de conectividade mais fraca desde que os nós que requisitam entrada na seção crítica sejam conhecidos. Caso contrário, as condições são iguais. Além disto, o algoritmo baseado em *token* troca menos mensagens e guarda menos informações em

cada nó, o que é uma vantagem quando se trata de uma rede dinâmica.

Referências

Anta, F. A., Milani, A., Mosteiro, M., and Zaks, S. (2010). Opportunistic information dissemination in mobile ad-hoc networks: The profit of global synchrony. In *DISC*, pages 374–388.

Arantes, L., Sens, P., Thomas, G., Conan, D., and Lim, L. (2010). Partition participant detector with dynamic paths in mobile networks. In *IEEE NCA*, pages 224–228.

Badrinath, B., Acharya, A., and Imielinski, T. (1994). Structuring distributed algorithms for mobile hosts. In *14th ICDCS*, pages 21–28.

Baldoni, R., Bertier, M., Raynal, M., and Tucci Piergiovanni, S. (2007). Looking for a definition of dynamic distributed systems. In *PaCT*, pages 1–14.

Baldoni, R., Virgillito, A., and Petrassi, R. (2002). A distributed mutual exclusion algorithm for mobile ad-hoc networks. In *IEEE ISCS*, pages 539–545.

Casteigts, A., Chaumette, S., and Ferreira, A. (2010). Characterizing topological assumptions of distributed algorithms in dynamic networks. *SIROCCO*, pages 126–140.

Ferreira, A. (2002). On models and algorithms for dynamic communication networks: The case for evolving graphs. In *AlgoTel*, pages 155–161.

Floriano, P., Arantes, L., and Goldman, A. (2012). Condições de Conectividade de Algoritmos de Exclusão Mútua em Redes Dinâmicas. Technical Report RT-MAC-2012-01, DCC, IME-USP.

Floriano, P., Goldman, A., and Arantes, L. (2011). Formalization of the necessary and sufficient connectivity conditions to the distributed mutual exclusion problem in dynamic networks. In *IEEE NCA*, pages 203–210.

Helary, J. M., Plouzeau, N., and Raynal, M. (1988). A distributed algorithm for mutual exclusion in an arbitrary network. *Comput. J.*, 31:289–295.

Lamport, L. (1978). Time, clocks, and the ordering of events in a distributed system. *CACM*, 21(7):558–564.

Litovsky, I. and Sopena, E. (1999). Graph relabelling systems and distributed algorithms. *Handbook of Graph Grammars and Computing by Graph Transformation. Concurrency, Parallelism, and Distribution*, 3:1–56.

Mostefaoui, A., Raynal, M., Travers, C., Patterson, S., Agrawal, D., and El Abbadi, A. (2005). From static distributed systems to dynamic systems. In *IEEE SRDS*, pages 109–118.

Raymond, K. (1989). A tree-based algorithm for distributed mutual exclusion. *ACM TOCS*, 7(1):61–77.

Ricart, G. and Agrawala, A. (1981). An optimal algorithm for mutual exclusion in computer networks. *Communications of the ACM*, 24(1):9–17.

Suzuki, I. and Kasami, T. (1985). A distributed mutual exclusion algorithm. *ACM TOCS*, 3(4):344–349.

Walter, J. E., Welch, J. L., and Vaidya, N. H. (2001). A mutual exclusion algorithm for ad hoc mobile networks. *Wireless Networks*, 7:585–600.