Especificação e Protocolo para a Gestão da Filiação ao Grupo em Redes Móveis Ad Hoc*

Bruno Rios Patriarca Nunes[1,2], **Fabíola Gonçalves Pereira Greve**[1]

nunes@dcc.ufba.br, fabiola@dcc.ufba.br

[1]Departamento de Ciência da Computação – Universidade Federal da Bahia (UFBA)
Av. Adhemar de Barros, S/N, Campus de Ondina - 40.170-110 - Salvador - BA - Brazil

[2]Instituto Recôncavo de Tecnologia
Av. Tancredo Neves, 805, 3o., Cam. das Árvores, 41820-021 - Salvador - BA - Brazil

Resumo. *A gestão da filiação ao grupo constitui um serviço essencial para se garantir comunicação confiável em sistemas dinâmicos sujeitos a falhas. Quando se adiciona a mobilidade, responsável por frequentes desconexões na rede, a tarefa de gerenciar a formação e evolução de um grupo torna-se especialmente complexa. Este trabalho propõe uma especificação e um protocolo para o problema da gestão da filiação ao grupo em sistemas dinâmicos, em especial, nas redes móveis ad hoc. A solução final consiste de um protocolo genérico que coordena de forma assíncrona a composição do grupo, tolerando a ocorrência de eventuais particionamentos e reconexões.*

Abstract. *Group membership is a basic building block in order to provide reliable communication in fault-prone dynamic systems. When node mobility cut off communication links, managing the creation and evolution of a group become a quite complex task. This paper proposes a specification and implementation for the group membership problem in dynamic systems and in particular, mobile ad hoc networks. The proposed algorithm consists of a generic partition-aware asynchronous protocol that provides useful guarantees for the application.*

1. Introdução

As redes móveis ad hoc, ou MANETs, são redes nas quais os nós são móveis e a comunicação entre eles é limitada pelo alcance dos transmissores sem fio. As principais características das MANETs são a auto-organização, a descentralização e o alto dinamismo [Basile et al. 2003]. Esse modelo, que se opõe em muitos aspectos às redes cabeadas tradicionais, abre perspectivas para novas aplicações como sistemas militares e operações de resgate, detecção de engarrafamentos via comunicação inter-veicular e sistemas inteligentes de monitoramento de fenômenos naturais. Essas aplicações, em geral, prescindem de um controle centralizado e toleram falhas e participantes desconhecidos [Oliveira 2007].

Num ambiente móvel e sujeito à falhas, os nós têm liberdade de entrar ou sair da rede voluntariamente e podem falhar arbitrariamente. Esses eventos podem criar partições na rede que, para que a confiabilidade dos serviços seja garantida, devem ser detectadas

*Este trabalho tem apoio do CNPQ - Brasil.

e tratadas pelos protocolos de comunicação. Desta maneira, um serviço de comunicação deve ser tolerante ao surgimento de partições na rede. O sistema pode ter seu desempenho degradado e reduzido, mas não deve ficar totalmente indisponível. Cada partição funcionará como um sistema autônomo, oferencendo serviços aos membros na medida do possível [Conan et al. 2008].

O problema da gestão da filiação ao grupo, ou *group membership* (GM), tem a responsabilidade de criar e manter um histórico coerente da evolução da composição do grupo de processos a partir da ocorrência de eventos como a entrada e saída de membros, e falhas nos canais e processos. A percepção que um processo tem do seu grupo é a visão do grupo, sendo esse conceito importante para a classificação do problema em duas variantes. A primeira delas é a chamada *gestão da filiação ao grupo com componente primária*. Nesse modelo, existe uma única visão compartilhada por todos os processos. Já a *gestão da filiação ao grupo com componentes particionáveis* é mais flexível do que a primeira, permitindo a existência de visões concorrentes distintas, que não se interceptam. Para isso, os protocolos que implementam esse tipo de serviço devem prover mecanismos que tolerem particionamento (*splits*) e reagrupamentos (*merges*) [Greve 2005].

A gestão da filiação ao grupo é um problema amplamente estudado no contexto de sistemas clássicos. Em [Greve et al. 2001], por exemplo, é desenvolvido um protocolo assíncrono de GM com componente primária, como extensão ao problema do consenso. Essa solução, entretanto, não dá suporte à mobilidade dos nós. Por sua vez, num contexto de MANETs, poucas são as propostas para o GM. Em [Huang et al. 2004], usa-se o conceito de distância segura entre nós, calculada em termos da velocidade máxima dos nós. As operações do grupo são coordenadas por um líder e o protocolo não considera a ocorrência de falhas. A solução apresentada em [Briesemeister 2001] propõe um serviço de filiação ao grupo baseado na localização geográfica dos nós. Essa proposta, entretanto, não oferece garantias úteis à aplicação, como o acordo entre os membros do grupo à respeito da visão.

Este artigo propõe uma especificação e um protocolo particionável para o problema da gestão da filiação ao grupo em sistemas distribuídos dinâmicos, visando, em especial, as redes móveis ad hoc. A especificação toma por base as propriedades propostas em [Babaoglu et al. 2001] para sistemas clássicos e as adapta para um contexto móvel. Note-se que as propriedades definidas para o GM não podem ser satisfeitas em sistemas assíncronos. Assim, será preciso estender o sistema com detectores de falhas não confiáveis [Chandra and Toueg 1996]. Estes abstraem as condições de sincronia necessárias para a resolução do problema. O protocolo de GM apresentado faz uso dos detectores da classe $\Diamond\mathcal{P}$. Estes podem se equivocar e suspeitar de processos indevidamente, até que, a partir de um tempo não definido, passam a ser perfeitos e não mais cometem erros.

No protocolo de GM, cada nó mantém a visão do grupo ao qual pertence, e a atualiza com base nas suspeitas apresentadas pelo detector de falhas e na percepção de novos membros. As mudanças efetuadas sobre a visão são feitas de maneira autônoma por cada nó e posteriormente comunicadas ao resto dos nós na rede. Essa característica distingue o protocolo proposto de outros, nos quais, em geral, os nós comunicam a intenção de realizar alguma operação, aguardam a autorização do grupo, e só então executam de fato a operação desejada. Tal autorização é normalmente obtida a partir da realização

de um protocolo de consenso e a decisão sobre a visão será única para todos os processos [Greve et al. 2001]. No caso de MANETs, esse acordo prévio sobre a visão do grupo não seria possível devido ao desconhecimento do conjunto de processos na rede e às constantes mudanças de topologia. Assim, propõe-se um protocolo de GM particionável que "incentiva" a existência inicial de visões divergentes (dado que os nós têm autonomia para incorporar mudanças nas visões de maneira unilateral), mas que, ao longo do tempo, promove a convergência e o acordo entre as visões, desde que condições de estabilidade e de sincronia sejam satisfeitas pela rede. Logo, é importante ressaltar que, mesmo com o caráter distribuído com que as operações sobre o grupo são realizadas, ainda é possível oferecer garantias úteis à aplicação, como o acordo entre visões.

O restante desse artigo está estruturado dessa forma: a seção 2 define o modelo de sistema. As seções 3 e 4 contêm a especificação do serviço de gestão da filiação ao grupo e a apresentação do protocolo, respectivamente. A seção 5 comenta uma possível aplicação para o protocolo e a seção 6 conclui o trabalho, apontando propostas de trabalhos futuros.

2. Modelo do Sistema

O sistema é modelado como um grafo não direcionado $G = (\Pi, E)$, onde Π é o conjunto de vértices e E o conjunto das arestas. Os vértices são os nós, e em cada nó executa apenas um processo. Cada nó possui um mecanismo de rádio difusão de alcance r. Se dois nós p e q estão a uma distância $d \leq r$ então existe uma aresta $e = (p, q) \in E$ que os conecta. Como o sistema é dinâmico e os nós são móveis, a conectividade entre os processos varia no tempo. Se há um caminho ligando dois nós em um dado momento t, então diz-se que p e q estão conectados em t ou, $p \rightsquigarrow_t q$. Como a propriedade de conectividade é comutativa, então $p \rightsquigarrow_t q \Leftrightarrow q \rightsquigarrow_t p$. A cardinalidade do conjunto Π não é previamente conhecida. Os nós são dotados de mobilidade e podem se mover livremente pelo ambiente. Variáveis como velocidade, direção e intenção de movimento são desconhecidas pelos processos. Para facilitar a apresentação das propriedades, um relógio global é definido como $T \subseteq \mathbb{N}$, entretanto ele não é acessado pelos processos. Os termos nó e processo serão utilizados indistintamente neste trabalho.

2.1. Histórico de Eventos

A execução de cada processo no sistema pode ser descrito como uma sequência de eventos realizados a cada passo do relógio [Babaoglu et al. 2001]. Seja S o conjunto de possíveis eventos, sendo que S contém, no mínimo, as funções $send()$, $receive()$ e o evento nulo ϵ. Pode-se então definir uma função $\sigma : \Pi \times T \rightarrow S$, que mapeia os eventos realizados pelos processos em cada instante da execução. Se o processo p realiza o evento e no instante t, pode-se dizer que $\sigma(p, t) = e$. Analogamente, $\sigma(p, t) = \epsilon$ indica que o processo p não praticou nenhum evento em t.

2.2. Modelo de Falhas

Considera-se a existência de falhas por colapso ou *crash*. Nesse tipo de falha os processos param precocemente e não mais retomam a execução. Um processo que falha realiza apenas o evento nulo. Pode-se então definir uma função de falha, $fail : T \rightarrow 2^{\Pi}$, que associa cada instante $t \in T$ aos processos que falharam nesse intervalo. Formalmente,

$$p \in fail(t') \Rightarrow \sigma(p, t) = \epsilon, \forall t \geq t'$$

Como os processos não se recuperam de uma falha por colapso, então $fail(t) \subseteq fail(t+1)$. Se um nó $p \notin fail(t)$ então dizemos que p é um processo correto em t. Assim, pode-se definir a função $correct : T \rightarrow 2^{\Pi}$, que, de forma complementar à função $fail$, fornece o conjunto de processos corretos em um determinado momento.

$$p \in correct(t) \Leftrightarrow p \notin fail(t)$$

2.3. Modelo de Comunicação

A comunicação entre os processos é feita pelo envio e recebimento de mensagens através de canais sem fio, via rádio frequência. Sem perda de generalidade, cada mensagem possui um identificador único no sistema (isso pode ser alcançado rotulando a mensagem com o identificador do nó juntamente com um código incremental). Uma mensagem m é enviada para processo q através de uma chamada $send(m, q)$. O recebimento de uma mensagem m é feito pela chamada $receive(m)$. Uma mensagem só pode ser recebida se ela foi enviada anteriormente por algum processo. Formalmente,

$$\sigma(q, t') = receive(m) \Rightarrow \sigma(p, t) = send(m, q), t < t'$$

Os canais de comunicação locais são confiáveis, ou seja, dado $(p, q) \in E$ se p envia a mensagem m para q, então q recebe m vinda de p. Não são consideradas perdas e colisões e os canais são FIFO. Dessa forma, as mensagens podem chegar em uma ordem diferente da qual foram enviadas. Essa consideração sobre a confiabilidade dos canais locais é realista, sendo já proporcionada por protocolos de acesso ao meio específicos para canais sem fio, como em [Si and Li 2004].

3. Especificação do Serviço de Gestão da Filiação ao Grupo

A especificação do serviço de gestão da filiação ao grupo (ou GM) com componentes particionáveis, propõe cinco propriedades a serem satisfeitas. Esse trabalho se baseou na especificação sugerida em [Babaoglu et al. 2001], com algumas modificações e adaptações para suportar a mobilidade dos nós. São condições que, além de manterem o problema do GM solucionável, dão garantias úteis às aplicações usuárias do serviço, como o acordo entre as visões, por exemplo.

Um grupo é definido como um conjunto de processos que compartilham um mesmo objetivo e que se relacionam por meio do identificador do grupo. Os grupos são criados pela aplicação e, a partir daí, os processos podem se unir ao novo grupo, pela primitiva $join$. Caso algum membro falhe ou se desconecte pela mobilidade, ele deve ser excluído da visão do grupo. A partir do instante que um processo se junta a um grupo, ele não invoca outro $join$ durante a sua execução. Os processos podem também sair espontaneamente do grupo através da primitiva $leave$.

A visão V de um processo p é o conjunto de nós que corresponde à percepção que p tem do seu grupo. Quando uma visão antiga é substituída por uma nova, diz-se que o processo instalou uma nova visão. Cada visão V possui um identificador, que pode ser acessado por $V.id$.

Com a mobilidade dos nós e as chances de ocorrência de falhas, possivelmente um grupo poderá ser particionado. Isso faz com que alguns membros com o mesmo identificador de grupo sejam temporariamente desconectados uns dos outros. Assim, as propriedades aqui definidas, valem para cada partição ou componente conexa. Um grupo

G pode ser particionado em n componentes conexas $G_1, ..., G_n$. Cada nó pertence a apenas uma partição a cada instante, ou seja, $G_1 \cap ... \cap G_n = \emptyset$. Se dois nós p e q estão em uma mesma componente conexa G_k em um dado momento t, então existe um caminho P entre eles tal que $\forall v \in P, v \in correct(t)$, e pode-se dizer que $p \rightsquigarrow_t q$. Caso contrário, $p \not\rightsquigarrow_t q$. Após um tempo, caso a comunicação entre as partições seja restaurada, os nós do mesmo grupo que antes estavam em partições diferentes, se reunirão novamente, assumindo a mesma configuração de visão anterior ao particionamento. Se um processo p pertence ao grupo G no momento t, diz-se que $p \in_t G$. Caso contrário, $p \notin_t G$.

O serviço de GM usará as primitivas $install(p, V)$ e $view(p, t)$. A primeira implica na instalação da visão V pelo processo p. Já a segunda retorna a visão que o processo p tem do grupo no instante t. As funções de entrada e saída do grupo serão representadas por $join(p, G)$ e $leave(p, G)$, onde G é o grupo em questão.

A seguir, serão apresentadas as propriedades de um GM particionável com suporte à mobilidade.

Integridade ou auto-inclusão: Se um nó p instala uma visão V então $p \in V$. Formalmente,

$$install(p, V) \Rightarrow p \in V, \forall p \in \Pi \tag{1}$$

Exatidão: Se um nó p, invoca a entrada no grupo G através da chamada $Join()$, então existe um tempo a partir do qual sua visão conterá todos os processos corretos do grupo que pertencem à sua componenente conexa.

$$join(p, G) \Rightarrow \exists t_i \in T : \forall q, \forall t > t_i, p \rightsquigarrow_t q \wedge q \in correct(t) \wedge q \in_t G \Rightarrow q \in view(p, t) \tag{2}$$

Completude: Existe um tempo a partir do qual cada nó p pertencente ao grupo G excluirá de sua visão os nós falhos, que se moveram para fora da componente conexa ou que efeturam uma chamada $leave$.

$$\exists t_i \in T : \forall q, \forall t > t_i, p \not\rightsquigarrow_t q \vee q \in fail(t) \vee q \notin_t G \Rightarrow q \notin view(p, t) \tag{3}$$

Acordo: Existe um tempo a partir do qual se dois processos p e q pertencem ao mesmo grupo e estão na mesma componente conexa, eles instalarão a mesma visão.

$$\exists t_i \in T : \forall t > t_i, p \rightsquigarrow_t q \Rightarrow p \in view(q, t) \Leftrightarrow q \in view(p, t) \tag{4}$$

Ordem entre as visões: Se o processo p instala uma visão V' após a visão V, então o identificador de V' é maior que o de V.

$$\sigma(p, t) = install(p, V) \wedge \sigma(p, t') = install(p, V') \wedge t' > t \Rightarrow V'.id > V.id \tag{5}$$

4. Protocolo de Gestão da Filiação ao Grupo Tolerante a Falhas

A especificação do serviço de composição de grupos proposta acima possui duas características importantes. O protocolo deve apresentar exatidão após um tempo e completude após um tempo. Para isso, precisa-se de um detector de falhas que, a partir de algum momento, seja capaz de suspeitar de todos os processos falhos, ao mesmo tempo em que não

suspeite dos processos corretos. Essas propriedades apontam para a classe de detectores de falhas não confiáveis perfeitos após um tempo ou $\Diamond\mathcal{P}$, cujo comportamento pode ser descrito através de duas propriedades [Chandra and Toueg 1996]:

Completude forte após um tempo: Existe um tempo a partir do qual cada processo suspeitará de todo processo falho.

Exatidão forte após um tempo: Existe um tempo a partir do qual cada processo não suspeitará de todo processo correto.

A detecção global de falhas pode ser muito custosa, especialmente em cenários com restrição de energia, devido ao grande número de mensagens enviadas e recebidas. Visando contornar essa limitação, em [Sridhar 2006] é proposto um detector da classe $\Diamond\mathcal{P}_l^m$, que restringe ao contexto local as propriedades de completude e exatidão. Além disso, o detector tolera a mobilidade, sendo capaz de distinguir um nó falho de um nó que se moveu. As informações locais são periodicamente disseminadas pela rede, a fim de compartilhar o conhecimento sobre as suspeitas. Esse detector pode também ser utilizado pelo serviço de GM, porém, para garantir a completude do protocolo, é preciso considerar que, após um tempo, o conjunto de vizinhos de cada nó permanecerá estável.

A seguir, será proposto um protocolo assíncrono de GM para redes móveis ad hoc. O protocolo segue a especificação da seção 3. O algoritmo final é assíncrono, sendo que as restrições temporais serão encapsuladas pelo uso de um detector de falhas $\Diamond\mathcal{P}$, conforme considerado anteriormente.

4.1. Princípios de Funcionamento

O protocolo proposto implementa um serviço de GM que tolera a ocorrência de eventuais particionamentos na rede. Os grupos são criados pela aplicação e, em seguida, ficam disponíveis para receber novos membros. Os nós suspeitos de falha devem ser retirados da visão corrente. Um grupo possivelmente terá membros em mais de uma partição. Nesse caso, cada partição se comporta como um sistema autônomo, sendo que, dentro da partição, o grupo mantém suas características, exceto pelo conjunto reduzido de membros. Caso as partições voltem a se comunicar, o protocolo se encarrega de mesclar as visões em uma única que contenha todos elementos do grupo.

Cada membro do grupo envia periodicamente mensagens com a sua visão. Essas mensagens não são enviadas a todos e têm alcance limitado aos vizinhos. Por transitividade, a partir de envios sucessivos, as informações podem ser compartilhadas por todos os membros do grupo. Com essa troca de informações, é possível que todo o grupo atualize seu estado de maneira uniforme. Informações recentes são diferenciadas das antigas através de contadores, presentes tanto em cada processo quanto nas mensagens de grupo. Quanto mais alto esse valor, mais recente é uma informação. Assim, garante-se que cada nó apenas considerará as informações mais novas do que as que ele já possui.

4.2. Propriedades Comportamentais

Define-se aqui as propriedades comportamentais necessárias para garantir a correção do protocolo de gerenciamento da composição do grupo.

Propriedade de Estabilização de Grupo ($\mathcal{GSP}$). *Após um tempo, os nós do grupo G se organizarão de tal forma, que, para quaisquer p_i e p_j pertencentes a G, existe um caminho P de nós corretos entre p_i e p_j, tal que todo $p_k \in P$ pertence a G.*

Pela propriedade $\mathcal{GSP}$, considera-se que, após um tempo, todo nó de um grupo G possui pelo menos um vizinho que também pertence a G. Essa hipótese é necessária pois a troca de mensagens sobre o grupo não é feita em difusão pela rede, mas sim apenas entre os vizinhos. Como cada processo apenas tratará das mensagens concernentes aos membros do seu grupo, é necessário que, a partir de um determinado tempo, todo o grupo esteja conectado, para que assim as informações mais atuais possam ser recebidas uniformemente por todos os seus membros. Tal propriedade coincide com o que a aplicação exige de um grupo, i.e., que seus membros estejam conectados.

Propriedade de Desligamento ($\mathcal{LP}$). *Sejam p_i, p_j processos distintos pertencentes ao grupo G. A propriedade estabelece que se p_i sair do grupo, invocando $leave(p_i, G)$, então, após um tempo, existe um nó correto p_j que receberá essa informação, invocando $receive(leave(p_i, G))$.*

A propriedade de desligamento $\mathcal{LP}$ é necessária à completude do protocolo, pois os processos que sairam de um grupo G, através do $leave$, devem ser retirados da visão dos processos que ainda pertencem a G. Assim, garante-se que o grupo tomará conhecimento da saída de algum membro, assegurando que este seja retirado das visões dos membros restantes.

4.3. Aspectos da Implementação do Protocolo

O algoritmo usará as seguintes variáveis e funções:

- gid_i: É o identificador do grupo do qual p_i é membro, representado por um número natural. Assume o valor ϵ (nulo) caso o nó não pertença a algum grupo.
- $view_i$: Conjunto dos processos que compõem a visão corrente de p_i.
- $view\_counter_i$: Contador das visões instaladas por p_i no grupo gid_i.
- $unique\_gid()$: Função que retorna um identificador de grupo único.
- $left_i$: Lista dos nós que deixaram o grupo através da primitiva $leave$.

O protocolo tem suas funções divididas em dois blocos. O primeiro módulo contém os procedimentos que interagem diretamente com a aplicação, enquanto o segundo, mais baixo nível, manipula com informações do detector de falhas e lida diretamente com a formação das visões.

Interface com a Aplicação: As tarefas de interface com a aplicação são três e estão detalhadas no algoritmo 1.

A função de criação de grupo, $CreateGroup()$, é responsável por criar e inicializar um novo grupo. O identificador do grupo é fornecido por uma função genérica $unique\_gid()$ que retorna um valor único a cada chamada (linha 2). A visão inicial contém apenas o próprio nó criador do grupo (linha 3). O contador de visões, $view\_counter_i$, recebe o valor inicial 0 (linha 3). A lista de membros que saíram do grupo está vazia no momento da criação do grupo (linha 4).

Um processo p_i que deseja se juntar a um grupo deve invocar o procedimento $Join()$. A aplicação chama essa rotina do serviço de GM, provendo como parâmetros o identificador do grupo (gid_j), a visão corrente ($view_j$), o contador de visões atual ($view\_counter_j$) e a lista de membros que saíram do grupo ($left_i$). p_i então atualiza seu

estado com as informações do grupo, garantindo que ele próprio estará na visão (linha 9), e que essa visão será a mais recente (linha 10).

Quando um nó p_i deseja se retirar do grupo corrente, ele faz uma chamada à função $Leave()$. Esse procedimento inclui p_i em sua própria lista $left_i$ (linha 14). Uma vez efetuada uma chamada à função $Leave()$, p_i apenas poderá se unir a um outro grupo caso modifique seu identificador.

Uma característica importante da solução proposta é que os nós têm um comportamento proativo quanto às operações de entrada e saída do grupo. Assim, quando um nó deseja se juntar a um grupo, através da chamada $Join()$, ele simplesmente se inclui na visão e passa a enviar mensagens para o grupo. Os outros membros, então, receberão suas mensagens e tomarão conhecimento da existência de um novo elemento no grupo. Assim, o restante dos processos atualiza suas visões, adicionando o nó recém chegado. O mesmo vale para uma saída espontânea do grupo pela primitiva $Leave()$. A propagação das mudanças na visão é feita a partir da execução do Algoritmo 2. É importante perceber a diferença desse modelo em relação ao que é geralmente adotado pelos protocolos clássicos. É comum que os processos solicitem sua entrada ou saída e aguardem a aprovação do grupo [Greve 2005]. A abordagem do presente trabalho, por outro lado, considera que as chamadas $Join()$ e $Leave()$ sempre provocarão a entrada ou a saída do grupo, respectivamente.

Algoritmo 1 Protocolo de Gestão da Filiação do Grupo - Interface com a aplicação

```
1:  CreateGroup() :
2:  gid_i ← unique_gid()
3:  view_i ← {p_i}
4:  view_counter_i ← 0
5:  left_i ← ∅
6:
7:  Join(gid_j, view_j, view_counter_j, left_j)
8:  gid_i ← gid_j
9:  view_i ← view_j ∪ {p_i}
10: view_counter_i ← view_counter_j + 1
11: left_i ← left_j
12:
13: Leave() :
14: left_i ← left_i ∪ {p_i}
```

Formação e Manutenção das Visões: O segundo módulo da gestão da composição do grupo lida diretamente com a formação e manutenção das visões, e está detalhado no Algoritmo 2. Dois eventos podem disparar modificações na visão: (i) a geração de novas suspeitas pelo detector (ii) o recebimento de uma informação mais recente à respeito da visão. Considera-se que o detector possui uma lista $suspected_i$ que contém os processos suspeitos de serem falhos. O módulo executa três tarefas principais.

A Tarefa 1 tem a responsabilidade de atualizar a visão do grupo a partir das suspeitas geradas pelo próprio nó p_i. Consiste em um laço infinito que, a cada iteração,

acessa a lista de suspeitos gerada pelo detector. Caso a lista de nós suspeitos, contenha algum nó da visão corrente de p_i, esses nós são retirados da visão e o contador de visões de p_i é incrementado (linhas 4-7).

Periodicamente, cada nó enviará suas informações sobre o grupo para os seus vizinhos, através da mensagem VIEW (p_i, gid_i, $view_i$, $view\_counter_i$, $left_i$) da Tarefa 2 (linhas 11-13). A partir dessa troca de informações, por transitividade, é possível que todo grupo (dentro da partição) atualize suas visões a partir do conhecimento de outros nós. O recebimento dessas mensagens é tratado na Tarefa 3.

Algoritmo 2 Protocolo de Gestão da Filiação do Grupo - Gerenciamento das Visões

```
1:
2:  Tarefa 1 : Mudanças de visão geradas pela saída do detector de falhas
3:  loop
4:     if view_i ∩ suspected_i ≠ ∅ then
5:        view_i ← view_i \ suspected_i
6:        view_counter_i ← view_counter_i + 1
7:     end if
8:  end loop
9:
10: Tarefa 2 : Envio de mensagens de grupo
11: loop
12:    send VIEW(p_i, gid_i, view_i, view_counter_i, left_i) to all neighbors
13: end loop
14:
15: Tarefa 3 : Recebimento de mensagens de grupo
16: upon reception of VIEW(p_j, gid_j, view_j, view_counter_j, left_j) from p_j do
17: if gid_i = gid_j then
18:    left_i ← left_i ∪ left_j
19:    if p_j ∈ left_j and p_j ∈ view_i then
20:       view_i ← view_i \ left_i
21:       view_counter_i ← view_counter_i + 1
22:    else if view_counter_j > view_counter_i then
23:       view_counter_i ← view_counter_j
24:       if p_i ∉ view_j or view_j ∩ left_i ≠ ∅ then
25:          view_j ← view_j ∪ {p_i} \ left_i
26:          view_counter_i ← view_counter_i + 1
27:       end if
28:       view_i ← view_j
29:    else if view_counter_j = view_counter_i then
30:       if view_i ≠ view_j then
31:          view_i ← view_i ∪ view_j \ left_i
32:          view_counter_i ← view_counter_i + 1
33:       end if
34:    end if
35: end if
```

A Tarefa 3 é disparada quando o processo p_i recebe do processo p_j uma mensagem

de grupo. Caso ambos pertençam ao mesmo grupo, p_i atualiza sua lista $left_i$ a partir da lista trazida por p_j. Após isso, podem ser tomadas três ações: (i) Se o processo p_j saiu do grupo (se incluiu na lista $left_i$) e p_i ainda ainda o tem em sua visão, p_j é excluído de $view_i$ e o contador de visões é incrementado (linhas 19-21). (ii) Se p_j possui um contador de visões maior que p_i, então p_i instala a visão $view_j \cup \{p_i\}$ e incrementa seu contador (linhas 22-28). (iii) Se p_i e p_j pertencem ao mesmo grupo e ambos possuem contadores de visão iguais, porém com conjuntos $view$ distintos, então p_i instala a visão $view_i \cup view_j$ e incrementa seu contador (linhas 26-31). Como é possível que p_j ainda tenha na sua visão $view_j$ membros que já deixaram o grupo, p_i sempre retira da visão a ser instalada os processos presentes em $left_i$ (linhas 25 e 31).

O compartilhamento das listas $left_i$ pode fazer com que estes conjuntos cresçam indefinidamente. Um problema semelhante foi apresentado em [Sridhar 2006] no contexto de detectores de falhas, onde as listas de processos suspeitos nunca eram esvaziadas. Esse trabalho sugere a adoção de um processo de limpeza periódico, que, com algum tipo de conhecimento global, seria responsável por tirar da lista $left_i$ os nós que já saíram do grupo e que todos os membros já tivessem conhecimento das suas saídas. Esse tipo solução, apesar de usar informações globais, se mostra essencial a fim de prover escalabilidade ao protocolo.

4.4. Aumentando a Justiça nas Mudanças de Visão

Uma característica do protocolo proposto é que, a cada suspeita gerada pelo detector, uma mudança de visão é disparada. Em um ambiente sujeito a falhas temporárias de comunicação, que são frequentes em canais sem fio, seriam produzidas várias mudanças de visão desnecessárias até que as condições de estabilidade fossem atingidas. Por conta disso, o protocolo de gestão da filiação ao grupo será estendido com o objetivo de garantir maior justiça nas mudanças de visão.

A solução sugerida neste trabalho se baseia em [Friedman 2003], onde aponta-se o uso de conjuntos *fuzzy* para o tratamento das suspeitas geradas. Como, em redes móveis, é comum a ocorrência de falhas temporárias, seria muito custoso instalar uma nova visão a cada suspeita produzida pelo detector. Em vez de se usar o modelo convencial, no qual cada processo assume apenas dois estados (vivo ou morto), esse proposta associa cada processo a um nível de participação no grupo (*fuzzyness level*). À medida em que um processo não responde a mensagens, seu nível *fuzzy* é reduzido. É importante notar que um processo apenas é excluído do grupo, caso atinga um nível *fuzzy* mínimo pré-determinado.

A modificação que o algoritmo sofrerá consiste em não retirar imediatamente da visão os processos suspeitos de serem falhos. Quando uma suspeita é gerada, o nó suspeito é colocado em uma lista à parte e lhe é atribuído um grau de confiança. Esse nível de confiança é decrementado a cada iteração (Tarefa 1) em que o nó suspeito não tenha sido retirado do conjunto $suspected_i$. Caso esse índice atinja um valor mínimo, o processo suspeito é automaticamente excluído da visão. Ainda que seja apenas uma heurística, esse mecanismo proporciona aos nós suspeitos o benefício da dúvida, diminuindo consideravelmente o número de mudanças de visão causadas por suspeitas infundadas. É importante observar que o grau de confiança não deve ter um valor muito alto, já que mudanças de visão causadas por falhas reais poderiam ser excessivamente postergadas.

Tampouco deve ter um valor muito baixo, pois assim a modificação do protocolo perderia seu efeito, que é justamente diminuir o número de mudanças de visão.

As seguintes variáveis e constantes serão usadas juntamente com as já definidas na primeira versão do protocolo:

- $quarantine_i$: É a lista dos processos atualmente considerados suspeitos por p_i. Os elementos dessa lista são pares da forma $< id, trust\_degree >$, onde id é o identificador do processo suspeito e $trust\_degree$ é o seu grau de confiança naquele momento. Quando inserido pela primeira vez, o processo recebe o grau de confiança padrão, DEFAULT_TRUST.
- TRUST_LIMIT: Ao atingir esse nível de confiança, o nó suspeito perde o direito de estar na lista de quarentena, e é removido da visão corrente.
- TRUST_DEC: A cada passo em que a suspeita sobre p_j não é revogada, o $trust\_degree$ é decrementado em TRUST_DEC unidades.

Para garantir maior justiça na exclusão de membros da visão, a Tarefa 1 foi estendida. Na nova versão do protocolo, detalhada no Algoritmo 3, quando uma suspeita é gerada, o processo suspeito é colocado em ”quarentena”, em vez de ser imediatamente excluído da visão (linhas 5-7). Caso um processo suspeito já esteja em quarentena, seu nível de confiança é então decrementado (linha 8). Se algum processo atingir o nível de confiança mínimo, ele é inserido em uma lista temporária de nós falhos (linhas 9-12), cujos elementos serão posteriormente retirados da visão (linhas 15-18). As tarefas 2 e 3 permancem as mesmas do algoritmo original.

Algoritmo 3 Protocolo de Gestão da Filiação do Grupo com Extensão para Aumentar Justiça nas Mudanças de Visão

```
1:  Tarefa 1 : Mudanças de visão geradas pela saída do detector de falhas - Estendida
2:  loop
3:     failed_members ← ∅
4:     for all p_j ∈ suspected_i do
5:        if < p_j, − > ∉ quarantine_i then
6:           quarantine_i ← quarantine_i ∪ {< p_j,DEFAULT_TRUST>}
7:        else
8:           trust_j ← trust_j− TRUST_DEC
9:           if trust_j ≤ TRUST_LIMIT then
10:             failed_members ← failed_members ∪ {p_j}
11:             quarantine_i ← quarantine_i \ < p_j, − >
12:          end if
13:       end if
14:    end for
15:    if failed_members ≠ ∅ then
16:       view_i ← view_i \ failed_members
17:       view_counter_i ← view_counter_i + 1
18:    end if
19: end loop
20:
```

4.5. Esboço da Prova de Correção

Para que o algoritmo fornecido seja realmente correto, é necessário que ele satisfaça a todas as cinco propriedades do GM definido na seção 3. Cada propriedade é assegurada para uma partição[1], que funciona de modo autônomo dentro da rede. Seja p_i um nó qualquer do sistema e G um grupo.

Integridade: A integridade é garantida no momento da criação de um grupo, já que a visão inicial sempre conterá o nó que executou a chamada $CreateGroup()$ (linha 3 - Algoritmo 1). No caso de efetuar um $Join()$, p_i também se inclui na visão instalada $view_j$, garantido a auto-inclusão (linha 9 - Algoritmo 1). Há ainda o caso de p_i receber uma informação de visão mais atualizada que a sua (linhas 22-28 - Algoritmo 2). Nessa situação, p_i instala a visão mais recente $view_j$ se incluindo na nova visão caso não esteja presente nela (linhas 24-27 - Algoritmo 2). Como p_i pode executar apenas um $Join()$, assegura-se que ele não pertencerá a nenhuma lista $left_i$ antes de deixar o grupo. Assim, retirar da visão os membros que saíram do grupo (linhas 20, 25, 31 - Algoritmo 2) não viola a integridade do protocolo, já que p_i não estará nesta lista. Dessa forma, a propriedade de auto-inclusão é satisfeita.

Exatidão: Pela propriedade de *exatidão após um tempo* do detector, tem-se que há um tempo $t \in T$ a partir do qual nenhum processo correto será considerado suspeito. Assim, a partir de t nenhuma suspeita sobre processos corretos será gerada pelo detector. Como a comunicação é confiável e, pela propriedade de estabilização de grupo $\mathcal{GSP}$, os membros do mesmo grupo serão vizinhos, existe um momento a partir do qual todos os nós em $correct(t)$ receberão as mensagens de pelo menos um nó do mesmo grupo. Sejam p_i e p_j nós corretos pertencentes a G. Caso, em t, a visão de p_i seja a mais recente e não contenha p_j (p_i suspeitava de p_j), p_j receberá em algum momento uma mensagem trazendo essa visão $view_i$ que não o contém. Assim, pelo algoritmo, p_j instalará a visão $view_i \cup \{p_j\}$, e essa será a mais recente (linhas 22-28 - Algoritmo 2). Essa visão será então propagada para os outros membros de G, e pela propriedade de exatidão do detector, nenhuma outra suspeita será gerada à respeito do nó p_j.

Completude: Através da *propriedade de completude*, o detector garante que, a partir de um determinado momento, todo processo falho será considerado suspeito por todo processo correto. Sejam p_i e p_j processos, sendo um correto e um falho, respectivamente. Se o processo p_i recebe a informação do detector que p_j é suspeito, p_i retira p_j de sua visão e a transmite como sendo a mais recente (linhas 4-7 - Algoritmo 2). A propriedade $\mathcal{GSP}$, assegura que a informação sobre a retirada de p_j alcançará, por transitividade, todos os membros do grupo. Como p_j realmente falhou e o detector assegura completude após um tempo, p_j sempre será considerado suspeito e não voltará mais a integrar a visão dos processos do grupo. É necessário também excluir da visão todos os processos que saíram espontaneamente do grupo. Seja p_k um nó que sai do grupo G através da chamada $Leave()$. A propriedade $\mathcal{LP}$ garante que algum processo de G tomará conhecimento da saída de p_k. Após um tempo, pela propriedade $\mathcal{GSP}$, todos os membros serão informados da saída de p_k e o excluirão das suas visões (linhas 20, 25, 31 - Algoritmo 2).

Acordo: Sejam p_i e p_j processos pertencentes ao grupo G, ambos da mesma

[1]Uma partição corresponde a uma componente conexa da rede.

partição. A propriedade $\mathcal{GSP}$ garante que, a partir de um dado tempo, as informações mais recentes alcançarão todos os nós do grupo. As saídas do detector também serão perfeitas a partir de um momento, o que assegura que os processos corretos não serão considerados suspeitos. Caso p_i receba de p_j uma visão mais recente, p_i atualizará sua visão e ambos concordarão sobre o conjunto $view$ (linhas 22-28 - Algoritmo 2). Já se p_i receber de p_j uma visão com o mesmo $view\_counter$, porém diferente da sua, a p_i instalará a visão $view_i \cup view_j$ e incrementará o seu $view\_counter$, que passará a ser o mais recente (linhas 29-34 - Algoritmo 2). Posteriormente p_j receberá de p_i a informação de uma visão mais recente, e a instalará, garantindo a sim o acordo entre p_i e p_j.

Ordem entre as visões: Cada visão é etiquetada com um número inteiro. Como, a cada nova visão instalada, esse contador é incrementado (linha 10 - Algoritmo 1) (linhas 6, 21, 26, 32 - Algoritmo 2), pode-se garantir que as visões sempre serão instaladas em ordem crescente de identificadores. Além disso, p_i instala uma visão apenas se esta não tiver um contador menor que o seu contador local $view\_counter_i$ (linhas 22, 29 - Algoritmo 2). Dessa forma, visões antigas nunca serão instaladas.

5. Aplicação para o Serviço de GM

O protocolo desenvolvido neste trabalho pode ter uma aplicação prática no contexto das redes *peer-to-peer* (P2P). Em [Godoi et al. 2007], é apresentada uma aplicação de compartilhamento de arquivos em ambientes P2P. Os processos podem oferecer ou requisitar arquivos para $download$. Os nós que oferecem o mesmo arquivo, se juntam em um grupo. Um algoritmo de escolha de líder define qual membro do grupo será o servidor propriamente dito. Caso o servidor falhe, o grupo deve detectar essa falha e excluí-lo da visão, para que então um novo líder seja escolhido. Esse serviço de compartilhamento de arquivos poderá ser executado sobre um módulo de gestão da composição do grupo. Essa camada de GM irá manter a visão consistente, tratando a ocorrência de eventuais falhas e saídas bruscas. Nesse caso, o líder servidor poderá ser qualquer membro da visão corrente do grupo. O algoritmo do presente trabalho se apresenta como uma alternativa viável, já que é tolerante a particionamentos e falhas, além de promover o acordo entre as visões.

6. Conclusão

Este trabalho especificou e propôs um protocolo para o serviço de gestão da filiação ao grupo para sistemas dinâmicos. Como a solução proposta suporta a mobilidade, ela é especialmente adequada às redes móveis ad hoc. O protocolo é assíncrono, sendo que as restrições temporais são encapsuladas por um módulo de detecção de falhas da classe $\Diamond\mathcal{P}$. Particionamentos no grupo são tolerados, sendo que o protocolo trata as operações de junção de visões de forma transparente à aplicação. A correção do protocolo apresentado foi demonstrada com base na satisfação de cada uma das cinco propriedades definidas. Uma possível aplicação ao protocolo proposto seria o problema do compartilhamento de arquivos em redes P2P. Nesse contexto, os processos se organizam em grupos com o objetivo de oferecer conteúdo para $download$.

Como trabalhos futuros, sugere-se a identificação de aplicações que podem se beneficiar do protocolo de GM proposto para um contexto de MANETs. A aplicação em redes P2P citada no artigo pode também ser um campo de estudo, com a implementação e avaliação do desempenho do algoritmo em um sistema real.

References

Babaoglu, O., Davoli, R., and Montresor, A. (2001). Group communication in partitionable systems: Specification and algorithms. *IEEE Transactions on Software Engineering*, 27(4):308–336.

Basile, C., Killijian, M.-O., and Powell, D. (2003). A survey of dependability issues in mobile wireless networks. Technical report, LAAS CNRS Toulouse, France.

Briesemeister, L. (2001). *Group Membership and Communication in Highly Mobile Ad Hoc Networks*. PhD thesis, School of Electrical Engineering and Computer Science, Technical University of Berlin, Germany.

Chandra, T. D. and Toueg, S. (1996). Unreliable failure detectors for reliable distributed systems. *J. ACM*, 43(2):225–267.

Conan, D., Sens, P., Arantes, L., and Bouillaguet, M. (2008). Failure, disconnection and partition detection in mobile environment. In *NCA '08: Proceedings of the 2008 Seventh IEEE International Symposium on Network Computing and Applications*, pages 119–127, Washington, DC, USA. IEEE Computer Society.

Friedman, R. (2003). Fuzzy group membership. In Schiper, A., Shvartsman, A. A., Weatherspoon, H., and Zhao, B. Y., editors, *Future Directions in Distributed Computing, Research and Position Papers*, volume 2584 of *Lecture Notes in Computer Science*, pages 114–118. Springer.

Godoi, A. F. B., Jr, E. P. D., and Greve, F. (2007). Uma ferramenta para comunicação confiável em sistemas p2p baseada em grupos de peers. *Workshop de Peer-to-Peer, with Brazilian Symposium on Computer Networks*, pages 51–62.

Greve, F. (2005). Protocolos fundamentais para o desenvolvimento de aplicações robustas. In *Minicursos SBRC 2005: Brazilian Symposium on Computer Networks*, pages 330–398, Fortaleza, CE, Brazil.

Greve, F., Hurfin, M., Raynal, M., and Tronel, F. (2001). Primary component asynchronous group membership as an instance of a generic agreement framework. In *Proceedings of the IEEE International Symposium on Autonomous Decentralized Systems (ISADS)*, pages 93–100.

Huang, Q., Julien, C., and Roman, G.-C. (2004). Relying on safe distance to achieve strong partitionable group membership in ad hoc networks. *IEEE Transactions on Mobile Computing*, 3(2):192–205.

Oliveira, T. B. (2007). The reliability of broadcasting protocols for mobile ad-hoc networks. Master's thesis, Departamento de Ciência da Computação, Instituto de Matemática, Universidade Federal da Bahia, Brasil.

Si, W. and Li, C. (2004). Rmac: A reliable multicast mac protocol for wireless ad hoc networks. In *ICPP '04: Proceedings of the 2004 International Conference on Parallel Processing*, pages 494–501, Washington, DC, USA. IEEE Computer Society.

Sridhar, N. (2006). Decentralized local failure detection in dynamic distributed systems. In *SRDS '06: Proceedings of the 25th IEEE Symposium on Reliable Distributed Systems*, pages 143–154, Washington, DC, USA. IEEE Computer Society.