Um Detector de Falhas Assíncrono para Redes Móveis e Auto-Organizáveis

Pierre Sens[1], **Fabíola Greve**[2], **Luciana Arantes**[1], **Mathieu Bouillaguet**[1], **Véronique Simon**[1]

[1]LIP6 - Université Paris 6 - INRIA
4, Place Jussieu 75252 Paris Cedex 05, França

[2]Departamento de Ciência da Computação (DCC) / Universidade Federal da Bahia (UFBA)
40170-110, Campus de Ondina, Salvador - Bahia, Brazil

{pierre.sens,luciana.arantes,mathieu.bouillaguet,veronique.simon}@lip6.fr, fabiola@dcc.ufba.br

Abstract. *We consider the problem of failure detection in dynamic networks such as MANETs. This paper presents an asynchronous implementation of a failure detector for unknown and mobile networks. Our approach does not rely on timers and neither the composition nor the number of nodes in the system are known. The proposal algorithm can implement failure detectors of class $\diamondsuit S$ when behavioral properties and connectivity conditions are satisfied by the underlying system. Simulation experiments have validated our approach.*

Resumo. *Detectores de falhas são serviços fundamentais para o projeto de aplicações tolerantes a falhas, pois provêm informações sobre falhas de processos no sistema. A grande maioria dos detectores existentes têm como base uma rede estática, cujo grafo de comunicação entre os nós é completo e tanto a quantidade, quanto o conjunto de participantes do sistema é conhecido. Além disso, quase todas elas baseiam-se no uso de temporizadores para a detecção. Tais condições, entretanto, revelam-se inadequadas para sistemas auto-organizáveis, já que a composição do sistema é mutável e o seu grafo de comunicação é dinâmico. Este trabalho apresenta uma implementação assíncrona de um detector de falhas não-confiável para redes onde a composição do sistema é desconhecida e cujos nós são móveis, a exemplo das redes MANETs. Caso o sistema satisfaça algumas propriedades comportamentais e de conectividade, o algoritmo proposto ainda implementa detectores da classe $\diamondsuit S$. O algoritmo foi implementado e resultados de simulação são apresentados como forma de validar a abordagem apresentada.*

1. Introdução

A computação distribuída moderna vem rapidamente evoluindo para integrar sistemas dinâmicos e auto-organizáveis, característicos das redes P2P não-estruturadas, redes de sensores sem fio e redes móveis auto-organizáveis (Manets). Entretanto, a oferta e o desenvolvimento de serviços confiáveis nesse novo contexto apresenta vários desafios. Tais redes são altamente dinâmicas e os nós podem mover-se, entrar ou sair da mesma a qualquer momento, eliminando-se a existência de uma infra-estrutura estática e de um controle centralizado. Por fim, devido à escassez de recursos, falhas são bastante frequentes.

Um detector de falhas é um serviço fundamental para o desenvolvimento de sistemas tolerantes a falhas. Sua importância foi evidenciada por Chandra e Toueg em [3]

quando propuseram a abstração de *detectores de falhas não-confiáveis* como uma forma de solucionar o problema do consenso num ambiente assíncrono [5]. Informalmente, um detector de falhas não-confiável (ou simplesmente, FD) é um oráculo distribuído que periodicamente provê uma lista de processos suspeitos de terem falhado. Neste artigo, estamos interessados em FD da classe $\diamondsuit S$. Esta classe apresenta as condições de sincronia minimais para a resolução do consenso de forma determinista e dado que existe uma maioria de processos corretos no sistema.

Nosso interesse está em desenvolver detectores para redes móveis e de composição desconhecida, tais como as redes móveis auto-organizáveis (MANETs). Este tipo de rede apresenta as seguintes propriedades: (1) um nó não conhece todos os participantes do sistema; ele apenas pode enviar mensagens para seus vizinhos, i.e., aqueles que estão no seu alcance de transmissão; (2) o tempo de transmissão entre os nós é altamente variável e imprevisível; (3) a rede não é totalmente conectada, o que significa que uma mensagem enviada por um nó precisa ser roteada através de um conjunto de nós intermediários até atingir o nó destino; (4) um nó pode mover-se e mudar o seu alcance de transmissão.

A grande maioria das implementações existentes para os detectores de falhas são baseadas numa comunicação do tipo todos-para-todos, onde os nós monitoram a vivacidade de todos os demais nós da rede através da troca de mensagens do tipo "eu estou vivo" [9, 12]. Dado que tais propostas baseiam-se no fato de que o conjunto de nós conhecem-se mutuamente e estão totalmente conectados, estas não são adequadas para sistemas dinâmicos. Alguns trabalhos recentes foram propostos para lidar com algum dinamismo e crescimento incremental das redes [9, 8, 2]. Porém, poucos deles toleram mobilidade [6, 14]. Além disso, todas essas propostas tomam como base o uso de temporizadores para a detecção da falha, pois, considera-se que, eventualmente, o sistema irá obedecer um limite para a comunicação entre os nós de forma permanente. De novo, tal hipótese não é realista para uma rede dinâmica em que, sobretudo, há movimentação dos nós. Em [10], Mostefaoui *et al.* propuseram uma implementação assíncrona de FD, cuja detecção fundamenta-se apenas no padrão de troca de mensagens e no conhecimento da quantidade de falhas (f) e do número de processos no sistema (n). Entretanto, seu modelo computacional consiste numa rede clássica onde os nós são totalmente conectados.

Este artigo apresenta um novo algoritmo assíncrono de detecção de falhas para sistemas dinâmicos de redes móveis e desconhecidas. Ele não se fundamenta no uso de temporizadores e não requer nenhum conhecimento a respeito da composição do sistema nem de sua cardinalidade. Além disso, tem características interessantes que permitem seu uso em larga escala. O processo de detecção das falhas é feito apenas com base na percepção local que um nó tem do estado da rede e não na troca de informações globais.

O princípio básico de nosso FD é a inundação de informações de suspeita de falhas na rede. Inicialmente, cada nó apenas conhece-se a si próprio. Depois, periodicamente, ele troca com os seus vizinhos um par de mensagens do tipo QUERY-RESPONSE. Assim, com base apenas na recepção dessas mensagens e no conhecimento parcial que o nó tem da composição do sistema (i.e., sua vizinhança), um nó é capaz de suspeitar de outros processos ou de revogar as informações de suspeita infundadas. Tais informações são encaminhadas através das próprias mensagens de QUERY e poderão atingir toda a rede, se o sistema atende a certas condições de conectividade, caracterizadas pela propriedade de *rede com f-cobertura*. Esta garante que existe sempre um caminho entre quaisquer dois nós ativos da rede, apesar da ocorrência de f falhas ($f < n$).

O detector proposto ainda implementa as propriedades da classe $\diamondsuit S$, caso os nós da rede apresentem certas propriedades comportamentais. Quatro propriedades foram definidas. A *propriedade de inclusão* (*membership property*) permite com que o nó possa ser reconhecido pelos demais; para tanto, ele deve ter interagido e enviado pelo menos uma mensagem na rede. A *propriedade de mobilidade* (*mobility property*) estabelece que um nó móvel deve se reconectar à rede por um período de tempo que seja suficiente para atualizar o seu estado quanto à suspeita de falhas e correções de suspeitas equivocadas. A *propriedade de receptividade* (*responsiveness property*) determina que, após um tempo, a comunicação entre um determinado nó e os demais nós da sua vizinhança serão mais rápidas do que quaisquer outras comunicações na sua vizinhança. Finalmente, a *propriedade de receptividade para mobilidade* (*mobility responsiveness property*) determina que ao menos um nó correto satisfaz a propriedade de receptividade e, além disso, após um tempo, sua vizinhança será composta por nós que não saem da sua área de cobertura.

O resto do artigo está organizado da seguinte maneira. A seção 2 especifica o modelo computacional e os detectores de falhas. A seção 3 apresenta o nosso FD assíncrono, considerando-se inicialmente que os nós não se movimentam na rede. Na seção 4, o algoritmo é estendido para suportar mobilidade. Resultados de simulação do seu desempenho são apresentados na seção 5 e uma comparação com trabalhos relacionados é feita em 6. Finalmente, a seção 7 conclui o trabalho.

2. Modelo de Sistema

Consideramos um sistema distribuído dinâmico formado pour um conjunto finito de $n > 1$ nós móveis, $\Pi = \{p_1, \ldots, p_n\}$. Ao contrário de uma rede clássica, num sistema dinâmico de *redes desconhecidas*, os processos[1] não conhecem Π nem a cardinalidade n do sistema. Ou seja, um processo p_i conhece apenas um subconjunto de Π. Processos se comunicam pela emissão e recepção de mensagens através de uma rede sem fio, por rádio frequência. Nenhuma hipótese temporal é feita com respeito à realização das ações efetuadas pelos processos ou pelos canais, ou seja, o sistema é *assíncrono*. Um processo pode falhar por parada (*crash*). Um processo *correto* é um processo que não falha durante a execução do sistema; senão ele é considerado *falho*. Seja f o número máximo de processos autorizados a falhar do sistema ($f < n$), então f é conhecido por todos os processos. Para simplificar, consideramos o intervalo $\mathcal{T}$ de pulsos de relógio como sendo o conjunto de número naturais. Processos não têm acesso a $\mathcal{T}$: ele é introduzido apenas por uma questão de conveniência da apresentação.

O sistema pode ser representado por um grafo de comunicação $G(V, E)$ no qual $V \subseteq \Pi$ representa o conjunto de nós e E, o conjunto de ligações. Assim, $(p_i, p_j) \in E$ se e somente se estes pertencem à mesma area de cobertura de difusão, aqui denotada por *range*. Neste caso, p_i e p_j são considerados *vizinhos* (*1-hop*). A topologia de G é dinâmica. A comunicação entre dois vizinhos é feita através da emissão de mensagens, seja por difusão (*broadcast*) ou diretamente (*ponto à ponto*). Os canais são considerados confiáveis; eles não alteram, não criam, nem perdem mensagens. Assim, uma mensagem m enviada por p_i através de difusão é recebida por todos os vizinhos corretos de p_i.

Quando um nó se move, consideramos que ele deixa de fazer parte de G. Posteriormente, quando ele pára de se movimentar reconectando-se à rede, ele é reintegrado à G. Desta forma, considera-se *nó móvel* aquele que se encontra fora de G, enquanto

[1] Os termos "processo" e "nó" serão usados indistintamente neste trabalho.

que um *nó estático* está conectado a G. Um nó pode continuamente se movimentar e se reconectar ou terminar por falhar. Entretanto, um nó correto irá sempre se reconectar à rede em algum momento. Seja p_m um nó móvel. Considera-se que p_m não tem conhecimento da sua mobilidade; assim, ele não pode notificar os seus vizinhos acerca do seu movimento. Desta forma, não é possível distinguir entre o movimento de p_m e a sua falha. Enquanto p_m se move, ele guarda os valores de suas variáveis.

Definição 1. Área de cobertura de difusão (*range*) : *Numa rede representada por $G(V, E)$, $range_i$ inclui p_i e o conjunto de seus vizinhos (*1- hop*). Neste caso, $|range_i|$ equivale ao grau de p_i em G mais 1. Observe que* ranges *são simétricos, i.e., $p_i \in range_j \Rightarrow p_j \in range_i$*

Definição 2. Densidade da área de cobertura (*range density*): *Numa rede representada por $G(V, E)$, a densidade da área de cobertura, denotada por d, é igual ao tamanho do menor* range *da rede:*

$$d \stackrel{def}{=} min(|range_i|), \forall p_i \in \Pi$$

Consideramos que d é conhecido por todos os processos.

Definição 3. Rede com f-cobertura (*f-covering network*): *Uma rede representada por $G(V, E)$ tem f-cobertura se e somente se G é $(f + 1)$-fortemente conexo.*

Pelo teorema de Menger [15], um grafo G é $(f + 1)$-fortemente conexo se e somente se ele contiver $f + 1$ caminhos independentes entre quaisquer dois nós p_i e p_j. Desta forma, removendo f nós de G ainda restará ao menos *um* caminho entre quaisquer dois nós. Além disso, a densidade da área de cobertura d da rede será sempre superior a $f + 1$, $d > f + 1$, o que nos leva à seguinte observação.

Observação 1. *Seja $G(V, E)$ uma rede com f-cobertura, então existe um caminho entre quaisquer dois nós em G, apesar de $f < n$ falhas.*

2.1. Detector de Falhas

Informalmente, um detector de falhas é um conjunto de "oráculos"que fornece dicas aos processos sobre quais deles estão falhos. O detector é não confiável porque ele pode se equivocar. Ou seja, tanto suspeitar da falha de processos corretos, quanto considerar que processos faltosos são corretos. A cada processo é associado um módulo de detecção, que fornece informações de falhas através de uma lista de processos. Um detector pode continuamente adicionar e remover processos de sua lista de suspeitos.

Formalmente, Chandra e Toueg caracterizam um detector de falhas por duas propriedades : *a completude* e a *exatidão* [3]. A *completude* assegura que processos faltosos terminarão por serem suspeitos enquanto que a *exatidão* restringe os equívocos que podem ser cometidos pelo detector. A combinação de tais propriedades dão origem a várias classes de detectores. Neste trabalho, nos concentramos na classe de *detectores de falhas forte após um tempo* (conhecido como $\diamondsuit S$). Esta classe garante que todo processo falho será finalmente suspeito por todos processos corretos (*completude forte*); além disto, existirá um instante a partir do qual algum processo correto não será considerado suspeito por nenhum outro processo correto (*exatidão fraca após um tempo*).

3. Detector de Falhas Assíncrono da Classe $\diamondsuit S$

Esta seção descreve nosso algoritmo de detecção de falhas assíncrono para redes desconhecidas. Para facilitar a apresentação, consideramos inicialmente que os nós não se movem e posteriormente, na seção 4, propomos uma extensão do algoritmo para suportar mobilidade.

3.1. O Mecanismo de Pergunta-Resposta (Query-Response)

O princípio básico de nosso algoritmo consiste na inundação na rede através de um mecanismo de *query-response* (pergunta-resposta) com informações sobre suspeitas de falhas de nós. O algoritmo se executa por rodadas. A cada nova rodada, caso não falhe, um nó envia a seus vizinhos uma mensagem QUERY. O intervalo de tempo entre duas rodadas consecutivas é finito mas arbitrário. Uma mensagem QUERY inclui dois tipos de informação: o conjunto de nós que são atualmente suspeitos de terem falho (*suspected*) e o conjunto de equívocos (*mistake*), ou seja, de nós erroneamente suspeitos de falhar nas rodadas precedentes. Cada nó possui um contador incrementado a cada rodada. Assim, toda nova informação sobre suspeita de falha ou equívoco é etiquetada com o valor corrente do contador deste nó. Tal mecanismo de etiquetagem evita que informações não atualizadas sejam consideradas válidas por outros nós.

Quando o nó p_i recebe uma mensagem QUERY de um nó da sua vizinhança, p_i lhe confirma a recepção com uma mensagem RESPONSE. Uma QUERY é então satisfeita pour un nó quando este receber pelo menos $d-f$ mensagens RESPONSE. Observe que cada par de *query-response* é identificado de modo único no sistema[2]. Além disso, consideramos que um nó enviará um nova pergunta somente depois que a precedente terminar. Evidentemente, um nó recebe a sua própria mensagem QUERY e a sua mensagem RESPONSE é sempre recebida dentre as primeiras $d-f$ respostas esperadas.

3.2. Propriedades Comportamentais

Definiremos agora duas propriedades comportamentais que asseguram que nossa proposta de implementação de um detector de falhas satisfaz as propriedades dos detectores da classe $\diamondsuit S$ para uma *rede desconhecida*.

Com o intuito de implementar um FD não-confiável cujos participantes são desconhecidos, é necessário que eles interajam entre si para se conhecerem. De acordo com Fernández *et al.* [4], se existir um processo no sistema cuja identidade nenhum outro processo conhece, então não existe algoritmo que possa implementar um detector de falhas que satisfaça a propriedade de *completude fraca*, mesmo se os canais forem confiáveis e o sistema síncrono. Consequentemente, para implementar um FD da classe $\diamondsuit S$, a seguinte *propriedade de inclusão*, denotada $\mathcal{MP}$, precisa ser satisfeita por todos os processos do sistema.

Propriedade 1. Propriedade de Inclusão (Membership) ($\mathcal{MP}$). *Seja $t \in \mathcal{T}$. Denotamos $known_j^t$ o conjunto de processos dos quais p_j recebeu uma mensagem* QUERY *no instante t. Seja K_i^t o conjunto de processos p_j, que antes ou no instante t, recebeu uma mensagem* QUERY *de p_i. Isto é, $K_i^t = \{p_j \mid p_i \in known_j^t\}$. Um processo p_i satisfaz a propriedade de membership se:*

$$\mathcal{MP}(p_i) \stackrel{def}{=} \exists t \geq 0 \in \mathcal{T} : |K_i^t| > f + 1$$

[2]Por uma questão de simplificação, esta identificação não aparece no código do algoritmo.

Esta propriedade afirma que, para ser membro do sistema, um processo p_i (correto ou não) precisa interagir pelo menos uma vez com outros processos da sua cobertura ($range_i$), enviando-lhes uma mensagem QUERY por difusão. Além disso, esta mensagem deve ser recebida e figurar no estado de pelo menos um processo correto, além de p_i.

Dado que nosso detector é assíncrono e não depende de temporizadores, uma segunda propriedade importante que precisamos definir é a *propriedade de receptividade*, denotada $\mathcal{RP}$, que expressa a capacidade de um nó em responder entre os primeiros processos a uma pergunta feita.

Propriedade 2. Propriedade de Receptividade (Responsiveness) ($\mathcal{RP}$). *Seja $t, u \in \mathcal{T}$. Denotamos $rec\_from_j^t$ o conjunto de $d - f$ processos dos quais p_j recebeu respostas à mensagem (*QUERY*) que se terminou antes ou no instante t. A propriedade $\mathcal{RP}$ do processo correto p_i é assim definida:*

$$\mathcal{RP}(p_i) \stackrel{def}{=} \exists u \in \mathcal{T} : \forall t > u, \forall p_j \in range_i,\ p_i \in rec\_from_j^t$$

Intuitivamente, a propriedade $\mathcal{RP}(p_i)$ afirma que depois de um intervalo de tempo finito u, o conjunto de $d - f$ respostas recebidas por um vizinho de p_i à sua última mensagem QUERY sempre inclui a resposta de p_i.

3.3. Implementação de um Detector de Falhas $\diamondsuit S$ para Redes Desconhecidas

O Algoritmo 1 descreve a implementação do nosso detector de falhas da classe $\diamondsuit S$. Consideramos que o sistema subjacente é uma rede com *f-cobertura* que satisfaz as propriedades comportamentais descritas anteriormente. A seguinte notação é utilizada.

• $counter_i$: denota o contador de rodadas do processo p_i.
• $suspected_i$: denota o conjunto atual de processos suspeitos de serem falhos por p_i. Cada um de seus elementos é composto por um par $\langle id, counter\rangle$, onde id representa o identificador do nó suspeito e $counter$, o valor do contador do processo, quando este gerou a informação de suspeita de falha do nó id.
• $mistake_i$: denota o conjunto de nós equivocadamente suspeitos nas rodadas anteriores. Tal qual no conjunto $suspected_i$, $mistake_i$ também é composto por pares $\langle id, counter\rangle$, onde $counter$ indica a rodada cuja informação sobre a correção do equívoco a respeito da falha de id foi gerada.
• rec_from_i: denota o conjunto de nós dos quais p_i recebeu uma resposta (RESPONSE) à sua última pergunta QUERY.
• $known_i$: denota o conhecimento atual que o nó p_i tem de sua vizinhança, ou seja, $known_i$ representa o conjunto de processos dos quais p_i recebeu uma mensagem QUERY desde o começo de sua execução.
• $Add(set, \langle id, counter\rangle)$: função que inclui o par $\langle id, counter\rangle$ em set. Se o par $\langle id, -\rangle$ já existe em set, ele é substituído por $\langle id, counter\rangle$.

Nosso algoritmo é composto de duas tarefas. A tarefa $T1$ é responsável pela geração das suspeitas e é constituída de um laço infinito. A tarefa $T2$ é responsável pela geração de informações de revogação de suspeitas (equívocos) e pela propagação das informações recentes na rede.

Geração de Suspeitas de Falhas. A cada rodada da tarefa T1, uma mensagem QUERY é enviada a todos os nós da área de cobertura de difusão ($range_i$) de p_i (linha 6). Esta mensagem inclui o conjunto de nós suspeitos atualmente por p_i ($suspected_i$) e o conjunto

de equívocos conhecidos por p_i ($mistake_i$). O processo p_i então espera por pelo menos $d - f$ respostas à sua pergunta, além da sua própria (linha 7).

Algorithm 1 Implementação Assíncrona de um Detector de Falhas

```
 1: init:
 2: suspected_i ← ∅; mistake_i ← ∅; counter_i ← 0
 3: known_i ← ∅

 4: Task T1:
 5: repeat forever
 6:     broadcast QUERY(suspected_i, mistake_i)
 7:     wait until RESPONSE received from at least (d − f) distinct processes
 8:     rec_from_i ← the set of distinct nodes from which p_i has received a response at line 7
 9:     for all p_j ∈ known_i \ rec_from_i | ⟨p_j, −⟩ ∉ suspected_i do
10:         if ⟨p_j, counter⟩ ∈ mistake_i then
11:             counter_i = max(counter_i, counter + 1)
12:             mistake_i = mistake_i \ ⟨p_j, −⟩
13:         end if
14:         Add(suspected_i, ⟨p_j, counter_i⟩)
15:     end for
16:     counter_i = counter_i + 1
17: end repeat

18: Task T2:
19: upon reception of QUERY (suspected_j,mistake_j) from p_j do
20: known_i ← known_i ∪ {p_j}
21: for all ⟨p_x, counter_x⟩ ∈ suspected_j do
22:     if ⟨p_x, −⟩ ∉ suspected_i ∪ mistake_i or ⟨p_x, counter⟩ ∈ suspected_i ∪ mistake_i | counter <
    counter_x then
23:         if p_x = p_i then
24:             counter_i = max(counter_i, counter_x + 1)
25:             Add(mistake_i, ⟨p_i, counter_i⟩)
26:         else
27:             Add(suspected_i, ⟨p_x, counter_x⟩)
28:             mistake_i = mistake_i \ ⟨p_x, −⟩
29:         end if
30:     end if
31: end for
32: for all ⟨p_x, counter_x⟩ ∈ mistake_j do
33:     if ⟨p_x, −⟩ ∉ suspected_i ∪ mistake_i or ⟨p_x, counter⟩ ∈ suspected_i ∪ mistake_i | counter ≤
    counter_x then
34:         Add(mistake_i, ⟨p_x, counter_x⟩)
35:         suspected_i = suspected_i \ ⟨p_x, −⟩
36:     end if
37: end for
38: send RESPONSE to p_j
```

As linhas 9-15 são responsáveis pela detecção de novas suspeitas de falhas. O processo p_j será suspeito de falhar por p_i se: (i) p_i conhece p_j ($p_j \in known_i$), (ii) p_i ainda não suspeitou anteriormente do mesmo ($p_j \notin suspected_i$) e (iii) p_i não recebeu de p_j um RESPONSE como resposta à sua última QUERY. Nestas condições, se uma informação de *mistake* referente a p_j existe no conjunto $mistake_i$, ela será removida (linha 12). Além disso, o contador da rodada $counter_i$ é atualizado com um valor superior ao contador do

mistake em questão (linha 11). Finalmente, a nova informação de suspeita é incluída no conjunto $suspected_i$ com uma etiqueta associada cujo valor é equivalente ao de $counter_i$ (linha 14). No final da tarefa $T1$, $counter_i$ é incrementado de uma unidade (linha 16).

Propagação de Suspeitas e Equívocos. A tarefa $T2$ permite a um nó tratar a recepção de mensagens de QUERY enviadas por nós vizinhos. Os dois laços da tarefa tratam respectivamente as informações recebidas referentes a suspeitas de falhas (linhas 21–31) e as referentes a equívocos (linhas 32–37). Para cada nó p_x existente no conjunto $suspected_j$ (respectivamente, $mistake_j$) da mensagem QUERY recebida de p_j, p_i inclui p_x no seu conjunto $suspected_i$ (respectivamente, $mistake_i$) somente se a seguinte condição é satisfeita: p_i recebeu uma informação sobre o status (falha ou equívoco) de p_x que é mais recente do que aquela que ele já possui em seus conjuntos $suspected_i$ e $mistake_i$. Uma informação será mais recente se: (i) p_x nunca foi suspeito por p_i ou (ii) p_x pertence a um dos conjuntos de p_i, porém com um valor de etiqueta ($counter$) inferior aquele trazido por p_j ($counter_x$) na sua QUERY (linhas 22 e 33). Neste caso, p_i remove também o nó p_x de seu conjunto $mistake_i$ (respectivamente $suspected_i$) (linhas 28 e 35).

Revogação de Equívocos. Caso o processo p_i pertença ao conjunto de suspeitos recebido na mensagem QUERY (linha 23) da tarefa $T2$, ele gerará um novo equívoco, ou seja, ele se adiciona em seu conjunto $mistake_i$ (linha 25) com um valor de contador mais recente do que aquele associado à sua suspeita (linha 24). No final da tarefa $T2$ (linha 38), p_i envia ao emissor da mensagem de QUERY uma mensagem de RESPONSE.

4. Extensão do Detector de Falhas para Suportar a Mobilidade dos Nós

Nesta seção apresentamos uma extensão ao Algoritmo 1 capaz de suportar a mobilidade dos nós. Para tanto, algumas novas propriedades comportamentais referentes à mobilidade e ao sistema subjacente necessitam ser definidas.

4.1. Propriedades Comportamentais para a Mobilidade

Seja p_m um nó móvel. Durante a execução, ele pode continuamente se deslocar e se reconectar ou acabar por falhar. Entretanto, para que p_m possa atualizar o seu estado com informações recentes relativas às falhas e equívocos, p_m precisa se conectar à rede por um período suficiente de tempo. Caso contrário, não há garantia que as propriedades de *completude* e *exatidão* do detector possam ser asseguradas. Para podermos capturar esta noção de "período suficiente de tempo de conexão", definimos a seguinte *propriedade de mobilidade*, denotada $\mathcal{M}obi\mathcal{P}$, que deverá ser satisfeita por todos os nós móveis da rede:

Propriedade 3. Propriedade de Mobilidade ($\mathcal{M}obi\mathcal{P}$). *Seja $t \in \mathcal{T}$. Seja Q_i^t o conjunto de processos dos quais p_i recebeu uma mensagem de* QUERY *que se terminou antes ou no instante t. Um processo p_i satisfaz a propriedade de mobilidade se:*

$$\mathcal{M}obi\mathcal{P}(p_i) \stackrel{def}{=} \exists t \geq 0 \in \mathcal{T} : |Q_i^t| > f + 1$$

$\mathcal{M}obi\mathcal{P}(p_m)$ assegura que, depois de ter se reconectado à rede, haverá um tempo no qual p_m terá recebido mensagens QUERY dos processos da sua nova vizinhança e dentre estas, pelo menos uma foi enviada por um processo correto, diferente de p_m. Dado que estas mensagens contêm informações sobre suspeitas de falhas e equívocos, p_m poderá atualizar corretamente seu estado.

Consideramos ainda que os nós móveis também satisfazem a *propriedade de inclusão*. $\mathcal{MP}(p_m)$ assegura que, depois de se reconectar, haverá um instante a partir do

qual p_m interagirá pelo menos uma vez com outros processos de sua nova vizinhança ($range_m$), enviando-lhes por difusão uma mensagem QUERY que será recebida por pelo menos um processo correto do $range_m$, além do próprio p_m.

Será necessário estender a propriedade $\mathcal{RP}$ de forma a garantir que existe um instante de tempo a partir do qual os vizinhos de un nó p_i, que satisfaz $\mathcal{RP}(p_i)$, não sairão mais do seu $range_i$. Caso contrário, mesmo que $\mathcal{RP}(p_i)$ seja satisfeita, um nó móvel p_m poderá adicionar p_i no seu conjunto $known_m$, caso $p_i \in range_m$, e posteriormente, poderá suspeitar de p_i, caso tenha se movimentado e p_i não faça mais parte da sua vizinhança, ou seja $p_i \notin range_m$. A extensão da propriedade $\mathcal{RP}$, denotada $\mathcal{M}obi\mathcal{RP}$, é assim definida:

Propriedade 4. Propriedade de Receptividade para Mobilidade (Mobility Responsiveness) ($\mathcal{M}obi\mathcal{RP}$). *Seja $t \in \mathcal{T}$. Denotamos $range_i^t$ o conjunto de processos em $range_i$ no instante t. Um processo p_i satisfaz a propriedade receptividade para mobilidade se:*

$$\mathcal{M}obi\mathcal{RP}(p_i) \stackrel{def}{=} \mathcal{RP}(p_i) : \exists u \in \mathcal{T} : \forall t > u, \forall t' > t, p_j \in range_i^t \Rightarrow p_j \in range_i^{t'}$$

$\mathcal{M}obi\mathcal{RP}$ precisa ser assegurada por pelo menos um nó correto da rede que nunca se move. Com relação ao comportamento do sistema subjacente, considera-se que a rede satisfaz a propriedade de *f-cobertura* e que a sua densidade d será preservada, apesar da mobilidade.

4.2. Implementação de un Detector de Falhas da Classe $\diamond S$ para Redes Móveis Desconhecidas

A nossa extensão para o Algoritmo 1 com suporte à mobilidade dos nós é baseada no mesmo princípio de *query-response*, apresentado na seção 3. Quando um nó p_m move-se para um outro *range*, ele começará a ser suspeito de falhar por aqueles nós de seu antigo *range*, visto que ele não responde mais à mensagens QUERY desses nós. A informação sobre a suspeita de falha de p_m será disseminada pela rede através de mensagens QUERY. Quando p_m se reconectar à rede, ele acabará por receber essas mensagens. Ele corrigirá a falsa suspeita que lhe concerne ao incluir-se no seu conjunto $mistake_m$, que por sua vez, será incluído na próxima mensagem QUERY que ele enviará. Tal informação se propagará pelos nós da rede. Por outro lado, p_m começará a suspeitar dos nós de seu antigo *range*, pois estes se encontram em seu conjunto $known_m$. Ele então irá incluir estas suspeitas na sua próxima mensagem QUERY. Esta, acabará por ser recebida pelos respectivos nós, que através do mesmo princípio de inclusão da sua identidade no conjunto $mistake$ irão corrigir o equívoco. Finalmente, os *mistakes* correspondentes serão então disseminados pela rede e a suspeita será revogada.

Para evitar um efeito"ping-pong" entre as informações de suspeitas de falhas e revogações das mesmas, que advirão desse procedimento acima, um nó deve ser capaz de remover de seu conjunto $known$ aqueles nós que não fazem mais parte de seu *range*. Para tanto, as linhas 36–38 do Algoritmo 2 devem ser adicionadas no bloco **if** do segundo laço de $T2$, logo depois da linha 35 do Algoritmo 1. Elas possibilitam a atualização do conjunto $known_m$ de p_m, assim como do conjunto $known$ dos nós de seu antigo *range*. Para cada *mistake* $\langle p_x, counter_x \rangle$ enviado por p_j tal que p_i contém um informação menos atualizada sobre p_x, p_i verifica se p_x é o nó emissor do *mistake* em questão. Em caso negativo ($p_x \neq p_i$), p_x pode pertencer a um *range* distante, tal que $p_x \notin range_i$. Consequentemente, p_x é excluído do conjunto $known_i$.

Algorithm 2 Implementação Assíncrona de um Detector de Falhas com Mobilidade

36: **if** ($p_x \neq p_j$) **then**
37: $known_i = known_i \setminus \{p_x\}$
38: **end if**

4.3. Prova de Correção

Apresentamos nesta sub-seção os principais argumentos da prova de que os Algoritmos 1 e 2 implementam um detector da classe $\diamondsuit S$. A demonstração completa desta corretude pode ser encontrada no relatório técnico [11].

Teorema 1. *Algoritmos 1 e 2 implementam um detector de falhas da classe $\diamondsuit S$ numa rede de nós móveis, que apresenta a propriedade de f-cobertura e que satisfaz as seguintes propriedades comportamentais: $\mathcal{RP}$, $\mathcal{MP}$, $\mathcal{M}obi\mathcal{P}$ e $\mathcal{M}obi\mathcal{RP}$.*

Demonstração. Considere um processo correto p_i e um processo falho p_f.

Para satisfazer a propriedade de *completude forte*, devemos provar que eventualmente p_f será permanentemente incluído no conjunto $suspected_i$ de p_i. Esta condição segue principalmente das seguintes constatações:
(1) Dado que p_f satisfaz $\mathcal{MP}(p_f)$, ele enviou ao menos uma mensagem QUERY e faz parte do conjunto $known_i$ de pelo menos um processo correto p_i;
(2) Logo, após a sua falha, p_f não irá emitir mais mensagens de RESPONSE e passará a ser suspeito por p_i, ou seja $p_f \in suspected_i$. Finalmente, essa informação será propagada na rede, dado que a mesma apresenta a propriedade de *f-cobertura*.

Para satisfazer a propriedade de *exatidão fraca após um tempo*, devemos provar que haverá um instante de tempo u após o qual p_i não pertence a nenhum conjunto $suspected_j$ de qualquer processo correto p_j. Esta condição segue principalmente de:
(1) Dado que existe ao menos um processo correto p_i que satisfaz $\mathcal{M}obi\mathcal{RP}(p_i)$, existe um instante de tempo t após o qual todos os processos corretos p_j de $range_i$ recebem o QUERY de p_i e além disso, nenhum deles se movimenta, além de $range_i$. Logo, a partir de t, $p_i \in rec\_from_j$ de qualquer processo correto p_j e nenhum deles irá suspeitar de p_i;
(2) Além disso, como p_i é correto, caso ainda existam informações de suspeitas sobre o mesmo após t, ele acabará por revogar com uma etiqueta (contador) superior, emitindo um QUERY, contendo $mistake_i$, que será recebido pelos vizinhos, pois $\mathcal{RP}(p_i)$. Finalmente, essa informação será propagada, pois a rede apresenta a propriedade de *f-cobertura*. Assim, haverá um tempo $u \geq t$, em que $p_i \notin suspected_j$, para qualquer p_j correto. □

5. Implementação e Avaliação de Desempenho do Detector de Falhas

Esta seção apresenta um estudo de avaliação de desempenho da implementação de nosso detector de falhas assíncrono. Os resultados obtidos são comparados com os apresentados pelo detector de Friedman e Tcharny proposto em [6]. O intuito dessa análise comparativa é fazer um contraponto entre nosso detector assíncrono e um detector baseado em mecanismos de temporização e propagação de mensagens do tipo "eu estou vivo"(*heartbeats*).

Modelo de Simulação. Nossos testes de desempenho foram efetuados com o simulador de eventos discretos OMNeT++ [1]. A área de simulação abrange um retângulo R de 700mx700m com $N = 100$ nós. Cada nó tem um raio de alcance de $r = 100m$ e o tempo de transferência de uma mensagem δ entre dois nós vizinhos é em média igual a $1ms$.

O tempo de simulação de cada teste foi de 30 minutos. Como nosso detector de falhas necessita de uma rede que satisfaz a *f_cobertura*, os N nós não podem ser distribuídos aleatoriamente no retângulo R. Antes da execução do teste, a topologia da rede é construída de forma gradual: um "clique" de grafo com $f + 2$ nós organizados em círculo com raio $r/2$ é inicialmente inserido em R; a cada nova iteração, um novo nó é escolhido aleatoriamente e este é incluído na rede somente se possuir $f+1$ vizinhos na configuração corrente. A fase de configuração é concluída quando a rede atinge N nós.

Detector de falhas proposto por Friedman e Tcharny. Neste detector, um nó periodicamente envia um mensagem de *heartbeat* a seus vizinhos contendo um vetor de N elementos que correspondem aos nós da rede. Cada nó p possui também um vetor local ($vector_p$) cujas N entradas armazenam o mais alto valor de *heartbeat*, enviado pelo nó correspondente, que p conhece. A cada Δ intervalo de tempo, p incrementa $vector_p[p]$ e emite a seus vizinhos a sua mensagem de *heartbeat* por difusão. Com base nos testes de desempenho descritos pelos autores em [6], fixamos o valor de Δ à $1s$. Ao receber uma mensagem de *heartbeat*, p atualiza seu $vector_p$ com o máximo entre este e o incluído na mensagem. Um nó associa também um *temporizador* a cada nó da rede. Assim, p inicializa o temporizador correspondente ao nó q com Θ unidades de tempo ao receber uma nova informação sobre q. Entretanto, se Θ expira, p considera q suspeito de falha. Para o cálculo do valor de Θ é necessário considerar o tempo de transmissão entre caminhos mais longos entre dois nós. Por esta razão, o valor de Θ foi fixado em $2s$ em nossos testes.

Detector de falhas assíncrono. Em relação à implementação de nosso detector, se os nós enviarem continuamente mensagens QUERY por difusão a seus vizinhos, a rede ficará sobrecarregada. Para evitar tal problema, incluímos um intervalo de tempo de Δ unidades de tempo entre as linhas 7 and 8 do Algoritmo 1. Como na implementação de Friedman and Tcharny, nós fixamos Δ a $1s$. Entretanto, ao adicionar este intervalo de espera, um nó pode acabar por receber mais do que $d - f$ respostas, que serão então também incluídas no conjunto rec_from deste nó (linha 8), o que reduzirá a quantidade de suspeitas falsas. Vale ressaltar que tal otimização não compromete a corretude do algoritmo.

5.1. Avaliação da Detecção de Falhas

Para avaliar a propriedade de *completude* de ambos os detectores, medimos o impacto da variação da densidade d da rede nos respectivos tempos de detecção de falha dos dois detectores (vide Figura 1). O *tempo de detecção de falha* consiste no intervalo entre o instante em a falha aconteceu e o instante em que ela é permanentemente detectada por todos os processos corretos. Cinco falhas foram uniformemente injectadas durante a duração de cada teste. A densidade d varia entre 7 e $N/2$ nós. Para cada densidade, o tempo *médio*, *máximo* e *mínimo* de detecção foram medidos.

Em ambos os detectores, a propagação das mensagens é consideravelmente rápida pois o diâmetro da rede é relativamente pequeno. No caso do detector de falhas de Friedman e Tcharny, o tempo médio de detecção de falha se encontra sempre entre o mínimo de $\Theta - \Delta$ e o máximo de Θ, independent de d pois a detecção de falhas é feita com base nos valores dos vetores e temporizadores. Estes valores de mínimo e máximo podem ser facilmente justificados. Se o nó q falha logo depois que o nó p inicializou o temporizador referente à q com Θ, p detectará a falha de p após Θ unidades de tempo; se q falhar justo antes de emitir a mensagem de *heartbeat*, i.e. após Δ unidades de tempo, p detectará sua falha após $\Theta - \Delta$ unidades de tempo. Entretanto, no caso de nosso detector, o tempo

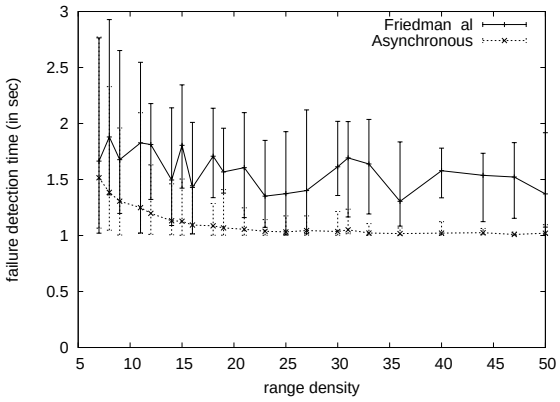

Figura 1. Tempo de Detecção de Falha X Densidade da Rede

de detecção de falhas diminui com a densidade d pois informações sobre a detecção de falhas são incluídas nas mensagens QUERY que se propagam mais rapidamente na rede quando a densidade aumenta. Para valores de d superiores a 22, o tempo de detecção de falhas é uniforme e equivalente em média a $\Delta + \delta$.

O tempo máximo de detecção de uma falha caracteriza o tempo necessário para que todos os nós possam detectar a falha (completude forte). Comparado com o detector de Friedman e Tcharny, este tempo é menor e homogêneo para o nosso detector devido à propagação de informações de falhas em mensagens de QUERY.

5.2. Impacto da Mobilidade

Avaliamos também a propriedade de *exatidão* quando um nó p_m, localizado numa das extremidades da rede, move-se por volta de $500m$ a uma velocidade de $2m/s$. O nó começa a se mover no instante $100s$. Ele possui originalmente 7 vizinhos e cada um destes possui $d-f+1$ vizinhos. Esta restrição é necessária afim de garantir que ao menos $d-f$ vizinhos responderão à mensagem QUERY desses vizinhos após a movimentação de p_m. Consideramos também que durante seu movimento, p_m não interage com os outros nós (é como se ele atravessasse uma área de instabilidade aonde nenhuma comunicação é possível). Consequentemente, ele pára sua execução. A densidade d da rede é igual a 7 e não há falhas.

A figura 2 mostra o número total de suspeitas falsas no intervalo tempo justo antes e depois que o nó p_m cessa de se movimentar no instante $356s$. Observamos que os $N-1$ outros nós suspeitam falsamente p_m antes deste instante para ambos os detectores. Entretanto, após este instante, essas suspeitas começam a ser corrigidas. No caso do detector de Friedman e Tcharny, não há mais suspeitas falsas depois de $1.5s$. No caso do nosso detector, elas começam também a serem corrigidas pois p_m genera um *mistake* que é propagado pela rede. Por outro lado, o nó p_m começa também a suspeitar seus vizinhos antigos e consequentemente envia estas suspeitas na sua próxima mensagem de QUERY a seus vizinhos. Esta informação se propagará pela rede o que explica que o número total de suspeitas falsas aumenta entre os instantes $375s$ e $358s$. Porém, neste instante, os 7 vizinhos antigos de p_m também recebem a informação sobre suas suspeitas. Estes generam então os *mistakes* correspondentes e os enviam pela rede nas suas próximas mensagens QUERY, que serão disseminadas pela rede. Todas as suspeitas falsas serão corrigidas por todos os nós no instante $359.5s$.

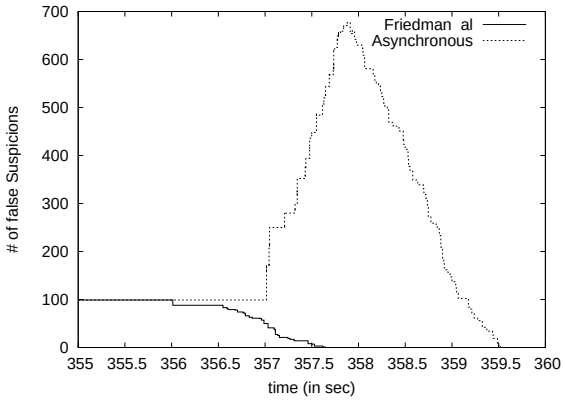

Figura 2. Número Total de Suspeitas Falsas

6. Trabalhos Relacionados

Analogamente ao nosso enfoque, encontramos na literatura alguns detectores de falhas não-confiáveis, cuja implementação é extensível e não considera que um nó se comunica com todos os outros do sistema. Na implementação do FD apresentada por Larrea *et al.* [9], os nós estão organizados em um anel lógico. O número de mensagens é então linear, mas o tempo para a propagação de informação sobre as falhas é bastante alto. Em [8], Gupta *et al.* propõem um algoritmo para detecção de falhas onde cada processo escolhe aleatoriamente alguns outros afim de monitorar sua vivacidade. Assim, há um balanceamento da carga de mensagens na rede. Entretanto, na prática, a opção por escolha aleatória torna difícil a definição de valores para os temporizadores utilizados. Em [2], os autores introduzem um FD extensível e hierárquico para grades computacionais. Porém, a configuração inicial da rede é conhecida por todos os nós. Vale ressaltar que nenhum destes trabalhos suporta mobilidade dos nós.

Algumas implementações de FDs não-confiáveis são adequadas para redes MANETs. Entretanto, todos elas são baseadas no uso de temporizadores. No algoritmo de Friedman and Tcharny [6], descrito na seção 5, os autores assumem que o número de nós da rede é conhecido e que falhas devidas à omissão de mensagens podem occorrer. Em [14], uma arquitetura baseada em *clusters* para redes ad-hoc sem fios é proposta afim de oferecer um serviço de detecção de falhas que tolera perda de mensagens e falhas de nós por parada. Sridhar apresenta em [13] uma implementação hierárquica de FDs que possui dois níveis independentes: um local, que fornece a lista de nós vizinhos suspeitos de falha e um segundo, que detecta a mobilidade dos nós, corrigindo assim possíveis equívocos. Contrariamente ao nosso detector que satisfaz a classe de detectores $\diamondsuit S$, o detector proposto por Sridhar satisfaz a classe de *detectores perfeito após um tempo* ($\diamondsuit P$), considerando-se apenas a vizinhança de cada nó.

Greve *et al.* em [7] oferecem uma solução para o consenso tolerante a falhas em redes desconhecidas e dinâmicas. Tal consenso, de nome FT-CUP, irá exigir do sistema um maior grau de conectividade entre os participantes, mas poderá ser resolvido com requisitos minimais de sincronia ($\diamondsuit S$). Nosso detector de falhas assíncrono poderá ser utilizado na implementação do FT-CUP sobre redes MANET.

7. Conclusão

Apresentamos neste trabalho uma nova implementação de detector de falhas não-confiáveis para redes dinâmicas como as redes MANETs, cujo número de nós não é

conhecido e a conectividade da rede não é completa. O algoritmo proposto baseia-se no mecanismo de QUERY-RESPONSE que não depende de temporizadores para detectar falhas. Consideramos que a rede possui a propriedade de *f-cobertura*, aonde f é o número máximo de falhas, o que garante que existe sempre um caminho entre quaisquer dois nós da rede, apesar das falhas. Nosso algoritmo implementa detectores de falhas da classe $\Diamond S$ quando as propriedades comportamentais de receptividade ($\mathcal{RP}$, $\mathcal{M}obi\mathcal{RP}$), inclusão ($\mathcal{MP}$) e mobilidade ($\mathcal{M}obi\mathcal{P}$) são satisfeitas pelo sistema subjacente. Resultados de simulação validaram a eficácia do novo algoritmo.

Referências

[1] *OMNet++ Discret Event Simulation System*. http://www.omnetpp.org.

[2] M. Bertier, O. Marin, and P. Sens. Performance analysis of a hierarchical failure detector. In *Proc. of the Int. Conf. on Dependable Systems and Networks*, San Francisco, CA, USA, june 2003.

[3] T. D. Chandra and S. Toueg. Unreliable failure detectors for reliable distributed systems. *Journal of the ACM*, 1996.

[4] A. Fernández, E. Jiménez, and S. Arévalo. Minimal system conditions to implement unreliable failure detectors. In *Proc. of the 12th Int. Symposium Pacific Rim Dependable Computing*, pages 63–72. IEEE Computer Society, 2006.

[5] M. J. Fischer, N. A. Lynch, and M. S. Paterson. Impossibility of distributed consensus with one faulty process. *Journal of the ACM*, 32(2):374–382, apr 1985.

[6] R. Friedman and G. Tcharny. Evaluating failure detection in mobile ad-hoc networks. *Int. Journal of Wireless and Mobile Computing*, 1(8), 2005.

[7] F. Greve and S. Tixeuil. Knowledge conectivity vs. synchrony requirements for fault-tolerant agreement in unknown networks. In *Proc. of the Int. Conf. on Dependable Systems and Networks*, Edinburgh, UK, June 2007.

[8] I. Gupta, T. D. Chandra, and G. S. Goldszmidt. On scalable and efficient distributed failure detectors. In *Proc. of the twentieth annual ACM symposium on Principles of distributed computing*, pages 170–179. ACM Press, 2001.

[9] M. Larrea, A. Fernández, and S. Arévalo. Optimal implementation of the weakest failure detector for solving consensus. In *Proc. of the 19th Annual ACM Symposium on Principles of Distributed Computing*, pages 334–334, NY, July 16–19 2000. ACM Press.

[10] A. Mostefaoui, E. Mourgaya, and M. Raynal. Asynchronous implementation of failure detectors. In *Proc. of Int. Conf. on Dependable Systems and Networks*, June 2003.

[11] P. Sens, L. Arantes, M. Bouillaguet, V. Simon, and F. Greve. Implementation asynchrone de detecteurs de fautes sans connaître les participants et en connectivité partielle (in english). Technical Report TR6088, INRIA - France, 1, 2007.

[12] I. Sotoma and E. Madeira. Adaptation - algorithms to adaptative fault monitoring and their implementation on corba. In *Proc. of the IEEE 3rd Int. Symposium on Distributed Objects and Applications*, pages 219–228, september 2001.

[13] N. Sridhar. Decentralized local failure detection in dynamic distributed systems. *The 25th IEEE Symposium on Reliable Distributed Systems*, 0:143–154, 2006.

[14] A. Tai, K. Tso, and W. Sanders. Cluster-based failure detection service for large-scale ad hoc wireless network applications. In *Proc. of the Int. Conf. on Dependable Systems and Networks*, pages 805–814, New York City, USA, June 2004. IEEE Computer Society Press.

[15] Jay Yellen and Jonathan L. Gross. *Graph Theory & Its Applications*. CRC Press, 1998.