Protocolo Assíncrono para Detecção de Falhas Bizantinas em Sistemas Distribuídos Dinâmicos*

Murilo Santos de Lima[1,2], **Fabíola Gonçalves Pereira Greve**[1]

[1]Departamento de Ciência da Computação – Universidade Federal da Bahia (UFBA)
Av. Adhemar de Barros, S/N, Campus de Ondina – 40170-110 – Salvador – BA – Brazil

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

murilolima87@gmail.com, fabiola@dcc.ufba.br

Resumo. *Em um sistema distribuído dinâmico, no qual os nós podem entrar e sair da rede aleatoriamente, o desafio de implementar serviços confiáveis é grande. Nesse contexto, um fator preocupante, em especial, é a segurança. Detectores de falhas bizantinas são uma solução elegante para problemas de segurança, uma vez que separam o tratamento das falhas do protocolo distribuído que os utiliza. No entanto, desconhecem-se trabalhos na literatura descrevendo soluções específicas para sistemas dinâmicos. Este artigo propõe um detector de falhas bizantinas para tais sistemas. Adicionalmente, o protocolo apresentado é assíncrono, isto é, não se baseia no uso de temporizadores para a detecção das falhas, o que favorece sua escalabilidade e adaptabilidade.*

Abstract. *Byzantine failure detectors provide an elegant solution for security problems. However, as far as we know, there is no specific solution for this problem in a dynamic distributed system. This paper presents a first Byzantine failure detector for this context. Besides, the protocol presented is asynchronous, that is, the failure detection process does not rely on timers to make suspicions. This characteristic favors scalability and adaptability.*

1. Introdução

Um sistema distribuído dinâmico é composto por uma população de nós[1] que podem entrar e sair da rede aleatoriamente, a qualquer momento da execução do sistema. Comumente, as seguintes características são apresentadas: (1) a latência na transmissão das mensagens é imprevisível, (2) a rede não está totalmente conectada, (3) não existem elementos centralizadores na rede, (4) não é possível prover aos processos uma visão global da topologia da rede, de forma que cada nó tem um conhecimento parcial da composição do sistema e (5) os nós podem alterar sua posição quanto à topologia da rede. Constata-se, portanto, que os protocolos distribuídos clássicos, que supõem uma rede com composição estática e conhecida, não são mais adequados neste novo contexto, característico das redes entre pares (P2P), redes móveis auto-organizáveis (Manets) e redes de sensores sem fio [Mostefaoui et al. 2005].

*Murilo é Bacharel em Ciência da Computação pela Universidade Federal da Bahia e este trabalho é resultado do seu projeto de final de curso. O trabalho tem apoio da FAPESB-Bahia e CNPQ-Brasil.

[1]Neste trabalho, os termos "nó" e "processo" serão utilizados indistintamente.

Um fator preocupante em sistemas distribuídos dinâmicos, em especial, é a segurança. A dinamicidade da população dos nós e o uso comum de redes sem fio ou da Internet como meios de comunicação facilitam a ação de agentes maliciosos sobre o sistema. O modelo de falhas bizantinas [Lamport et al. 1982] lida com este tipo de ataque ao considerar a existência de processos corruptos, que podem se comportar de maneira arbitrária na tentativa de impedir que o sistema funcione conforme a sua especificação. Um processo bizantino pode, por exemplo, tentar assumir a identidade de outro, enviar mensagens com valores incorretos, duplicar mensagens ou simplesmente não enviar mensagens que o protocolo em execução especificar. O desenvolvimento de sistemas tolerantes a faltas bizantinas, isto é, que mantenham o seu funcionamento correto apesar do comportamento maligno de alguns de seus processos, é portanto de especial interesse.

A abstração de detectores de falhas [Chandra and Toueg 1996] provê uma forma modular de tratar as falhas em sistemas assíncronos, isto é, sistemas que não atendam a restrições temporais. O detector separa o tratamento das falhas e os requisitos de sincronia do protocolo distribuído que o utiliza, de forma que este pode lidar apenas com a tarefa a que se propõe. O problema do consenso, por exemplo, que sumariza diversos problemas de acordo distribuído, não pode ser resolvido num sistema sem quaisquer suposições de sincronia [Fischer et al. 1985], mesmo se apenas um processo falhar por colapso (*crash*) [Correia et al. 2006]. Se o sistema dispõe, no entanto, de um detector da classe $\Diamond\mathcal{S}$ [Chandra and Toueg 1996], o consenso pode ser resolvido para falhas por colapso sem lidar diretamente com questões de sincronia.

A grande maioria das implementações para detectores de falhas considera a existência de uma rede estática, cuja composição dos nós é conhecida [Larrea et al. 2000]. Mais recentemente, algumas propostas têm sido feitas para sistemas móveis e auto-organizáveis. [Gupta et al. 2001] considera uma população dinâmica dos nós, já [Friedman and Tcharny 2005] tolera a mobilidade dos nós. Todas essas propostas utilizam um mecanismo de temporização para detecção das falhas, i.e., os nós monitoram a vivacidade dos demais nós da rede através da troca de mensagens do tipo “eu estou vivo”.

Detectores de falhas assíncronos não fazem uso do recurso de temporização. Nessa linha, um trabalho pioneiro foi apresentado por [Mostefaoui et al. 2003]. Entretanto, ele considera uma rede estática com composição dos nós conhecida. Em [Sens et al. 2008] apresenta-se uma implementação assíncrona de um detector de falhas que trata tanto a dinamicidade da composição da rede quanto a mobilidade dos nós. Todos esses trabalhos, no entanto, são focados na detecção de falhas por colapso, i.e., quando um processo se comporta de maneira benigna até falhar e haver o término da execução.

Alguns trabalhos [Kihlstrom et al. 2003, Baldoni et al. 2007] são voltados para detecção de falhas bizantinas. Entretanto, nenhum deles lida com a dinamicidade do sistema e mobilidade dos seus nós. Além disso, todos fundamentam-se no uso de temporizadores. Os autores desconhecem, portanto, trabalhos que proponham detectores de falhas bizantinas específicos para sistemas distribuídos dinâmicos e também desconhecem detectores de falhas bizantinas que sejam assíncronos. Existem trabalhos voltados para problemas específicos, como é o caso de [Awerbuch et al. 2002], que resolve a detecção para o roteamento, ou então, só toleram falhas de omissão e não consideram falhas de segurança [Aguilera et al. 1997].

Este artigo apresenta, então, o primeiro protocolo para detecção de falhas bizantinas em sistemas dinâmicos e que, além disso, tem a característica de ser assíncrono. O protocolo tem por base as abordagens descritas por [Kihlstrom et al. 2003] para falhas bizantinas e [Sens et al. 2008] para a detecção assíncrona. Assim, as falhas de segurança são detectadas através da definição de um formato pré-estabelecido para as mensagens trocadas pelos processos. Estas devem ser autenticadas e seu conteúdo certificado. Quanto às falhas de omissão, a detecção se baseia no padrão de mensagens imposto pelo algoritmo que utiliza o detector, o que é razoável, uma vez que as falhas bizantinas são definidas como desvio do comportamento especificado pelo algoritmo [Malkhi and Reiter 1997]. A detecção assíncrona é possível graças ao padrão de troca de mensagens, à topologia da rede e a certas propriedades comportamentais seguidas pelos processos no sistema. O protocolo não tolera, entretanto, a mobilidade dos nós.

O restante deste artigo encontra-se estruturado da seguinte forma: na Seção 2, define-se o modelo de sistema. Uma caracterização do modelo de falhas bizantinas é apresentada na Seção 3. As Seções 4-6 apresentam o protocolo desenvolvido, as propriedades comportamentais que o sistema deve atender para que o protocolo funcione corretamente e sua implementação. Um esboço de prova da sua corretude é exibido na Seção 7. Por fim, na Seção 8 apresentam-se as conclusões e propostas de trabalhos futuros.

2. Modelo de sistema

Supõe-se um sistema distribuído composto por um conjunto $\Pi = \{p_1, p_2, \ldots, p_n\}$ com $n > 4$ processos. Nós podem entrar e sair da rede aleatoriamente, a qualquer momento da execução do sistema. Não são feitas quaisquer restrições sobre a velocidade dos processadores, sobre o desvio relativo dos relógios ou sobre os atrasos de transmissão das mensagens; isto é, supõe-se um sistema *assíncrono*. Supõe-se a ocorrência de falhas bizantinas: os processos podem falhar de maneira arbitrária, inclusive maliciosa; supõe-se entretanto a existência de um mecanismo de autenticação de mensagens. Os processos não têm conhecimento de Π ou n, apenas de um subconjunto de Π com os quais estabeleceram comunicação anteriormente. O número máximo de falhas tolerado f é conhecido por todos os processos. Supõe-se, para simplificação das definições apresentadas, a existência de um tempo global t, embora o mesmo não seja conhecido pelos processos. Cada processo é identificado unicamente e processos faltosos não podem obter mais de um identificador, de forma que é impossível a ocorrência de *ataques sybil* [Douceur 2002].

Os processos trocam mensagens por difusão local (*broadcast*) através de uma rede de comunicação sem fio. Supõe-se a existência de canais locais confiáveis, de forma que as mensagens difundidas são recebidas em algum momento no futuro por todos os processos corretos dentro da área de cobertura (*range*) do transmissor [Tang and Gerla 2001, Sun et al. 2002]. Os canais não duplicam, alteram ou inserem novas mensagens e todo processo correto autentica suas mensagens de forma incorruptível.

O sistema pode ser representado por um grafo não-direcionado $G(V, E)$, sendo $V = \Pi$ e $(p_i, p_j) \in E$ se p_i e p_j encontram-se no *range* um do outro (p_i e p_j são denominados *vizinhos*). A área de cobertura $range_i$ de um nó p_i é definida pelo conjunto: $range_i := \{p_j \in \Pi : (p_i, p_j) \in E\}$. Note-se que $|range_i|$ equivale ao grau de p_i em G e que $p_i \in range_j \Leftrightarrow p_j \in range_i$. Ou seja, a comunicação entre os processos é *simétrica*.

Definição 1 (Densidade da área de cobertura (d)) *A densidade d de um grafo de*

comunicação $G(V, E)$ consiste no tamanho do menor range da rede, isto é: $d := min \{|range_i| : i \in \{1, 2, ..., n\}\}$. d é portanto o grau mínimo do grafo.

Considera-se que o parâmetro d da rede é conhecido por todos os processos.

Definição 2 (Rede com f-cobertura bizantina) *Uma rede de comunicação representada pelo grafo $G(V, E)$ tem f-cobertura bizantina se e somente se G é $(2f + 1)$-conexo.*

Um grafo k-conexo possui k caminhos distintos nos vértices entre quaisquer par de vértices, o que leva à seguinte observação:

Observação 1 *Em uma rede com f-cobertura bizantina, apesar da ocorrência de $f < n$ falhas, existirão pelo menos $f + 1$ caminhos distintos entre cada par de nós.*

Verifica-se que uma rede com f-cobertura bizantina apresenta $d \geq 2f + 1$.

3. Falhas bizantinas

A identificação das falhas bizantinas é caracterizada pela satisfação de dois requisitos: (1) os processos corretos devem possuir uma visão coerente das mensagens enviadas por cada processo e (2) os processos corretos devem poder verificar se as mensagens enviadas pelos demais processos estão consistentes com os requisitos do algoritmo sendo executado, o que evidencia que a detecção de falhas bizantinas é definida em função de determinado algoritmo ou protocolo. O primeiro requisito pode ser atendido utilizando duas técnicas distintas: a *redundância da informação* e o uso de *assinaturas digitais não-forjáveis* [Lamport et al. 1982]. O segundo, pela adição de informação às mensagens, na forma de *certificados*, que possam ser utilizados para validar o conteúdo sendo transmitido [Kihlstrom et al. 2003].

A Figura 1 ilustra as classes de falhas bizantinas descritas em [Kihlstrom et al. 2003], distinguindo-se duas superclasses: *detectáveis*, quando comportamento externo do processo faltoso fornece evidências de que o mesmo falhou e *não-detectáveis*, caso contrário. Falhas não-detectáveis podem ser subdivididas em *não-observáveis*, quando os demais processos não podem notar a ocorrência da falha (por exemplo, um processo faltoso informa um parâmetro fornecido pelo usuário incorretamente) e *não-diagnosticáveis*, quando não é possível identificar o processo que gerou a falha (por exemplo, os processos recebem uma mensagem não assinada).

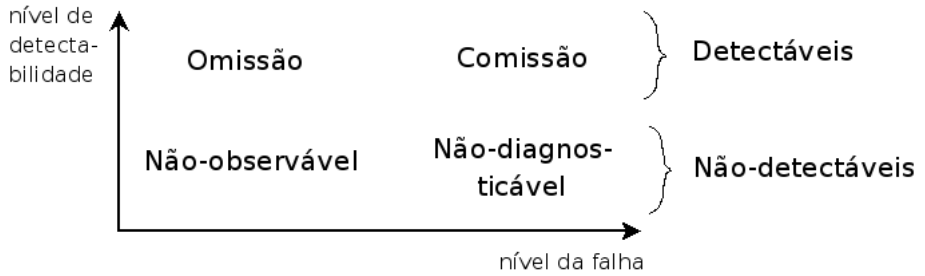

Figura 1. Categorização das falhas bizantinas [Kihlstrom et al. 2003]

As falhas detectáveis são classificadas em falhas *de progresso* (ou falhas por *omissão*) e falhas *de segurança* (ou falhas por *comissão*). Falhas de progresso atrapalham a terminação da computação, uma vez que o processo faltoso não envia mensagens

requeridas por sua especificação ou as envia a apenas parte dos processos do sistema. Falhas de segurança violam propriedades invariantes às quais os processos devem atender, podendo ser definidas como o não-cumprimento de uma das seguintes restrições: (1) um processo deve enviar as mesmas mensagens para todos os outros (um processo faltoso poderia, portanto, enviar a mesma mensagem com valores distintos para processos distintos); (2) as mensagens enviadas devem estar de acordo com o algoritmo sendo executado.

[Kihlstrom et al. 2003] também define classes de detectores de falhas bizantinas, as quais diferem das descritas em [Chandra and Toueg 1996], uma vez que ali são detectadas apenas falhas por colapso. Seja $\mathcal{A}$ um algoritmo que utiliza o detector de falhas como módulo subjacente. A classe $\Diamond\mathcal{S}(\text{Byz}, \mathcal{A})$, que é o foco deste trabalho, é definida pelas seguintes propriedades: (i) *Completude forte bizantina* (para algoritmo $\mathcal{A}$): em algum momento no futuro todo processo correto suspeita permanentemente de todo processo que se desviou de $\mathcal{A}$ de forma detectável; (ii) *Precisão fraca após um tempo*: em algum momento no futuro, um processo correto não será suspeito por qualquer processo correto.

4. Princípios básicos do protocolo de detecção assíncrona de falhas bizantinas

Nesta seção são apresentados os princípios fundamentais para o projeto do detector assíncrono para falhas bizantinas.

Padrão de troca de mensagens. Grande parte dos protocolos de detecção de falhas por colapso baseiam-se num mecanismo de troca de mensagens do tipo "eu estou vivo" (*I'm alive*). Entretanto, no modelo bizantino, devido à ocorrência de processos maliciosos, tal mecanismo não é mais suficiente. Um processo faltoso pode responder corretamente às mensagens do detector de falhas sem no entanto garantir o progresso e a segurança do algoritmo sendo executado. Conseqüentemente, a detecção das falhas deve se basear no padrão das mensagens enviadas na execução do algoritmo $\mathcal{A}$ que utiliza o detector de falhas. Assim sendo, de forma similar a [Kihlstrom et al. 2003], as suspeitas são levantadas em função das mensagens requeridas por $\mathcal{A}$. Entretanto, diferentemente de [Kihlstrom et al. 2003], tais suspeitas são identificadas de maneira assíncrona, sem recorrer ao uso de temporizadores (mecanismos de *timeout*) para detectar falhas de omissão. Além disso, a detecção segue um padrão de troca de mensagens local, ou seja, entre os nós que se encontram na vizinhança.

A detecção assíncrona das falhas é feita aguardando-se a recepção de determinada mensagem a partir de $(d - f)$ remetentes distintos. $(d - f)$ corresponde à quantidade mínima de nós corretos na vizinhança de determinado nó na rede. Esta estratégia é a mesma seguida em [Sens et al. 2008]. Entretanto, o padrão de mensagens QUERY-RESPONSE, no qual [Sens et al. 2008] se fundamenta, não é adequado ao modelo bizantino, pois, como dito anteriormente, um processo faltoso pode enviar mensagens RESPONSE sem no entanto atender aos requisitos de progresso do algoritmo $\mathcal{A}$. O protocolo aqui apresentado irá, portanto, efetuar as suas suspeitas com base na troca de mensagens requeridas por $\mathcal{A}$. Assim, propõe-se que os processos aguardem a recepção das mensagens de $\mathcal{A}$ a partir de pelo menos $(d - f)$ remetentes distintos, suspeitando-se da omissão dos demais processos. É importante observar que para que tal suspeita possa ser feita, o padrão de comunicação do algoritmo $\mathcal{A}$ deve ser distribuído. Ou seja, em cada passo, todos processos devem trocar mensagens entre si, seguindo o padrão $n \rightarrow n$. Protocolos com esse padrão são denominados "simétricos". Como o detector segue um padrão local

de troca de mensagens, essa comunicação deve ocorrer pelo menos entre os processos que estão na mesma vizinhança. Algoritmos de consenso simétricos que utilizam um detector de falhas $\Diamond\mathcal{S}$ foram sugeridos em [Guerraoui and Raynal 2004].

Um outro aspecto importante a considerar é que a detecção segue um padrão assíncrono. Como as suspeitas são feitas com base no padrão de troca de mensagens do algoritmo, conjectura-se ser impossível realizar a detecção de falhas por omissão se tal padrão é do tipo $1 \rightarrow n$. Ou seja, se, em algum momento da execução do algoritmo, requer-se que apenas um processo envie mensagens). Isto porque não haveria como diferenciar uma falha por omissão de um retardo na recepção da mensagem proveniente de tal processo, dado que o sistema subjacente é assíncrono [Chandra and Toueg 1996]. Portanto, identificamos a seguinte conjectura, que se correta deriva o seguinte corolário:

Conjectura 1 *Num sistema assíncrono, é impossível detectar falhas bizantinas por omissão, de forma assíncrona, caso o padrão de troca de mensagens do algoritmo $\mathcal{A}$ seja do tipo $1 \rightarrow n$; isto é, se em determinado momento, o algoritmo $\mathcal{A}$ determina que apenas um processo envia mensagens aos demais (n).*

Corolário 1 *A forma assíncrona da detecção de falhas bizantinas só pode ser adotada por protocolos simétricos, nos quais todos os nós executam o mesmo papel.*

Geração de suspeitas. Cada suspeita (*suspicion*) sobre um processo p_i é associada a uma mensagem m requerida por $\mathcal{A}$. Requer-se, portanto, que as mensagens recebam identificadores únicos. As suspeitas são propagadas na rede e um processo correto adotará uma suspeita não gerada por ele próprio se e somente se receber pelo menos $f + 1$ ocorrências devidamente autenticadas provenientes de processos distintos. O requisito de pelo menos $f + 1$ ocorrências impede que processos maliciosos imponham suspeitas sobre corretos.

Geração de equívocos. Seja p_i um processo suspeito de não ter enviado uma mensagem m. Se em algum momento um processo correto p_j receber m de p_i, p_j declarará um equívoco ($mistake$) sobre a suspeita e difundirá a mensagem m aos demais, a fim de que façam o mesmo. Esse procedimento permite que um processo malicioso, de forma intermitente, provoque uma suspeita e em seguida a revogue, levando-se a um mascaramento de parte das falhas por omissão e a uma degradação do desempenho do detector de falhas. Não é possível, no entanto, distinguir um processo com este comportamento de um processo lento ou de um período de instabilidade no canal de comunicação. Esta forma de geração de equívocos difere da apresentada em [Sens et al. 2008], na qual o próprio processo suspeito gera um equívoco.

Detecção de falhas de segurança. Para ser possível a detecção das falhas de segurança, um formato de mensagens deve ser estabelecido. Cada mensagem deve também incluir um certificado que possibilite aos demais processos verificar a coerência da mesma com o algoritmo $\mathcal{A}$. Caso um processo correto detecte a invalidade de uma mensagem recebida, seja por não atender ao formato ou por não estar devidamente justificada, suspeitará permanentemente do processo remetente e encaminhará a mensagem original aos demais processos, de forma que a suspeita seja propagada.

No protocolo, suspeitas, equívocos e provas de falhas de segurança são todos encaminhados através de uma única mensagem do tipo SUSPICION, as quais não precisam ser certificadas. Mensagens que não estejam devidamente autenticadas são descartadas, uma

vez que configuram uma falha não-diagnosticável (ver Seção 3). O detector, portanto, não comete erros na detecção das falhas de segurança do algoritmo $\mathcal{A}$.

Em [Kihlstrom et al. 2003], adicionalmente, é necessário detectar quando um processo encaminha duas versões diferentes da mesma mensagem, ou seja, *mensagens mutantes*. Para isto requer-se, de um lado, que os processos corretos reencaminhem para todos os demais cada mensagem recebida, e, por outro lado, seja gerado um histórico das mensagens provenientes de cada processo. Supõe-se, no entanto, que a comunicação entre os processos é ponto a ponto. No modelo aqui proposto, em que os processos se comunicam apenas por difusão local em canais confiáveis, é natural supor que uma mensagem transmitida será recebida, com conteúdo idêntico, por todos os processos corretos, de forma que é impossível a ocorrência de mensagens mutantes. Chega-se, portanto, à seguinte observação:

Observação 2 *Em um ambiente de falhas bizantinas, a suposição de comunicação por difusão (*broadcast*) em canais confiáveis simplifica o tratamento das falhas de segurança, uma vez que os processos vizinhos do remetente têm uma visão consistente das mensagens enviadas pelo mesmo.*

5. Propriedades comportamentais do sistema

Para que o protocolo proposto neste trabalho detecte as falhas de maneira assíncrona e satisfaça as propriedades definidas para a classe $\Diamond\mathcal{S}(\text{Byz}, \mathcal{A})$, é necessário que o sistema atenda a certas propriedades comportamentais.

Propriedade 1 (Propriedade de Inclusão Bizantina ($Byz\mathcal{MP}$)) *Seja t um instante de tempo e KB_i^t o conjunto de processos que receberam uma mensagem* SUSPICION *de p_i até o instante t. Um processo p_i satisfaz a propriedade de inclusão bizantina se:*

$$Byz\mathcal{MP}(p_i) := \exists t \geq 0 : |KB_i^t| > 2f + 1$$

Esta propriedade garante que um processo novo p_i no sistema em algum momento no futuro será conhecido por pelo menos $2f + 1$ processos, dos quais pelo menos $f + 1$ serão corretos. Para tanto, é necessário que p_i se comunique com pelo menos $2f + 1$ processos dentro de sua área de cobertura, enviando-lhes uma mensagem SUSPICION por difusão. Dessa forma, se p_i falhar, em algum momento no futuro pelo menos $f+1$ processos corretos suspeitarão de p_i e transmitirão a suspeita para o restante do sistema, de forma que a propriedade de completude forte bizantina do detector $\Diamond\mathcal{S}(\text{Byz}, \mathcal{A})$ seja atendida. Note-se que essas comunicações devem ser realizadas antes do início do próximo passo do algoritmo $\mathcal{A}$, de forma a garantir a propriedade de completude forte bizantina do detector de falhas. Se um processo novo não se comunicar com nenhum outro, é impossível atender à propriedade de completude fraca [Fernández et al. 2006].

Para que a propriedade de precisão fraca após um tempo da classe $\Diamond\mathcal{S}(\text{Byz}, \mathcal{A})$ seja atendida, é necessário que exista um processo correto cujas mensagens a partir de algum momento estejam sempre entre as $d - f$ primeiras recebidas pelos seus vizinhos, a cada requisição de $\mathcal{A}$. Assim sendo, após um tempo p_i não será mais suspeito por nenhum processo correto e terá quaisquer suspeitas anteriores revogadas através de mensagens de equívocos. Logo, a rede deve possuir um nó correto que atenda à propriedade de *receptividade bizantina*, definida como:

Propriedade 2 (Propriedade de Receptividade Bizantina ($Byz\mathcal{RP}$)) *Sejam t e u instantes de tempo e seja $byz\_rec\_from_j^t$ o conjunto de pelo menos $d-f$ processos dos quais p_j recebeu a mensagem requerida por $\mathcal{A}$ no último passo em execução até o instante t. A propriedade $Byz\mathcal{RP}$ do processo correto p_i é definida como:*

$$Byz\mathcal{RP}(p_i) := \exists u : \forall t > u, \forall p_j \in range_i, p_i \in byz\_rec\_from_j^t$$

6. Protocolo de detecção assíncrona de falhas bizantinas em sistemas dinâmicos

Os Algoritmos 1 e 2 implementam um detector de falhas da classe $\Diamond\mathcal{S}(\text{Byz}, \mathcal{A})$, desde que a rede atenda às propriedades de f-cobertura bizantina, inclusão bizantina e receptividade bizantina descritas anteriormente e que o protocolo executado pelo algoritmo $\mathcal{A}$ seja simétrico. Cada processo executa três tarefas em paralelo, descritas a seguir. Posteriormente são descritas as variáveis, primitivas e procedimentos para cada processo p_i.

T1. Geração de novas suspeitas (linhas 5-14, Algoritmo 1). Quando o algoritmo $\mathcal{A}$ requer que os processos troquem entre si uma mensagem m (linha 6), cada processo p_i aguarda receber m de pelo menos $d - f$ processos, cujos identificadores são armazenados no conjunto rec_from_i (linhas 7-8). Para os demais processos conhecidos por p_i, é adicionada uma suspeita interna de falha por omissão (linhas 9-11). Cada mensagem é então verificada quanto à sua formação e certificação (linhas 12-14, Algoritmo 1 e 1-12, Algoritmo 2). Mensagens incorretas levam a suspeitas de falhas de segurança (linhas 5-6, Algoritmo 2) e à atualização da saída do detector; mensagens corretas levam à geração de equívocos de possíveis suspeitas de falhas por omissão (linhas 8-9, Algoritmo 2).

T2. Recepção de mensagens de processos lentos e de mensagens SUSPICION (linhas 16-18, Algoritmo 1). Uma vez que as mensagens enviadas por um processo lento podem ser recebidas após a geração das suspeitas pelos seus vizinhos, é necessário que estes identifiquem tais eventos separadamente, o que é feito por esta tarefa. As mensagens são tratadas de forma similar à tarefa T1. O tratamento das mensagens SUSPICION será explanado adiante.

T3. Divulgação do novo estado de suspeitas e equívocos (linhas 20-23, Algoritmo 1). Tarefa executada periodicamente para enviar aos vizinhos de p_i (linha 22) a visão de p_i quanto às suspeitas (internas e externas), equívocos e provas de falhas de segurança. Os vizinhos de p_i receberão esta mensagem na tarefa T2 e tratá-la-ão da seguinte forma:

Atualização do estado interno (linhas 11 e 14-36, Algoritmo 2). Ao receber uma mensagem SUSPICION de um vizinho q (linha 15), um processo p_i atualiza seu estado interno com novas informações. Suspeitas internas e externas de q são adicionadas ao conjunto de suspeitas externas de p_i (linhas 16-27), possivelmente gerando novas suspeitas internas (linhas 31-33, Algoritmo 1). Note-se que será gerada uma suspeita de falha de segurança do processo q (linhas 5-7) caso a mensagem SUSPICION m esteja mal-formada ou não justificada. Provas de falhas de segurança e informações de equívoco são tratadas de forma similar às mensagens recebidas diretamente do remetente (linhas 30 e 34).

VARIÁVEIS:
• $output_i$: armazena a saída do detector de falhas, isto é, o conjunto de identificadores dos processos de que p_i suspeita terem falhado;
• $known_i$: armazena o conjunto dos processos que se comunicaram com p_i, isto é, sua

Algoritmo 1 Detector Assíncrono de Falhas Bizantinas em Sistemas Dinâmicos

1: **init:**
2: $output_i \leftarrow known_i \leftarrow \emptyset$; $extern\_susp_i \leftarrow [][]$
3: $intern\_susp_i \leftarrow mistake_i \leftarrow []$; $byzantine_i \leftarrow \emptyset$
4:
5: **Task** T1: */* geração de novas suspeitas */*
6: **when** p_i requer uma mensagem m **do**
7: **wait until** receber m devidamente autenticada pela primeira vez de pelo menos $(d - f)$ processos distintos
8: $rec\_from_i \leftarrow \{p_j \mid p_i$ recebeu uma mensagem de p_j na linha 7$\}$
9: **for all** $p_j \in (known_i \setminus rec\_from_i)$ **do**
10: AddInternalSusp(p_j, m)
11: **end for**
12: **for all** m_j recebida na linha 7 **from** p_j **do**
13: ValidateReceived(p_j, m_j)
14: **end for**
15:
16: **Task** T2: */* recepção de mensagens de processos lentos e do estado interno de outro processo */*
17: **upon receipt of** m devidamente autenticada **from** p_j **do**
18: ValidateReceived(p_j, m)
19:
20: **Task** T3: */* difusão do estado das suspeitas */*
21: **loop**
22: broadcast $\langle$SUSPICION, $intern\_susp_i$, $extern\_susp_i$, $mistake_i$, $byzantine_i\rangle$
23: **end loop**
24:
25: /* PROCEDIMENTOS AUXILIARES */
26: **procedure** AddInternalSusp(q, m):
27: $intern\_susp_i[q] \leftarrow intern\_susp_i[q] \cup \{m.\text{id}\}$; $output_i \leftarrow output_i \cup \{q\}$
28:
29: **procedure** AddExternalSusp(q, idm, p_s):
30: $extern\_susp_i[q][idm] \leftarrow extern\_susp_i[q][idm] \cup \{p_s\}$
31: **if** $|extern\_susp_i[q][idm]| \geq f + 1$ **then**
32: AddInternalSusp(q, message(idm))
33: **end if**
34:
35: **procedure** AddMistake(q, m):
36: $mistake_i[q] \leftarrow mistake_i[q] \cup \{m\}$; $extern\_susp_i[q][m.\text{id}] \leftarrow \emptyset$
37: $intern\_susp_i[q] \leftarrow intern\_susp_i[q] \setminus \{m.\text{id}\}$
38: **if** $intern\_susp_i[q] = []$ **and** $\nexists\langle q, -\rangle \in byzantine_i$ **then**
39: $output_i \leftarrow output_i \setminus \{q\}$
40: **end if**
41:
42: **procedure** AddByzantine(q, m):
43: $output_i \leftarrow output_i \cup \{q\}$; $byzantine_i \leftarrow byzantine_i \cup \{\langle q, m\rangle\}$

Algoritmo 2 Detector Assíncrono de Falhas Bizantinas em Sistemas Dinâmicos (continuação)

```
1: procedure ValidateReceived(q, m):
2: if m foi enviada diretamente por q then
3:    known_i ← known_i ∪ {q}
4: end if
5: if m não está devidamente formada or m não está devidamente justificada then
6:    AddByzantine(q, m)
7: else
8:    if m.id ∈ intern_susp_i[q] then
9:       AddMistake(q, m)
10:   end if
11:   UpdateSuspicions(q, m)
12: end if
13:
14: procedure UpdateSuspicions(q, m):
15: if m = ⟨SUSPICION, intern_susp_q, extern_susp_q, mistake_q, byzantine_q⟩ then
16:   for all p_x ∈ keys(extern_susp_q) | ∄⟨p_x, −⟩ ∈ byzantine_i do
17:      for all idm_x ∈ keys(extern_susp_q[p_x]) devidamente autenticado | idm_x ∉
         intern_susp_i[p_x] ∪ ids(mistake_i[p_x]) do
18:         for all p_y ∈ extern_susp_q[p_x][idm_x] do
19:            AddExternalSusp(p_x, idm_x, p_y)
20:         end for
21:      end for
22:   end for
23:   for all p_x ∈ keys(intern_susp_q) | ∄⟨p_x, −⟩ ∈ byzantine_i do
24:      for all idm_x ∈ intern_susp_q[p_x] devidamente autenticado | idm_x ∉
         intern_susp_i[p_x] ∪ ids(mistake_i[p_x]) do
25:         AddExternalSusp(p_x, idm_x, q)
26:      end for
27:   end for
28:   for all p_x ∈ keys(mistake_q) | ⟨p_x, −⟩ ∉ byzantine_i do
29:      for all m_x ∈ mistake_q[p_x] | m_x ∉ mistake_i[p_x] do
30:         ValidateReceived(p_x, m_x)
31:      end for
32:   end for
33:   for all ⟨p_x, m_x⟩ ∈ byzantine_q | ∄⟨p_x, −⟩ ∈ byzantine_i do
34:      ValidateReceived(p_x, m_x)
35:   end for
36: end if
```

suposta vizinhança. É atualizada a cada recepção de mensagem m, seja m do tipo SUSPICION ou uma mensagem inerente a $\mathcal{A}$;
• $extern\_susp_i$: matriz que armazena suspeitas externas (geradas por outros processos). A matriz é indexada por um identificador de processo q e por um identificador de mensagem idm. Cada entrada armazena o conjunto de processos dos quais p_i recebeu suspeitas de q não ter enviado a mensagem com identificador idm;
• $intern\_susp_i$: vetor de suspeitas internas. Uma suspeita interna é gerada pela não-recepção de uma mensagem requerida por $\mathcal{A}$ ou pela existência de pelo menos $f + 1$ suspeitas externas sobre um mesmo par processo-mensagem;
• $mistake_i$: vetor que armazena, para cada processo aplicável p_j, o conjunto de equívocos relacionados a p_j, na forma de mensagens requeridas por $\mathcal{A}$ sobre as quais foram levantadas suspeitas;
• $byzantine_i$: conjunto de tuplas do tipo ⟨processo, mensagem⟩ que provam comportamento bizantino do processo associado. A notação $\langle p, - \rangle$ indica "qualquer tupla associada ao processo p";
• rec_from_i: conjunto de processos dos quais p_i recebeu a mensagem requisitada por $\mathcal{A}$.

PRIMITIVAS:
• m.id - retorna o identificador da mensagem m;
• message(idm) - retorna a mensagem associada ao identificador idm;
• verificação de boa formação, justificação (certificação do conteúdo) e autenticação das mensagens;
• requisição por $\mathcal{A}$ de uma determinada mensagem;
• broadcast m - difunde a mensagem m para os vizinhos de p_i;
• keys(v) - retorna o conjunto de índices de um vetor dinâmico v;
• ids(s) - retorna o conjunto de identificadores associados às mensagens do conjunto s.

PROCEDIMENTOS AUXILIARES:
• AddInternalSusp(q, m) (linhas 26-27, Algoritmo 1): adiciona uma suspeita interna sobre o processo q em relação à mensagem m;
• AddExternalSusp(q, idm, p_s) (linhas 29-33, Algoritmo 1): adiciona uma suspeita externa sobre o processo q proveniente do processo p_s em relação à mensagem com identificado idm. Adicionalmente, caso existam $f + 1$ ou mais suspeitas externas sobre q em relação a message(idm), gera uma suspeita interna equivalente, caso ainda não exista;
• AddMistake(q, m) (linhas 35-40, Algoritmo 1): adiciona um equívoco sobre a suspeita de q não ter enviado m, retirando todas as suspeitas internas e externas associadas. Caso q não tenha outras suspeitas nem tenha apresentado comportamento bizantino, remove-o da saída do detector de falhas;
• AddByzantine(q, m) (linhas 42-43, Algoritmo 1): adiciona q permanentemente à lista de processos maliciosos (e, conseqüentemente, à saída do FD), juntamente com a mensagem m associada à falha bizantina, servindo como prova da falha;
• ValidateReceived(q, m) (Algoritmo 2, linhas 1-12): verifica a validade (boa formação e justificação) da mensagem m recebida de q, retirando quaisquer suspeitas associadas ao par (q, m), caso m seja válida, e gerando uma suspeita de falha de segurança, caso contrário. Adicionalmente, atualiza o conjunto de nós conhecidos por p_i ($known_i$) e repassa as mensagens ao procedimento a seguir, de forma a atualizar o estado de suspeitas;
• UpdateSuspicions(q, m) (Algoritmo 2, linhas 14-36): caso m seja do tipo SUSPICION, atualiza o estado interno de p_i com as informações de falhas contidas em m.

7. Esboço da prova de corretude

A corretude do algoritmo é definida em função das propriedades de completude forte bizantina e precisão fraca após um tempo da classe $\Diamond\mathcal{S}(\text{Byz}, \mathcal{A})$ de detectores de falhas. Dessa forma, os argumentos são exibidos para cada propriedade separadamente:

Completude forte bizantina. Se um processo p_i falhar por omissão, não enviando uma mensagem m requerida pelo algoritmo $\mathcal{A}$, os demais processos em $range_i$ terão recebido mensagens de p_i, conterão p_i em seus respectivos conjuntos $known$ e suspeitarão de p_i com base no mecanismo de espera de $d - f$ mensagens descrito na Seção 4 (linhas 7-11 do Algoritmo 1). Devido às propriedades de f-cobertura bizantina e $Byz\mathcal{MP}$, pelo menos $f + 1$ processos corretos suspeitarão corretamente de p_i, armazenando a suspeita em seus conjuntos $intern\_susp$ (linhas 26-27, Algoritmo 1), de forma que as suspeitas serão divulgadas aos seus vizinhos em uma mensagem SUSPICION na próxima execução da tarefa T3 (linhas 20-23, Algoritmo 1).

A propagação das suspeitas na rede é garantida pela tarefa T2 e pelos procedimentos ValidateReceived e UpdateSuspicions. Seja p_j um vizinho correto de um nó falho p_i. Ao receber uma mensagem SUSPICION de p_j, um processo correto p_k armazena as suspeitas de p_j como suspeitas externas (linhas 23-27, Algoritmo 2), no caso de falhas por omissão. Como as suspeitas externas são reencaminhadas pelos nós que as recebem (linhas 22, Algoritmo 1 e 16-22, Algoritmo 2), devido à propriedade f-cobertura bizantina, por indução, em algum momento futuro p_k receberá mais f suspeitas externas associadas (p_i, m), adotando assim tal suspeita (linhas 31-33, Algoritmo 1). Por indução, verifica-se que todos os nós corretos da rede acabarão por suspeitar de p_i não ter enviado m.

Um processo p_x que entrar na rede após o início da execução da computação, devido à propriedade $Byz\mathcal{MP}$, será conhecido por pelo menos $2f + 1$ processos (linhas 2-3 e 22, Algoritmo 1, linhas 2-4, Algoritmo 2). Desses, pelo menos $f + 1$ serão corretos, terão suspeitado de p_i não ter enviado m e encaminharão a suspeita a p_x nas próximas iterações do laço da tarefa T3. Por receber pelo menos $f + 1$ suspeitas distintas (ao menos dos seus vizinhos corretos), p_x pode suspeitar corretamente de p_i não ter enviado m.

Da mesma forma, se p_i cometer uma falha de segurança, tendo em vista o uso de um formato padrão de mensagens e de certificação do conteúdo, a mesma será identificada por seus vizinhos nas chamadas a ValidateReceived, linhas 13 e 18, Algoritmo 1. Por argumento semelhante ao exibido para falhas por omissão, a mensagem associada m é armazenada no conjunto $byzantine$ dos nós em $range_i$ e transmitida a seus vizinhos em uma mensagem SUSPICION (linhas 5-6 do Algoritmo 2, 42-43 e 22, Algoritmo 1). Ao receber a mensagem SUSPICION de um processo p_j vizinho a p_i, um processo p_k vizinho a p_j analisará a mensagem m (linhas 33-35, Algoritmo 2) e, uma vez que as mensagens são autenticadas, poderá constatar a falha do nó p_i. Por indução, em algum momento futuro todo processo correto na rede recebe a evidência da falha. De forma semelhante ao argumentado para as falhas por omissão, nós corretos que entrarem na rede após o início da computação receberão a informação da falha de p_i e também suspeitarão do mesmo.

Precisão fraca após um tempo. Seja p_i um processo que nunca falhe e que atenda à propriedade $Byz\mathcal{RP}$. Como as mensagens são autenticadas e p_i é correto, nenhum processo malicioso terá sucesso em divulgar que p_i cometeu uma falha de segurança. Pela propriedade $Byz\mathcal{RP}$, existirá um instante t após o qual as mensagens de p_i requeridas

por $\mathcal{A}$ serão sempre recebidas pelos processos em $range_i$ na linha 7, Algoritmo 1. Desta forma, os processos corretos em $range_i$ nunca mais suspeitarão de p_i ter se omitido. Haverá ainda um instante $u > t$ após o qual toda suspeita sobre p_i anterior a t será revogada pelos vizinhos de p_i (linhas 12-14, 16-18 e 35-40, Algoritmo 1 e 8-10 do Algoritmo 2). O equívoco será divulgado por toda a rede, por argumento semelhante ao feito quanto às suspeitas e pela atualização feita nas linhas 28-32, Algoritmo 2. Logo, nenhum processo correto suspeitará de p_i após u. Se um ou mais processos maliciosos levantarem uma suspeita de omissão sobre p_i após t, como existem no máximo f processos faltosos no sistema, não seria possível convencer os processos corretos de adotarem tal suspeita, visto que seriam necessárias suspeitas provenientes de $f + 1$ processos distintos.

8. Conclusão e trabalhos futuros

Este trabalho apresentou um detector de falhas bizantinas com duas características inovadoras: (i) ele foi projetado para sistemas distribuídos dinâmicos, cujo conjunto de participantes na rede é desconhecido e além disso, (ii) tem a característica de ser assíncrono, isto é, não se utiliza de temporizadores para a detecção das falhas de progresso. Para que a detecção seja realizada de forma assíncrona, conjectura-se ser necessário que o algoritmo que utiliza o detector de falhas seja simétrico, isto é, que a cada passo todos os processos troquem mensagens entre si. Um aspecto interessante é que a comunicação por difusão (*broadcast*), típica das redes sem fio, simplifica o tratamento de falhas bizantinas, uma vez que os processos vizinhos de um remetente têm uma visão uniforme das mensagens por ele enviadas. Especificamente, não é necessário tratar a ocorrência de *mensagens mutantes*, isto é, quando um mesmo processo envia a mesma mensagem com conteúdos diferentes para diferentes processos. Objetivam-se como trabalhos futuros (i) estender o protocolo para prover tolerância à mobilidade dos nós; (ii) implementar o protocolo, com o objetivo de avaliar seu desempenho e viabilidade prática; (iii) provar ou contra-exemplificar a impossibilidade de realização de detecção de falhas de forma assíncrona em sistemas dinâmicos sujeitos a falhas bizantinas com comunicação do tipo $1 \rightarrow n$.

Referências

Aguilera, M. K., Chen, W., and Toueg, S. (1997). Heartbeat: A Timeout-Free Failure Detector for Quiescent Reliable Communication. In *Proc. of the 11th Int. Workshop on Distributed Algorithms*, pages 126–140, London, UK. Springer-Verlag.

Awerbuch, B., Holmer, D., Nita-Rotaru, C., and Rubens, H. (2002). An on-demand secure routing protocol resilient to byzantine failures. In *Proc. of the 1st ACM Workshop on Wireless Security*, pages 21–30, New York, NY, USA. ACM.

Baldoni, R., Hélary, J.-M., and Piergiovanni, S. T. (2007). A Component-Based Methodology to Design Arbitrary Failure Detectors for Distributed Protocols. In *Proc. of the 10th IEEE Int. Symp. on Object and Component-Oriented Real-Time Distributed Computing*, pages 51–61, Washington, DC, USA. IEEE Computer Society.

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

Correia, M., Veríssimo, P., and Neves, N. F. (2006). Byzantine Consensus in Asynchronous Message-Passing Systems: a Survey. In *Resilience-building Technologies: State of Knowledge*, Deliverable D12, Part Algo, chapter 1. RESIST Network of Excellence.

Douceur, J. R. (2002). The sybil attack. In *Revised Papers from the First Int. Workshop on Peer-to-Peer Systems*, pages 251–260, London, UK. Springer-Verlag.

Fernández, A., Jiménez, E., and Arévalo, S. (2006). Minimal system conditions to implement unreliable failure detectors. In *12th Pacific Rim Int. Symp. on Dependable Computing*, pages 63–72, Los Alamitos, CA, USA. IEEE Computer Society.

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

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

Guerraoui, R. and Raynal, M. (2004). The Information Structure of Indulgent Consensus. *IEEE Trans. Comput.*, 53(4):453–466.

Gupta, I., Chandra, T. D., and Goldszmidt, G. S. (2001). On scalable and efficient distributed failure detectors. In *Proc. of the 20th Annual ACM Symp. on Principles of Distributed Computing*, pages 170–179, New York, NY, USA. ACM Press.

Kihlstrom, K. P., Moser, L. E., and Melliar-Smith, P. M. (2003). Byzantine Fault Detectors for Solving Consensus. *The Computer Journal*, 46(1):16–35.

Lamport, L., Shostak, R., and Pease, M. (1982). The byzantine generals problem. *ACM Trans. Program. Lang. Syst.*, 4(3):382–401.

Larrea, M., Fernández, A., and Arévalo, S. (2000). Optimal implementation of the weakest failure detector for solving consensus. In *Proc. of the 19th IEEE Symp. on Reliable Distributed Systems*, pages 52–59, Washington, DC, USA. IEEE Computer Society.

Malkhi, D. and Reiter, M. (1997). Unreliable intrusion detection in distributed computations. In *Proc. 10th Computer Security Foundations Workshop*, pages 116–124, Rockport, MA. IEEE Computer Society Press, Los Alamitos, CA.

Mostefaoui, A., Mourgaya, E., and Raynal, M. (2003). Asynchronous Implementation of Failure Detectors. In *Proc. of the 2003 Int. Conf. on Dependable Systems and Networks*, page 351, Los Alamitos, CA, USA. IEEE Computer Society.

Mostefaoui, A., Raynal, M., Travers, C., Patterson, S., Agrawal, D., and El Abbadi, A. (2005). From Static Distributed Systems to Dynamic Systems. In *Proc. of the 24th IEEE Symp. on Reliable Distributed Systems*, pages 109–118, Los Alamitos, CA, USA. IEEE Computer Society.

Sens, P., Greve, F., Arantes, L., Bouillaguet, M., and Simon, V. (2008). Um Detector de Falhas Assíncrono para Redes Móveis e Auto-Organizáveis. In *Simpósio Brasileiro de Redes de Computadores e Sistemas Distribuídos*, Rio de Janeiro, RJ, Brazil.

Sun, M.-T., Huang, L., Arora, A., and Lai, T.-H. (2002). Reliable MAC Layer Multicast in IEEE 802.11 Wireless Networks. In *ICPP '02: Proceedings of the 2002 International Conference on Parallel Processing*, pages 527–536, Washington, DC, USA. IEEE Computer Society.

Tang, K. and Gerla, M. (2001). MAC reliable broadcast in ad hoc networks. In *IEEE Military Communications Conference, 2001. MILCOM 2001. Communications for Network-Centric Operations: Creating the Information Force*, volume 2, pages 1008–1013.