Consenso Bizantino entre Participantes Desconhecidos*

Eduardo A. P. Alchieri[1], Alysson N. Bessani[2], Joni S. Fraga[1], Fabíola Greve[3]

[1]DAS, Universidade Federal de Santa Catarina, Florianópolis

[2]LaSIGE, Faculdade de Ciências da Universidade de Lisboa, Lisboa

[3]DCC, Universidade Federal da Bahia, Bahia

Resumo. *O problema do consenso é a base para a solução da maioria dos problemas que envolvem sistemas distribuídos confiáveis. Apesar do consenso estar amplamente estudado em ambientes clássicos, onde o conjunto de participantes é conhecido, poucos trabalhos consideram este problema em ambientes dinâmicos e auto-organizáveis, onde os participantes da computação são, a priori, desconhecidos. Neste trabalho, propomos duas soluções para o consenso bizantino entre participantes desconhecidos (BFT-CUP), com e sem o uso de assinaturas digitais, e provamos que o grau de conhecimento sobre os participantes do sistema necessário nestas soluções é o mínimo necessário para resolver o BFT-CUP.*

1. Introdução

O problema do consenso [Lamport et al. 1982, Chandra and Toueg 1996], e de um modo mais geral os algoritmos de acordo, formam a base para a solução da maioria dos problemas encontrados no desenvolvimento de sistemas distribuídos confiáveis, pois possibilitam que os participantes da computação distribuída coordenem suas ações de forma a manter a consistência em seus estados e garantir o progresso do sistema. Primeiramente, este problema foi estudado em redes clássicas, onde o conjunto de processos que participam de determinada computação é estático e conhecido por todos os participantes do sistema. Mesmo nestes ambientes, várias dificuldades são encontradas na solução deste problema, que não pode ser resolvido de forma determinista em um sistema assíncrono com possibilidade de falhas [Fischer et al. 1985].

Quando consideramos um sistema dinâmico e auto-organizável, como redes *ad-hoc*, redes de sensores e, num contexto diferente, redes entre pares (P2P) não estruturadas, as dificuldades encontradas na elaboração de protocolos para resolver o problema do consenso aumentam significativamente, pois nestes ambientes os participantes da computação são, *a priori*, desconhecidos (tanto a número de participantes quanto suas identidades). Desta forma, não é possível o simples emprego de protocolos de consenso estático neste novo ambiente, pois tais protocolos consideram que os participantes do sistema são, *a priori*, conhecidos.

Sendo assim, os primeiros esforços para resolver o consenso em redes desconhecidas (ou sistemas dinâmicos e auto-organizáveis) surgiram em iniciativas como o CUP (*Consensus with Unknown Participants*) [Cavin et al. 2004], que têm por objetivo resolver o consenso em redes desconhecidas, considerando que os processos não podem falhar. Outras iniciativas, como o FT-CUP (*Fault-Tolerant* CUP) [Cavin et al. 2005, Greve and Tixeuil 2007], buscam estender o modelo anterior fornecendo suporte a falhas por parada (*crash*) dos participantes do sistema.

Neste trabalho, definimos uma nova versão do problema de consenso em redes desconhecidas, chamada de BFT-CUP (*Byzantine Fault-Tolerant* CUP), onde o problema do consenso em redes desconhecidas é resolvido considerando que os participantes do sistema podem se comportar de forma maliciosa [Lamport et al. 1982]. Deste modo, além de tratar as questões inerentes aos sistemas auto-organizáveis, os protocolos devem tolerar a possibilidade de participantes maliciosos estarem presentes no sistema (o que é bastante factível, dada sua natureza aberta), tentando impedir que o consenso seja atingido. Duas soluções para o BFT-CUP são

*Alchieri é bolsista CAPES; Fraga e Greve são bolsistas CNPq.

propostas neste artigo, com e sem o uso de assinaturas digitais, requerendo diferentes graus de conectividade do sistema. Além disso, é provado que estas soluções são ótimas em termos da conectividade requerida entre os participantes do sistema neste modelo faltas, quando assumimos o mínimo de sincronia necessária para resolver consenso.

2. Modelo de Sistema

Assume-se o modelo de sistema com sincronia parcial: existem limites para o tempo necessário para a transmissão de uma mensagem e para a realização de qualquer computação que terminam por valer no sistema, no entanto, esses limites não são conhecidos [Dwork et al. 1988]. A idéia por trás deste modelo é de que o sistema trabalha de forma assíncrona (não respeitando nenhum limite de tempo) a maior parte do tempo. Porém, durante períodos de estabilidade, o tempo para transmissão de mensagens é limitado. Este modelo é necessário para a execução de um protocolo de consenso bizantino clássico ou estático (seção 3.4.4).

Em relação aos processos, considera-se um sistema distribuído formado por um conjunto finito Π de $n > 1$ processos[1], sendo que cada processo $p_i \in \Pi$ conhece apenas um subconjunto Π_i de Π. As comunicações entre estes processos se dão através do algoritmo definido na seção 3.3. Um processo p_i apenas pode enviar mensagens para outro processo p_j se $p_j \in \Pi_i$. Além disso, se p_i envia uma mensagem para p_j, tal que $p_i \notin \Pi_j$, então, após p_j receber a mensagem o mesmo pode adicionar p_i em Π_j, i.e., p_j passa a poder enviar mensagens para p_i. Implicitamente, considera-se que existe uma camada de roteamento tolerante a faltas bizantinas [Castro et al. 2002, Awerbuch et al. 2002] responsável por encaminhar as mensagens entre os processos que se conhecem. Processos do sistema estão sujeitos a falhas bizantinas [Lamport et al. 1982]: um processo que apresenta este tipo de falha pode exibir qualquer comportamento, podendo parar, omitir envios ou entregas de mensagens, ou desviar arbitrariamente de sua especificação. Um processo que apresenta comportamento de falha é dito falho, de outra forma é dito correto. O número máximo de processos que podem falhar f é conhecido por todos os processos do sistema.

3. BFT-CUP: Consenso Bizantino entre Participantes Desconhecidos

Em um sistema distribuído, formado por vários processos independentes, o problema do consenso consiste em fazer com que todos os processos corretos acabem por decidir o mesmo valor, o qual deve ter sido previamente proposto por algum dos processos do sistema. Formalmente, o consenso pode ser definido pelas seguintes propriedades [Castro and Liskov 2002, Chandra and Toueg 1996]: *Validade*: um processo correto decide pelo valor v somente se v foi previamente proposto por algum processo; *Acordo*: se um processo correto decide pelo valor v, então todos os processos corretos terminam por decidir v; *Terminação*: todos os processos corretos terminam por decidir; *Integridade*: cada processo correto decide no máximo uma vez.

Nesta seção são apresentadas duas soluções para o BFT-CUP. A primeira utiliza assinaturas digitais e necessita de uma menor conectividade no grafo que representa o conhecimento dos participantes. A segunda, que não faz uso de assinaturas digitais, requer um maior grau de conhecimento sobre os participantes que estarão presentes em uma dada execução do consenso. As principais diferenças entre as duas abordagens estão relacionadas com a comprovação da autenticidade das mensagens enviadas pelos participantes (seção 3.3) e com a fase de descoberta de participantes (seção 3.4.2). Além disso, quando assinaturas digitais são empregadas, torna-se necessário a existência de uma autoridade certificadora[2], reconhecida por todos os processos,

[1]Neste trabalho os termos nós, vértices, processos e participantes serão usados indistintamente.

[2]Não é necessário que esta autoridade certificadora esteja *online* durante a execução dos protocolos, mas sim na criação dos certificados (e suas possíveis revogações). Somente é necessário que os participantes conheçam a chave pública desta autoridade, a fim de validar os certificados emitidos pela mesma.

responsável pelo gerenciamento das chaves dos participantes do BFT-CUP. Assim, cada participante do sistema possui um par de chaves pública-privada, sendo a chave privada conhecida apenas pelo próprio participante. Já as chaves públicas de todos os participantes são conhecidas por todos os outros participantes (através de certificados).

As principais ações que um participante malicioso pode executar para prejudicar a realização de BFT-CUP são: (*i.*) omitir o conhecimento sobre determinado(s) participante(s); (*ii.*) inventar a existência de participantes ou se fazer passar por outros participantes; (*iii.*) mandar dados diferentes para cada participante, tentando impedir que as propriedades do consenso sejam atendidas. Diante disto, as soluções para o BFT-CUP devem mascarar estas ações executadas por participantes maliciosos. As seções seguintes apresentam os protocolos propostos.

3.1. Detectores de Participação

É impossível resolver o consenso caso cada processo tenha conhecimento apenas sobre si próprio. Desta forma, as informações que um processo obtém sobre os demais participantes do sistema são capturadas através da noção de detectores de participação [Cavin et al. 2004], os quais são oráculos distribuídos que fornecem "dicas" aos processos sobre quais destes estão presentes na computação distribuída. Seja $i.PD$ o detector de participação do processo i, uma consulta a $i.PD$ retorna um subconjunto de processos de Π com os quais i pode colaborar, i.e., os vizinhos de i. Sendo $i.PD(t)$ o resultado de uma consulta de i no tempo t, a informação retornada por $i.PD$ pode evoluir entre consultas, mas segue as seguintes propriedades [Cavin et al. 2004]: *Inclusão da Informação*, a informação retornada pelo detector de participação não diminui com o tempo, i.e., $\forall i \in \Pi, \forall t' \geq t : i.PD(t) \subseteq i.PD(t')$; *Exatidão da Informação*, os detectores de participação não cometem erros, i.e., $\forall i \in \Pi, \forall t : i.PD(t) \subseteq \Pi$.

Os detectores de participação fornecem aos processos um contexto inicial sobre os participantes do sistema, através do qual é possível expandir o conhecimento sobre Π. Deste modo, cada participante obterá um grafo de conectividade por conhecimento que é orientado, pois a relação de conhecimento não é necessariamente bidirecional [Cavin et al. 2004].

Definição 1 *Grafo de conectividade por conhecimento: Seja $G_{di} = (V, \xi)$ um grafo orientado representando a relação de conhecimento induzida por um detector de participação PD. Então, $V = \Pi$ e $(i, j) \in \xi$ sse $j \in i.PD$, i.e., i conhece j.*

Definição 2 *Grafo não orientado de conectividade por conhecimento: Seja $G = (V, \xi)$ um grafo não orientado representando a relação de conhecimento induzida por um detector de participação PD. Então, $V = \Pi$ e $(i, j) \in \xi$ sse $j \in i.PD$ ou $i \in j.PD$.*

De acordo com as características observadas nos grafos de conhecimento, algumas classes de detectores de participação foram propostas para resolver o CUP [Cavin et al. 2004] e o FT-CUP [Greve and Tixeuil 2007]. A principal classe abordada na resolução do FT-CUP é:

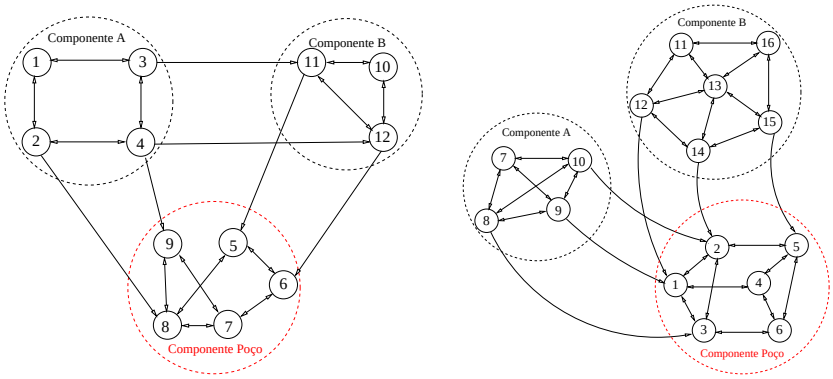
PD *k*-Redutível a Único Poço (*k*-OSR): O grafo *orientado* G_{di}, que representa a relação de conhecimento induzida pelo detector de participação PD, satisfaz as seguintes condições: (*i.*) o grafo de conectividade por conhecimento G, obtido de G_{di}, é *conexo*; (*ii.*) o grafo direcionado acíclico, obtido pela redução de G_{di} às suas componentes *k*-fortemente conexas[3], tem uma e somente uma *componente poço*[4]; (*iii.*) considere quaisquer duas componentes *k*-fortemente conexas G_1 e G_2, se existe um caminho de G_1 para G_2, então existem k *caminhos disjuntos* entre os vértices de G_1 para G_2.

A Figura 1 apresenta dois grafos G_{di} induzidos por detectores de participação pertencentes a classe *k*-OSR. As Figuras 1(a) e 1(b) correspondem a relações de conhecimento induzidas por detectores de participação 2-OSR e 3-OSR, respectivamente. Na solução que utiliza assina-

[3]Uma componente G_{fc} é *k-fortemente conexa* se para qualquer par de vértices (v_i,v_j) de G_{fc}, v_i pode alcançar v_j através de *k* caminhos disjuntos nos vértices.

[4]Uma componente G_p de G_{di} é considerada *poço* quando não existem caminhos saindo de vértices em G_p para vértices pertencentes a outras componentes de G_{di}.

turas, o retorno do detector de participação associado a um participante i é assinado pelos seus vizinhos, i.e., cada participante j assina a informação indicando que é vizinho de i ($\langle j \rangle_{\sigma j}$).



(a) 2-OSR (b) 3-OSR

Figura 1. Grafos de conectividade induzidos por detectores de participação.

3.2. Detector de Participação *k*-Redutível a Única Fonte (*k*-OFR)

Apesar de resolvermos o BFT-CUP através do PD k-OSR (seção 3.4), esta seção introduz uma nova classe de detectores de participação, chamada **PD *k*-Redutível a Única Fonte (*k*-OFR)**, assim definida: O grafo de conectividade por conhecimento G_{di} satisfaz as mesmas propriedades definidas para a classe *k*-OSR, sendo que a propriedade (*ii.*) deve ser alterada: (*ii.*) O grafo direcionado acíclico, obtido pela redução de G_{di} às suas componentes *k*-fortemente conexas, tem uma e somente uma *componente fonte*[5].

De acordo com a definição de *equivalência entre detectores de participação* [Cavin et al. 2004], os detectores *k*-OFR e *k*-OSR são equivalentes. Então, qualquer resultado obtido com um detector pode ser aplicado ao outro. A prova desta equivalência é baseada no fato da componente poço de um detector exercer o papel da componente fonte do outro, e vice-versa. Note que, o grafo de conhecimento apresentado na Figura 1(a) pode ser obtido também através de um PD 2-OFR (a componente A é fonte). Assim, o PD *k*-OFR também pode ser usado para resolver o BFT-CUP.

3.3. Protocolo de Disseminação

Esta seção discute como ocorre a comunicação entre os processos participantes do sistema. Duas soluções são propostas (com e sem a utilização de assinaturas digitais), onde consideramos que um canal confiável e autenticado é estabelecido entre os participantes vizinhos, possibilitando o envio de mensagens de forma confiável entre os nós vizinhos. A autenticidade das mensagens é comprovada através do uso ou de assinaturas ou de redundância de mensagens (dependendo da solução adotada), como será explicado a seguir. Além disso, apenas é necessário que este canal seja estabelecido entre os nós vizinhos. Deste modo, um participante não é capaz de mentir sobre sua identidade para um vizinho (canais autenticados), uma premissa fundamental para a realização de algoritmos tolerantes a faltas bizantinas [Castro and Liskov 2002].

A comunicação é feita através de duas primitivas: ***dependable_send(***$message$***,***$sender$***)*** - que faz com que um participante $sender$ envie uma mensagem $message$ para todos os participantes alcançáveis[6] a partir de $sender$, i.e., realiza um *flooding* da mensagem; e ***dependable_deliver(***$message$***,***$sender$***,***$routes$***)*** - chamada no participante receptor para entrega da mensagem $message$ difundida pelo participante $sender$, $routes$ contém as rotas (caminhos) através das quais $message$ foi recebida. Estas primitivas devem satisfazer as seguintes propriedades:

[5]Uma componente G_f de G_{di} é considerada *fonte* quando não existem caminhos saindo de vértices pertencentes a outras componentes de G_{di} para vértices em G_f.

[6]Uma participante q é alcançável por outro participante p se existir um número suficiente de caminhos disjuntos nos nós de p para q. Neste caso, q será um receptor da mensagem difundida por p.

- **Validade**: *Uma mensagem difundida por um participante correto $sender$ sempre acabará por ser entregue por algum participante correto alcançável a partir de $sender$ ou não existe nenhum participante correto alcançável por $sender$;*
- **Acordo**: *Se algum participante correto entregar a mensagem $message$, difundida por um participante correto $sender$, então todos os participantes corretos alcançáveis por $sender$ também entregarão $message$;*
- **Integridade**: *Para qualquer mensagem $message$, cada participante correto i entregará $message$ apenas se esta foi previamente difundida por algum participante $sender$, neste caso i é alcançável a partir de $sender$.*

O algoritmo 1 representa este protocolo de disseminação, onde estas primitivas são implementadas. Em nossas soluções, assumimos que o detector de participação de cada processo é consultado exatamente uma vez, o que contribui para que o grafo de conectividade por conhecimento obtido seja consistente, i.e., o conjunto de vizinhos não sofrerá alterações durante toda a execução do protocolo. Esta premissa pode ser implementada através do armazenamento da primeira consulta ao detector de participação e do retorno deste valor em consultas futuras.

Algoritmo 1 Algoritmo de comunicação executado pelo participante i

```
constant:
(1) f:int //número máximo de faltas suportadas

variables:
(2) i.use_signature:boolean //variável que indica o uso de assinaturas
(3) i.received_msgs:set of <message,route> tuples //conj.  de mensagens recebidas

message:
(4) DEPENDABLE_FLOODING: //estrutura da mensagem DEPENDABLE_FLOODING
(5)     message:value to flood // valor a ser difundido
(6)     route:ordered list of nodes //caminho percorrido pela mensagem

** Initiator Only **
procedure:
dependable_send(message,sender) // sender == i
(7)     if i.use_signature then
(8)         message = ⟨message⟩_{σ_i}; //i assina a mensagem
(9)     end if
(10)    for each j ∈ i.PD do
(11)        SEND DEPENDABLE_FLOODING(message,i) to j;
(12)    end for

** All Nodes **
INIT:
(13) i.use_signature = true/false; i.received_msgs = ∅;

(14) upon receipt of DEPENDABLE_FLOODING(m.message,m.route) from j
(15)    if getLastElement(m.route) == j ∧ i ∉ m.route then
(16)        addAtEnd(m.route,i);
(17)        initiator = getFirstElement(m.route);
(18)        if i.use_signature then
(19)            if verify_signature(⟨m.message⟩_{σ_initiator}) then
(20)                dependable_deliver(m.message,initiator,m.route);
(21)                for each z ∈ i.PD \ {j} do
(22)                    SEND DEPENDABLE_FLOODING(m.message,m.route) to z;
(23)                end for
(24)            end if
(25)        else
(26)            i.received_msgs = i.received_msgs ∪ {⟨m.message,m.route⟩};
(27)            routes = computeRoutes(m.message,i.received_msgs);
(28)            if (#⟨m.message,*⟩ ∈ i.received_msgs ≥ f + 1) ∧ (routes ≥ f + 1) then
(29)                dependable_deliver(m.message,initiator,routes);
(30)                i.received_msgs = i.received_msgs \ {⟨m.message,*⟩};
(31)            end if
(32)            for each z ∈ i.PD \ {j} do
(33)                SEND DEPENDABLE_FLOODING(m.message,m.route) to z;
(34)            end for
(35)        end if
(36)    end if
```

A idéia principal deste algoritmo é que os participantes façam *flood* de suas mensagens, atingindo todos os processos que podem alcançar. Desta forma, um participante que deseja difundir uma mensagem deve executar o procedimento $dependable\_send$. Neste procedimento, a mensagem é enviada a todos os seus vizinhos (linhas 10-12), caracterizando um

flooding, onde a mensagem recebida de um vizinho é encaminhada para os demais vizinhos e assim sucessivamente até alcançar todos os participantes possíveis (linhas 21-23 e 32-34). Uma característica importante desta difusão é o fato de cada mensagem ir acumulando a rota conforme o caminho percorrido até chegar em determinado destino. Um participante apenas considera determinada mensagem quando o processo que a está enviando (ou encaminhando) estiver adicionado no final da rota acumulada (linha 15). Esta solução é baseada na abordagem apresentada em [Dolev 1982], que impossibilita um participante malicioso de não se adicionar no final das informações de rota durante o envio (ou encaminhamento) de uma mensagem.

No entanto, um participante malicioso pode retirar ou adicionar participantes na rota acumulada (sejam eles corretos ou não), modificar a mensagem que está sendo propagada e até mesmo não retransmitir a mesma. Assim, é necessário que a conectividade do grafo que representa o conhecimento dos participantes seja suficientemente grande para fazer com que a mensagem chegue aos destinatários e que os mesmos possam provar sua autenticidade, possibilitando que o protocolo continue funcionando como especificado. Além disso, esta abordagem impossibilita que um participante malicioso invente mensagens que não foram difundidas por outros participantes, pois a autenticidade da mesma não será comprovada (ver adiante).

Conforme já comentado, para que um participante seja capaz de difundir suas mensagens apropriadamente, o grau de conectividade k do grafo que representa o conhecimento dos participantes deve ser suficientemente grande e varia de acordo com o uso ou não de assinaturas. Além disso, como as provas de corretude do algoritmo 1 diferem entre as duas soluções, estas abordagens serão estudadas separadamente.

Com assinaturas. Nesta abordagem, para que um participante i seja capaz de alcançar um participante p, devem existir no mínimo $f + 1$ caminhos disjuntos nos nós de i para p, i.e., o grau de conectividade k deve ser assumido como $k > f$. As linhas 19-24 do algoritmo 1 representam o processamento quando assinaturas são utilizadas. Assim, quando uma mensagem é recebida por algum participante p e sua assinatura é verificada (linha 19), comprovando a autenticidade da mesma, tal mensagem é entregue por p (linha 20), que também a encaminha para seus vizinhos (linhas 21-23).

Provas de Correção. *Validade*: Esta solução requer $k > f$. Desta forma, teremos no mínimo $f + 1$ caminhos disjuntos nos nós entre o emissor e os receptores (nós alcançáveis pelo emissor) de uma mensagem. Sendo assim, existe no mínimo um caminho formado apenas por participantes corretos, através do qual é garantido que a mensagem chegará aos receptores, os quais detectarão a autenticidade da mesma através da assinatura e a entregarão através da operação $dependable\_deliver$ (linha 20). *Acordo*: Como o acordo é definido sobre mensagens enviadas por participantes corretos, esta prova acaba se tornando idêntica a prova da propriedade de validade. *Integridade*: Como uma mensagem apenas é entregue após sua assinatura ser comprovada (linha 19), não é possível que um participante malicioso forje mensagens, i.e., apenas mensagens autênticas são entregues pelo protocolo. Sendo assim, toda mensagem entregue por algum participante correto foi previamente difundida pelo seu emissor ($sender$). □

Sem assinaturas. Nesta abordagem, para que um participante i seja capaz de alcançar um participante p, devem existir no mínimo $2f + 1$ caminhos disjuntos nos nós de i para p, i.e., o grau de conectividade k deve ser assumido como $k > 2f$. As linhas 26-34 do algoritmo 1 representam o processamento quando assinaturas não são utilizadas. Desta forma, uma mensagem recebida é armazenada em um conjunto (linha 26) e somente é entregue (linha 29) após a mesma mensagem ser recebida $f + 1$ vezes através de $f + 1$ caminhos disjuntos nos nós[7] (linhas 27-28), i.e., ter sua autenticidade comprovada. Além disso, sempre que uma mensagem é recebida por algum participante, o mesmo a encaminha para seus vizinhos (linhas 32-34).

[7] A função $computeRoute(m.message, i.received\_msgs)$ calcula o número de caminhos (rotas) disjuntos nos nós através dos quais $m.message$ foi recebida.

Provas de Correção. *Validade*: Esta solução requer $k > 2f$. Neste caso, teremos no mínimo $2f + 1$ caminhos disjuntos nos nós entre o emissor e os receptores (nós alcançáveis pelo emissor) de uma mensagem. Então, existe no mínimo $f + 1$ caminhos disjuntos nos nós formados apenas por participantes corretos, através dos quais é garantido que a mensagem chegará aos receptores, que detectarão a autenticidade da mesma através da redundância, i.e., do recebimento de $f + 1$ mensagens iguais através destes $f + 1$ caminhos disjuntos nos nós. Assim, após a autenticidade da mensagem ser comprovada, a mesma é entregue através da operação $dependable\_deliver$ (linha 29). *Acordo*: Como o acordo é definido sobre mensagens enviadas por participantes corretos, esta prova acaba se tornando idêntica a prova da propriedade de validade. *Integridade*: Uma mensagem apenas é entregue após ser recebida $f + 1$ vezes através de $f + 1$ caminhos disjuntos nos nós (linhas 27-28), o que garante que tal mensagem é autêntica, i.e., realmente foi enviada pelo emissor ($sender$). Desta forma, um participante malicioso i não é capaz de inventar que uma mensagem foi enviada por outro participante j (forjar emissor), pois a autenticidade da mesma não será comprovada. Isto é, um receptor r não será capaz de obter os $f + 1$ caminhos disjuntos nos nós de j para r. Mesmo com um conluio de até f participantes maliciosos, r obterá no máximo f caminhos disjuntos nos nós através dos quais recebeu a mensagem "enviada" por j. □

Uma característica encontrada em ambas as soluções é que uma mesma mensagem, enviada por determinado participante, pode ser entregue mais de uma vez pelos seus receptores. Esta característica não prejudica o uso deste protocolo em nosso algoritmo de consenso[8] (seção 3.4). Assim, não tratamos esta limitação no protocolo de comunicação desenvolvido. No entanto, esta entrega duplicada pode ser resolvida através do uso de *buffers* de armazenamento de mensagens já entregues juntamente com a adição de identificadores únicos nas mensagens.

3.3.1. Respondendo às Difusões

Considerando que as mensagens difundidas sejam requisições, e como os caminhos (rotas) percorridos pela mensagem (requisição) em uma determinada difusão são recebidos nos receptores ($dependable\_deliver$), é possível que os mesmos respondam a esta requisição enviando uma mensagem (resposta) pelos caminhos contrários (inversos) ao que recebeu tal requisição. Para isso, é necessário que o receptor de uma requisição descubra $f + 1$ (com assinaturas) ou $2f + 1$ (sem assinaturas) rotas através das quais uma requisição foi recebida, possibilitando que o envio da resposta seja completado mesmo na presença de até f processos faltosos. Deste modo, o grau de conectividade do grafo de conhecimento deve aumentar em f, i.e., $2f + 1$ (com assinaturas) ou $3f + 1$ (sem assinaturas). Assim, um recepetor deverá esperar por $f + 1$ rotas quando utiliza assinaturas (linha 19) ou $2f + 1$ rotas na solução sem assinaturas (linha 28). Note que, esta conectividade apenas é necessária para possibilitar que um receptor responda a uma difusão. Esta abordagem é utilizada em nossos protocolos (seção 3.4), que requerem este grau de conectividade mesmo que os receptores não necessitem respoder à difusões (seção 3.4.2).

Duas primitivas definem o envio da resposta: ***dependable_reply_send(***$message$***,***$sender$***,***$routes$***)*** - usada pelo receptor de uma requisição ($sender$), que responde fazendo com que $message$ percorra os caminhos ($routes$) inversos; e ***dependable_reply_deliver(***$message$***,***$sender$***)*** - chamada no receptor da resposta $message$ (difusor da requisição), para entrega da mesma, que foi enviada por $sender$ (receptor da requisição). A implementação destas primitivas segue os mesmos moldes da implementação da difusão (algoritmo 1), obdecendo às mesmas propriedades. No entanto, é mais simples pelo fato das rotas já estarem predefinidas.

[8]Para que protocolos suportem esta característica, basta considerar apenas a primeira mensagem recebida em uma dada difusão, como é o caso dos nossos algoritmos (seção 3.4).

3.4. Resolvendo o BFT-CUP com o Detector de Participação *k*-OSR

Esta seção apresenta um protocolo para a resolução do BFT-CUP através de um detector de participação *k*-OSR. Como comentado na seção anterior, nossos protocolos requerem $3f+1$ (ou $2f+1$ com assinaturas) caminhos disjuntos nos nós para a descoberta dos participantes (seção 3.4.2). Assim, primeiramente é provado que esta conectividade e este detector de participação são necessários para resolver o BFT-CUP, sendo que suas suficiências são comprovadas através das provas de corretude dos algoritmos apresentados.

Teorema 1 *O detector de participação PD, com $PD \in k\text{-}OSR$ ou equivalente*[9]*, é necessário para resolver o BFT-CUP, dado que f nós podem falhar, onde $f < \frac{k}{2} < n$ (usando assinaturas) ou $f < \frac{k}{3} < n$ (não usando assinaturas).*

Prova: Considere por contradição que existe um algoritmo que resolva o BFT-CUP através de um detector de participação $PD \notin k\text{-}OSR$. Seja G_{di} o grafo de conhecimento induzido por PD, dois cenários são possíveis: (*i.*) ou existem menos de k caminhos disjuntos nos nós conectando um participante p em G_{di} ou (*ii.*) a decomposição de G_{di} em suas componentes *k*-fortemente conexas resulta em mais de um poço. No primeiro cenário, a falha de f nós impossibilitará que p descubra de forma correta os demais participantes do sistema. Esta prova é simples e pode ser assim entendida: considere uma consulta realizada por p em um conjunto de n_c processos, p deve aguardar por $n_c - f$ respostas (pois f processos podem falhar), no entanto, as respostas dos f faltosos podem estar no conjunto de respostas recebidas, assim, é garantido que p recebeu respostas de $n_c - 2f$ processos corretos, sendo que para a consulta ser corretamente realizada é necessário a presença de pelo menos um processo correto caso assinaturas sejam usadas ($n_c - 2f \geq 1$) ou $f + 1$ processos corretos quando não usamos assinaturas ($n_c - 2f \geq f + 1$). Assim, em ambas as situações, chega-se a uma contradição. No segundo cenário, sejam G_1 e G_2 duas das componentes poço e considerando que os participantes em G_1 têm valor de proposição v e os em G_2 valor w, sendo $v \neq w$. Pela propriedade de *terminação* do consenso, os nós em G_1 decidem em um tempo t_1 e os nós em G_2 em um tempo t_2. Atrasando o tempo de recepção de mensagens provenientes de outras componentes para todos os nós em G_1 e G_2 para um tempo $t > max\{t_1, t_2\}$. Como os nós nas componentes poço não sabem da existência de outros nós, pela propriedade da *validade* do consenso, os nós em G_1 decidem pelo valor v e os nós em G_2 pelo valor w, violando a propriedade de *acordo* do consenso ($v \neq w$), chegando-se a uma contradição. □

3.4.1. Visão Geral do Algoritmo

O protocolo para resolução do BFT-CUP tem como base o algoritmo de disseminação apresentado no seção 3.3, que engloba todos os detalhes relacionados com a comunicação entre os participantes do sistema. A partir daí, inspirando-se em [Greve and Tixeuil 2007], o protocolo de consenso é dividido em três fases. Na primeira fase, de *descoberta de participantes* (seção 3.4.2), cada participante do sistema aumentará o seu conhecimento a respeito dos outros participantes, descobrindo o número máximo possível de participantes que estão presentes em determinada computação. A segunda fase, de *determinação da componente poço* (seção 3.4.3), tem como objetivo definir quais são os participantes que pertencem à componente poço do grafo de conhecimento gerado a partir de um PD *k*-OSR. Assim, cada participante é capaz de determinar se pertence ou não à componente poço. Na última fase (seção 3.4.4), os membros da componente poço *executam um consenso tolerante a faltas bizantinas* clássico e propagam o valor de decisão aos demais participantes do sistema. Note que, a componente poço deve conter um número de participantes $n_{sink} \geq 3f + 1$, necessário para a resolução de algum algoritmo de consenso tolerante a faltas bizantinas clássico (ex.: Paxos Bizantino [Castro and Liskov 2002]).

[9] A seção 3.2 apresenta o detector de participação redutível a uma única fonte (*k*-OFR), juntamente com sua equivalência ao detector *k*-OSR. Assim, ambos podem ser empregados na solução do BFT-CUP.

3.4.2. Descoberta dos Participantes

Antes que qualquer computação possa ser iniciada, é necessário que os participantes do sistema obtenham o máximo de conhecimento possível a respeito dos outros participantes. Note que, através de seu detector de participação local, um processo é capaz de determinar quem são seus vizinhos. No entanto, este conhecimento não é suficiente para resolver o BFT-CUP, sendo necessária a sua expansão. Neste sentido, o algoritmo 2 (DISCOVERY) realiza este procedimento. A idéia principal é que cada participante i difunda uma mensagem solicitando informações sobre os vizinhos de cada participante alcançável por i, realizando uma espécie de busca em largura no grafo de conhecimento. No final do algoritmo, i obterá o conjunto maximal de participantes alcançáveis, que representa os participantes conhecidos por i.

Na inicialização do algoritmo em um processo i, o conjunto de conhecidos é atualizado para o próprio i e seus vizinhos e o conjunto de mensagens pendentes (se $j \in i.msg\_pend$, então i deve receber uma mensagem de j) é atualizado para os vizinhos de i (linha 6). Além disso, uma mensagem é difundida para todos os participantes alcançáveis por i (linha 7) solicitando informações a respeito de seus vizinhos. Quando esta mensagem, difundida através do protocolo de disseminação discutido na seção 3.3, é entregue em determinado participante, o mesmo responde para i com seu conjunto de vizinhos (linhas 8-9).

Algoritmo 2 Algoritmo DISCOVERY executado pelo participante i

```
constant:
(1) f:int //número máximo de faltas suportadas

variables:
(2) i.known:set of nodes //conjunto de processos conhecidos por i
(3) i.nei_pend:set of ⟨node,node.neighbor⟩ tuples//i não conhece todos os vizinhos de node
(4) i.msg_pend:set of nodes //conj.  de nós que i espera por mensagens (respostas)
(5) i.use_signature:boolean //variável que indica o uso de assinaturas

** All Nodes **
INIT:
(6) i.known = {i} ∪ i.PD ;i.nei_pend = ∅; i.msg_pend = i.PD;
(7) dependable_send(GET_NEIGHBOR,i);

(8) upon execution of dependable_deliver(GET_NEIGHBOR,sender,routes)
(9)  dependable_reply_send(i.PD,i,routes);

(10) upon execution of dependable_reply_deliver(sender.neighbor,sender)
(11)  i.known = i.known ∪ {sender};
(12)  i.nei_pend = i.nei_pend ∪ {⟨sender,sender.neighbor⟩};
(13)  i.msg_pend = i.msg_pend \ {sender};
(14)  if i.use_signature  then
(15)    for each ⟨j⟩_σj ∈ sender.neighbor do
(16)      if(verify_signature(⟨j⟩_σj) ∨ #_⟨*,⟨j⟩⟩ ∈ i.nei_pend > f ) ∧ j ∉ i.known then
(17)         i.known = i.known ∪ {j};
(18)         i.msg_pend = i.msg_pend ∪ {j};
(19)      end if
(20)    end for
(21)  else
(22)    if  (∃j : #_⟨*,⟨j⟩⟩ ∈ i.nei_pend > f) ∧ j ∉ i.known then
(23)     i.known = i.known ∪ {j};
(24)     i.msg_pend = i.msg_pend ∪ {j};
(25)    end if
(26)  end if
(27)  for each ⟨j,j.neighbor⟩ ∈ i.nei_pend  do
(28)    if  (∀⟨z⟩ ∈ j.neighbor ⟶ z ∈ i.known) then
(29)     i.nei_pend = i.nei_pend \ {⟨j,j.neighbor⟩};
(30)    end if
(31)  end for
(32)  if  (|i.nei_pend| + |i.msg_pend|) ≤ f  then
(33)    return(i.known);;
```

Na computação das respostas em um processo i, o seu conjunto de conhecidos é atualizado, juntamente com os conjuntos de vizinhos pendentes (se $\langle j, j.neighbor\rangle \in i.nei\_pend$, então i conhece j mas não conhece todos os vizinhos de j[10]) e de mensagens pendentes. Além disso, é determinado se i adquiriu conhecimento sobre algum outro participante j (linhas 14-26), i.e., se recebeu esta informação assinada por j ou $f + 1$ outros participantes conhecidos de i informaram que têm j como vizinho. Após estas verificações, o conjunto de vizinhos pendentes é atualizado (linhas 27-30). Para determinar se falta conhecer algum participante, i utiliza

[10]Como i alcança j, i também alcança os vizinhos de j e deve aguardar por suas respostas à difusão inicial de i.

os conjuntos $i.msg\_pend$ e $i.nei\_pend$, que indicam as pendências relacionadas com as mensagens (respostas) recebidas por i (linhas 32-33). O algoritmo termina retornando o conjunto de participantes descobertos por i (linha 33), o qual contém todos os participantes (corretos ou faltosos) alcançáveis a partir de i. Este processamento pode ser entendido como uma busca em largura realizada por i no grafo de conhecimento.

Lema 1 *Seja G_{di} um grafo de conhecimento induzido por um detector de participação e dado que f nós podem falhar, onde $f < \frac{k}{2} < n$ (usando assinaturas) ou $f < \frac{k}{3} < n$ (não usando assinaturas). O algoritmo DISCOVERY (algoritmo 2), executado por cada participante p do sistema, satisfaz as seguintes propriedades:*

- Terminação: *p termina a execução e retorna uma lista contendo nós conhecidos de p;*
- Exatidão: *o algoritmo DISCOVERY retorna o conjunto maximal de nós alcançáveis (conhecidos) por p em G_{di}.*

Prova: *Terminação.* O algoritmo termina quando p receber mensagens (respostas) de pelo menos todos os processos corretos alcançáveis (linha 32), como Π é finito e através das propriedades do protocolo de disseminação (seção 3.3), é garantido que esta condição acabará por ser satisfeita. *Exatidão.* O algoritmo apenas termina quando restarem no máximo f pendências, as quais podem estar divididas entre processos que informam vizinhos que não existem no sistema ($i.nei\_pend$) e processos dos quais p ainda não recebeu mensagens/respostas ($i.msg\_pend$). Além disso, na solução sem assinaturas (resp. com assinaturas) cada participante z (sendo z alcançável por p) do sistema é vizinho de pelo menos $3f+1$ (resp. $2f+1$) outros participantes, pois $f < \frac{k}{3} < n$ (resp. $f < \frac{k}{2} < n$). Desta forma, caso z seja malicioso, p computa mensagens (respostas) de no mínimo $2f + 1$ (resp. $f + 1$) vizinhos corretos de z, descobrindo z (linhas 16 e 22). Se z for correto, no pior caso p não computa mensagens de f vizinhos corretos de z ($i.msg\_pend$) e outros f vizinhos de z são faltosos, mas ainda assim, p computará $f + 1$ (resp. 1) mensagens de vizinhos corretos de z, descobrindo z (linhas 16 e 22). □

3.4.3. Determinação da Componente Poço

Esta fase visa estabelecer quais são os processos corretos que pertencem a componente poço do grafo de conhecimento induzido por um detector de participação *k*-OSR. Mais precisamente, através do processamento do algoritmo 3 (SINK), cada participante é capaz de determinar se é membro da componente poço deste grafo. A idéia de funcionamento deste algoritmo é que após a execução do procedimento DISCOVERY, os membros da componente poço obterão uma visão parcial do sistema que é necessariamente menor do que o conhecimento obtido pelos demais participantes, que conhecerão pelo menos os membros da componente a que pertencem e os membros da componente poço.

Na inicialização do algoritmo em um processo i, i executa o procedimento DISCOVERY com o intuito de estabelecer suas relações de conhecimento (linha 6) e envia uma mensagem com seu conjunto de processos conhecidos para todos os participantes alcançáveis/conhecidos (linha 7). Quando estas mensagens são entregues por algum participante, o mesmo responde para i com *ack* caso seu conjunto de conhecidos for igual ao de i (pertencem a mesma componente). Caso contrário, a resposta enviada para i é um *nack* (linhas 8-13).

No processamento das respostas em i (linha 14-26), i atualiza o conjunto de processos que responderam (linha 15). Além disso, caso a resposta recebida seja *nack*, o conjunto de processos que pertencem a outras componentes é atualizado (linha 17) e caso o número de processos que não pertencem a mesma componente de i seja maior do que f (linha 18), i conclui que não pertence à componente poço (linhas 18-20). No entanto, caso i receba respostas de todos os processos conhecidos, excluindo-se os possíveis faltosos (linha 23), e o número de processos que pertencem a outras componentes ($i.nacked$) não é maior do que f, i conclui que pertence à componente poço (linhas 24-25).

Lema 2 *Considerando um detector de participação k-OSR (ou equivalente) e dado que f nós podem falhar, onde $f < \frac{k}{2} < n$ (usando assinaturas) ou $f < \frac{k}{3} < n$ (não usando assinaturas). O algoritmo SINK (algoritmo 3), executado por cada participante p do sistema que possuí no mínimo $3f + 1$ nós na componente poço, satisfaz as seguintes propriedades:*

- Terminação: *p termina a execução determinando se pertence ou não à componente poço;*
- Exatidão: *p é membro da componente poço sse o algoritmo SINK retornar true.*

Algoritmo 3 Algoritmo SINK executado pelo participante i

```
constant:
(1) f:int //número máximo de faltas suportadas

variables:
(2) i.known:set of nodes //conjunto de processos conhecidos de i
(3) i.responded:set of nodes //conj.  de processos que se comunicaram com i
(4) i.nacked:set of nodes //conj.  de processos que ∉ compomente de i
(5) i.in_the_sink:boolean //variável indicando se i ∈ poço

** All Nodes **
INIT:
(6) i.known = DISCOVERY(); i.responded {i}; i.nacked = ∅;
(7) dependable_send(i.known,i);

(8) upon execution of dependable_deliver(sender.known,sender,routes)
(9)     if i.known == sender.known then
(10)      dependable_reply_send(ack,i,routes)
(11)    else
(12)      dependable_reply_send(nack,i,routes)
(13)    end if

(14) upon execution of dependable_reply_deliver(reply,sender)
(15)    i.responded = i.responded ∪ {sender}
(16)    if reply == nack then
(17)      i.nacked = i.nacked ∪ {sender};
(18)      if|i.nacked| ≥ f+1 then
(19)        i.in_the_sink = false;
(20)        return(i.in_the_sink,i.known);
(21)      end if
(22)    end if
(23)    if |i.responded| ≥ |i.known| - f  then
(24)      i.in_the_sink = true;
(25)      return(i.in_the_sink,i.known);
(26)    end if
```

Prova: *Terminação*. Em cada participante p, o algoritmo retorna quando p receber ou (*i.*) $f + 1$ respostas de processos de outras componentes (linha 18) ou (*ii.*) respostas de pelo menos todos os processos corretos determinados na inicialização (linha 23). Através das propriedades do protocolo de comunicação (seção 3.3), mesmo que $f < \frac{k}{2}$ ou $f < \frac{k}{3}$ (dependendo do uso de assinaturas) participantes falhem, é garantido que ou (*i.*) ou (*ii.*) sempre ocorrerá. *Exatidão*. Através do Lema 1, é garantido que ao final do procedimento DISCOVERY todos os processos corretos de uma mesma componente obtém o mesmo conhecimento. Assim, como os membros da componente poço apenas recebem respostas de membros da própria componente, é garantido que estes participantes concluam corretamente (linha 23). Além disso, os membros de uma componente não poço computarão no mínimo $f + 1$ mensagens de membros corretos da componente poço (que têm necessariamente um conhecimento menor sobre Π - Lema 1), pois adquiriram o conhecimento sobre pelo menos os $2f + 1$ participantes corretos da componente poço (Lema 1). Assim, através destas $f + 1$ mensagens os membros de componentes não poço concluirão corretamente (linha 18). □

3.4.4. Realização do Consenso

Esta é a etapa final na execução do consenso. A idéia principal é que os membros da componente poço executem um consenso bizantino clássico (ex. Paxos Bizantino [Castro and Liskov 2002]) e enviem o valor da decisão para os demais participantes do sistema. A resiliência ótima para algoritmos de consenso bizantino clássico é $3f + 1$ [Castro and Liskov 2002]. Sendo assim, é necessário a presença de no mínimo $3f + 1$ participantes na componente poço.

O algoritmo 4 (CONSENSUS) representa este processamento. Na inicialização, cada participante determina se pertence à componente poço e obtêm suas relações de conhecimento

(procedimento SINK - linha 9). Dependendo do resultado obtido, dois comportamentos distintos são possíveis: **(1)** Os membros da componente poço executam um consenso bizantino clássico (linha 11) e enviam a decisão para os demais participantes (linhas 17-19 e 21-26). Por construção, os membros corretos da componente poço estão presentes na visão de cada outro membro desta componente (Lema 1). Assim, como cada membro da componente poço conhece no mínimo $2f + 1$ participantes corretos (membros da mesma componente), o acordo do protocolo de consenso bizantino clássico sempre será obtido, juntamente com as outras propriedades definidas para estes protocolos de consenso [Castro and Liskov 2002]. Note que, as trocas de mensagens realizadas durante o estabelecimento deste consenso podem ser implementadas através do protocolo de comunicação definido na seção 3.3. **(2)** Os membros das outras componentes solicitam o valor da decisão aos participantes conhecidos, i.e., participantes alcançáveis, que inclui os membros do poço (linha 13). Um participante decidirá por um valor $value$ somente após receber $value$ de no mínimo $f + 1$ outros participantes, garantindo a presença de no mínimo um participante correto (linhas 27-34).

Algoritmo 4 Algoritmo CONSENSUS executado pelo participante i

```
constant:
(1) f:int //número máximo de faltas suportadas

input:
(2) i.initial:value //valor a ser proposto por i (input)

variables:
(3) i.in_the_sink:boolean //variável indicando se i ∈ poço
(4) i.known:set of nodes //conjunto de processos conhecidos de i
(5) i.decision:value //valor de decisão
(6) i.asked:set of ⟨node,routes⟩ tuples//conj.  de processos que solicitaram o valor de decisão
(7) i.values:set of ⟨node,value⟩ tuples //conjunto de decisões de outros nós

** All Nodes **
INIT: {Main Decision Task}
(8)    i.decision = ⊥; i.values = i.asked = ∅;
(9)    (i.in_the_sink,i.known) = SINK();
(10)   if i.in_the_sink then {Underline classical byzantine consensus with all p ∈ i.known}
(11)     Consensus.propose(i.initial);
(12)   else
(13)     dependable_send(GET_DECISION,i);
(14)   end if

** Node In Sink **
(15) upon Consensus.decide(v)
(16)    i.decision = v;
(17)    for each ⟨j,routes⟩ ∈ i.asked  do
(18)      dependable_reply_send(i.decision,i,routes);
(19)    end for
(20)    return(i.decision);

(21) upon execution of dependable_deliver(GET_DECISION,sender,routes)
(22)    if i.decision == ⊥ then
(23)       i.asked = i.asked ∪ {⟨sender,routes⟩};
(24)    else
(25)       dependable_reply_send(i.decision,i,routes);
(26)    end if

** Node Not In Sink **
(27) upon execution of dependable_reply_deliver(value,sender)
(28)    if i.decision == ⊥ then
(29)       i.values = i.values ∪ {⟨sender,value⟩};
(30)       if #⟨∗,value⟩ ∈ i.values ≥ f+1 then
(31)          i.decision = value;
(32)          return(i.decision);
(33)       end if
(34)    end if
```

Teorema 2 *O detector de participação k-OSR (ou equivalente) é suficiente para resolver o BFT-CUP, dado que f nós podem falhar em um sistema parcialmente síncrono que possuí no mínimo $3f + 1$ nós na componente poço, onde $f < \frac{k}{2} < n$ (usando assinaturas) ou $f < \frac{k}{3} < n$ (não usando assinaturas).*

Prova: Todos os participantes corretos da componente poço determinam esta condição (Lema 2) e participam do algoritmo de consenso bizantino clássico. Sendo assim, como esta componente possui no mínimo $2f + 1$ participantes corretos e o sistema é parcialmente síncrono, é garantido que as propriedades do consenso clássico serão atendidas. Desta forma, os membros

da componente poço obtém a decisão do consenso (linha 15), enviam esta decisão para os demais participantes (linha 17-19 e 21-26) e retornam o valor da decisão (linha 20) *terminado* o algoritmo. Como a componente poço possui no mínimo $2f + 1$ participantes corretos, que respondem às solicitações dos demais participantes[11] mesmo que $f < \frac{k}{2}$ ou $f < \frac{k}{3}$ (dependendo do uso de assinaturas) participantes falhem (seção 3.3), é garantido que os membros das outras componentes receberão $f + 1$ mensagens contendo o mesmo valor de decisão, podendo *terminar* decidindo por este valor (linhas 27-34). A *integridade* é garantida através do teste da linha 28, onde cada participante correto decide apenas caso ainda não tenha decidido. Além disso, um conluio entre os f possíveis participantes faltosos não será suficiente para fazer com que algum processo decida por valores incorretos, garantindo a propriedade de *acordo* do consenso. Note que, a propriedade de *validade* do consenso é conseguida através do protocolo de consenso clássico subjacente, i.e., o valor de decisão será o valor proposto por algum membro da componente poço. Já as propriedades de *acordo*, *terminação* e *integridade* são estabelecidas através do protocolo de consenso subjacente em conjunto com o algoritmo CONSENSUS. Esta prova de corretude do algoritmo CONSENSUS atesta que o detector de participação *k*-OSR é suficiente para resolver o BFT-CUP. □

4. Trabalhos Relacionados

A maioria dos trabalhos sobre consenso encontrados na literatura considera que o conjunto de processos do sistema é estático e previamente conhecido por todos os participantes do sistema. No entanto, nos últimos anos surgiram algumas pesquisas visando o estudo deste problema em ambientes dinâmicos, onde o número de participantes e suas identidades são, *a priori*, desconhecidos. Neste sentido, o primeiro trabalho a abordar o consenso em ambientes dinâmicos foi [Cavin et al. 2004], que resolve este problema para uma rede assíncrona onde os processos não podem falhar (CUP). Além disso, este trabalho define que o detector de participação OSR (redutível a único poço) é necessário e suficiente para resolver o CUP nestes ambientes. Em [Cavin et al. 2005] este modelo é estendido para tolerar faltas de parada nos participantes do sistema (FT-CUP), que funciona corretamente desde que as falhas obedeçam a um padrão de falhas, i.e., os processos podem falhar desde que o grafo de conhecimento continue pertencendo a classe OSR, sendo que as falhas devem ser detectadas por meio de detectores de faltas perfeitos [Chandra and Toueg 1996].

Um estudo comparando as condições de conectividade do grafo de conhecimento e as condições de sincronia do sistema é apresentado em [Greve and Tixeuil 2007], onde é estabelecido que o FT-CUP admite solução em redes assíncronas enriquecidas com o mais fraco detector de faltas capaz de resolver o consenso, desde que se aumente o grau de conhecimento entre os participantes (k-OSR). Nosso trabalho estende esse resultado mostrando que uma conectividade maior é requerida quando os processos podem falhar de forma bizantina (BFT-CUP) em um sistema parcialmente sincrono (sincronia mínima requerida para a realização do $\diamond S$ [Chandra and Toueg 1996]). A tabela 1 apresenta uma comparação entre as exigências de conectividade e sincronismo de cada solução para o consenso com participantes desconhecidos.

5. Conclusões

Este trabalho apresenta duas soluções para o consenso bizantino entre participantes desconhecidos (BFT-CUP), com e sem o uso de assinaturas digitais. Para cada solução, é provado quais são as condições necessárias e suficientes para resolver o BFT-CUP. A solução que utiliza assinaturas exige o uso de certificados emitidos por uma autoridade certificadora reconhecida pelos participantes, mas requer menor conectividade no grafo de conhecimento e menor número de envio de mensagens entre os participantes, resultando em menor custo de transmissão de dados.

[11] Note que todos os participantes do sistema alcançam os participantes da componente poço.

O modelo adotado neste trabalho, bem como os empregados nas soluções do FT-CUP [Cavin et al. 2005, Greve and Tixeuil 2007], suporta a mobilidade dos nós, mas não é forte o suficiente para tolerar entradas e saídas arbitrárias (as saídas podem ser computadas como faltas). Note que, após as relações de conhecimento serem estabelecidas, novos nós apenas serão considerados em execuções futuras do consenso.

Solução	modelo de faltas	detector de participação	k	participantes no poço/fonte	conectividade entre componentes	sincronismo
CUP [Cavin et al. 2004]	sem faltas	OSR	–	1	OSR	assíncrono
FT-CUP [Cavin et al. 2005]	paradas	OSR	–	1	OSR + padrão de falhas	assíncrono + P
FT-CUP [Greve and Tixeuil 2007]	paradas	k-OSR	$f+1$	$2f+1$	k caminhos disjuntos nos nós	assíncrono + $\diamondsuit S$
BFT-CUP com assinatura (este artigo)	bizantinas	k-OSR ou k-OFR	$2f+1$	$3f+1$	k caminhos disjuntos nos nós	parcialmente síncrono
BFT-CUP sem assinatura (este artigo)	bizantinas	k-OSR ou k-OFR	$3f+1$	$3f+1$	k caminhos disjuntos nos nós	parcialmente síncrono

Tabela 1. Características das soluções do consenso com participantes desconhecidos.

O principal resultado deste artigo é demonstrar que com o mesmo detector de participação e sincronia usada para resolver consenso com faltas por parada em sistemas auto-organizáveis podemos resolver este problema com faltas bizantinas, desde que se aumente a conectividade do grafo e o número de elementos em seu poço (ou fonte). Além disso, definimos um protocolo de disseminação tolerante a faltas bizantinas que é empregado em nossos protocolos e interessante por si só, podendo ser utilizado em outros protocolos para redes auto-organizáveis.

Referências

Awerbuch, B., Holmer, D., Nita-Rotaru, C., and Rubens, H. (2002). An on-demand secure routing protocol resilient to byzantine failures. In *WiSE '02: Proceedings of the 1st ACM workshop on Wireless security*, pages 21–30, New York, NY, USA. ACM.

Castro, M., Druschel, P., Ganesh, A., Rowstron, A., and Wallach, D. S. (2002). Secure routing for structured peer-to-peer overlay networks. *SIGOPS Oper. Syst. Rev.*, 36(SI):299–314.

Castro, M. and Liskov, B. (2002). Practical Byzantine fault-tolerance and proactive recovery. *ACM Transactions on Computer Systems*, 20(4):398–461.

Cavin, D., Sasson, Y., and Schiper, A. (2004). Consensus with unknown participants or fundamental self-organization. In *Proc. of the 3rd Int. Conf. on Ad hoc Networks and Wireless*, pages 135–148.

Cavin, D., Sasson, Y., and Schiper, A. (2005). Reaching agreement with unknown participants in mobile self-organized networks in spite of process crashes. Technical Report IC/2005/026, EPFL - LSR.

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

Dolev, D. (1982). The Byzantine generals strike again. *Journal of Algorithms*, (3):14–30.

Dwork, C., Lynch, N. A., and Stockmeyer, L. (1988). Consensus in the presence of partial synchrony. *Journal of ACM*, 35(2):288–322.

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

Greve, F. G. P. and Tixeuil, S. (2007). Knowledge connectivity vs. synchrony requirements for fault-tolerant agreement in unknown networks. In *Proc. of the Int. Conf. on Dependable Systems and Networks - DSN 2007*, pages 82–91.

Lamport, L., Shostak, R., and Pease, M. (1982). The Byzantine generals problem. *ACM Transactions on Programing Languages and Systems*, 4(3):382–401.