UNIVERSIDADE DE LISBOA
Faculdade de Ciências
Departamento de Informática

Comunicação e filiação em redes ad-hoc móveis com participantes desconhecidos

Emanuel José Branco Alves

DISSERTAÇÃO

MESTRADO EM ENGENHARIA INFORMÁTICA
Especialização em Arquitectura, Sistemas e Redes de Computadores

2013

UNIVERSIDADE DE LISBOA
Faculdade de Ciências
Departamento de Informática

Comunicação e filiação em redes ad-hoc móveis com participantes desconhecidos

Emanuel José Branco Alves

DISSERTAÇÃO

MESTRADO EM ENGENHARIA INFORMÁTICA
Especialização em Arquitectura, Sistemas e Redes de Computadores

Dissertação orientada pelo Prof. Doutor Nuno Fuentecilla Maia Ferreira Neves
e co-orientado pelo Prof. Doutor Alysson Neves Bessani

2013

Agradecimentos

Em primeiro lugar, quero agradecer ao meu orientador e mentor, Doutor Nuno Fuentecilla Maia Ferreira Neves, por todo o apoio e acompanhamento neste trabalho, assim como a disponibilidade, a simpatia e compreensão sempre demonstrada. Quero agradecer ao meu co-orientandor, Doutor Alysson Neves Bessani, pelo esclarecimento de dúvidas. E por fim, e não menos importante, quero também agradecer ao Doutor Hugo Alexandre Tavares Miranda pelos seus ensinamentos e conselhos sobre as redes ad hoc sem fios móveis. O meu muito obrigado aos três!

Gostaria também de agradecer aos restantes professores e colegas do grupo de investigação Navigatores/LaSIGE pela boa disposição.

Não posso deixar de referir também os colegas que trabalharam no laboratório mais famoso da faculdade: André Guerreiro, Cristiano Iria, Fábio Botelho, Hugo Sousa, Igor Antunes, João Alves, José Sá, Marcos Vasco, Pedro Costa e ao Ruben Campos (e o seu skate), que contribuíram para boa disposição no local de trabalho, nos cafés na hora do almoço e nos "chás" das 17 horas na ponte do C1. Quero elogiar e agradecer ao meu amigo, Miguel Garcia, pelos seus conselhos.

Aos restantes colegas do DI que conheci na FCUL, obrigado pelos bons momentos que passámos.

Obrigado a todos os meus amigos que me acompanham há já longos anos, principalmente à Marta Ferraz Dias, por oferecer sua a amizade incondicionalmente.

Não posso deixar de agradecer aos meus pais que lutaram desde sempre para entrar na faculdade.

Quero agradecer pelo suporte parcialmente dado pela CE através do projeto FP7-257475 (MASSIF), pela FCT através do programa Multianual (LaSIGE) e pelo projeto PTDC/EIA-EIA/113729/2009 (SITAN).

E por último, mas não menos importante, agradeço à minha perdição, Sofia Mendo, por toda a paciência e compreensão, pelos chocolates e por ter-me acompanhado desde sempre neste longo e árduo percurso.

Para o meu querido irmão.

Resumo

A proliferação de dispositivos sem fios de variadas capacidades (e.g., smartphones e tablets) levaram ao aumento do desenvolvimento de aplicações móveis. Algumas aplicações exploram protocolos de redes sem fios e a mobilidade dos dispositivos com o objetivo de aumentar as suas funcionalidades. A classe de protocolos de redes sem fios com dispositivos móveis chama-se MANET. Devido ao meio de comunicação, as redes MANET têm um conjunto de vulnerabilidades e limitações que dificultam a concretização de aplicações distribuídas. Contudo, o uso de um middleware que suporte a comunicação e a coordenação entre dispositivos, i.e., participantes, ainda que na presença de faltas bizantinas e participantes desconhecidos, seria extremamente útil para a construção destas aplicações. Este trabalho descreve a concretização e otimização de uma pilha de protocolos para a comunicação e coordenação em redes MANET. A pilha protocolar foi avaliada no simulador *ns-3*, em vários ambientes e condições de configuração, mostrando que esta pode efetivamente ser executada num conjunto interessante de cenários.

Palavras-chave: MANET, Filiação, Acordo mútuo, Participantes bizantinos, Participantes desconhecidos

Abstract

The proliferation of wireless devices of several capabilities (e.g., smartphones and tablets) led to an increasing development of mobile applications. Some applications explore protocols for wireless networks and mobile devices to improve their functionality. The class of protocols for wireless networks with mobile devices is called MANET. Due to the medium, MANETs have a set of vulnerabilities and limitations that hinder the implementation of distributed applications. However, the use of a middleware that supports communication and coordination between devices, i.e., participants, even in the presence of Byzantine and unknown participants, would be extremely useful for the construction of these applications. This work describes the implemention and optimization of a protocol stack with communication protocols and coordination support in MANETs. The stack was evaluated in the simulator *ns-3* in various configurations simulating multiple execution environments, showing that the protocol stack can effectively be executed in a set of interesting scenarios.

Keywords: MANET, Groupmembership, Consensus, Byzantine Participants, Unknown Participants

Conteúdo

Lista de Figuras

Lista de Tabelas

Capítulo 1

Introdução

A proliferação de dispositivos sem fios com diversas capacidades (e.g., "smartphones" e "tablets") levaram ao aumento do desenvolvimento de aplicações móveis. Os dispositivos têm um conjunto de características em comum: interfaces de rede sem fios de curto alcance, poder computacional reduzido, apresentam alguma mobilidade e são relativamente baratos. Juntamente com a proliferação dos dispositivos, surgiram várias aplicações interessantes, como as plataformas de disseminação de ficheiros, as agendas distribuídas e o trabalho colaborativo [25]. Algumas das aplicações dependem da comunicação com sistemas centralizados, normalmente distantes do utilizador, para efetuar a coordenação dos dispositivos móveis quando não é possível efetuar diretamente entre os participantes. As aplicações necessitam de comunicação mas, por vezes, esta apenas é possível recorrendo a redes convencionais encontradas em habitações, locais de trabalho ou locais públicos. Estas redes requerem uma infraestrutura fixa com administração centralizada, que exige um esforço de configuração e manutenção não negligenciável. Uma solução para ultrapassar esta limitação passa por executar as aplicações sobre as redes ad-hoc móveis sem fios, i.e., Mobile Ad-hoc Network (MANET) [15, 25].

Uma das vantagens das MANET é a flexibilidade de encaminhamento dos seus protocolos face às alterações frequentes na topologia de rede por causa da mobilidade dos dispositivos, o que as torna atrativa para aplicações móveis distribuídas. Contudo, estas redes levantam novos problemas interessantes para a computação móvel, como a comunicação fiável, a descoberta de dispositivos na rede e a filiação de dispositivos, devido à dificuldade de construir uma visão global da rede [22].

Muitas das tarefas de coordenação entre processos reduzem-se a um problema conhecido de sistemas distribuídos, designado como o *problema do acordo*. O acordo mútuo [5] é exposto da seguinte forma: dado um conjunto de processos onde cada um propõe um valor, todos deverão acordar num desses valores. O problema foi intensivamente estudado em vários modelos de sistemas (e.g., síncronos, assíncronos, com faltas bizantinas, etc [14]).

No entanto, as soluções propostas assumem que todos os participantes se conhecem

entre si, exceto nos trabalhos de Cavin et al. [8, 9], Greve et al. [22] e Alchieri et al. [4]. Os trabalhos de Cavin et al. [8, 9] introduzem pela primeira vez o problema de acordo tolerante a faltas com participantes desconhecidos - Fault-Tolerant Consensus with Unknown Participants (FT-CUP) - para resolver o problema de arranque das redes auto-organizáveis [46]. Greve et al. [22] especificam os requisitos suficientes e uma solução modular para resolver o problema: efetuar a descoberta de participantes na rede, selecionar os participantes que tenham a mesma visão da rede e, em seguida executar um protocolo de acordo clássico [1] nestes participantes. Posteriormente, Alchieri et al. [4] definem o problema do protocolo de acordo tolerante a faltas bizantinas com participantes desconhecidos - Byzantine Fault-Tolerant Consensus with Unknown Participants (BFT-CUP) - e modifica o algoritmo de Greve et al. [22] para suportar este tipo de faltas.

1.1 Motivação

Uma das motivações deste trabalho é desenvolver um conjunto de protocolos de suporte à criação de aplicações distribuídas e cooperativas, que não dependem de infraestruturas fixas mas sim dos próprios dispositivos móveis. Estes protocolos encontram-se organizados numa pilha que fornece serviços de comunicação e filiação de participantes que conseguem executar um protocolo de acordo clássico.

Outra motivação é a avaliação da pilha protocolar em ambientes com e sem mobilidade de participantes e, em diferentes protocolos de encaminhamento de MANET inclusive.

Tanto quanto sabemos, este trabalho é a primeira proposta de uma concretização de uma pilha protocolar concebida para criação de aplicações distribuídas que necessitam de resolver o problema BFT-CUP e que inclui a sua avaliação em ambientes de MANET através do simulador de redes denominado por *ns-3*. Portanto, uma outra motivação, também importante, é adaptar o algoritmo teórico de Alchieri et al. [4] para que seja exequível em redes MANET.

1.2 Contribuições

O trabalho desenvolvido na tese insere-se no âmbito do projeto de SITAN - Serviços Tolerantes a Intrusões para Redes Ad Hoc. Este trabalho envolve a implementação de uma pilha protocolar tolerante a faltas bizantinas, assim como a avaliação da sua eficiência em diferentes ambientes, e.g., participantes com ou sem mobilidade. A pilha é executada sobre uma abstração de canais autenticados mas não fiáveis, baseados nos protocolos padrão da Internet: o User Datagram Protocol (UDP) efetua um melhor esforço para entregar pacotes de dados mas não oferece garantias de recepção; o IP Security Protocol

[1] Um protocolo que assume que todos participantes se conhecem.

(IPSec) protege a integridade e a autenticidade de dados recorrendo a métodos criptográficos (Kent et al. [29]).

A camada mais baixa da pilha tem o nome *Comunicação*, e oferece suporte para a transmissão de dados das camadas superiores, disponibilizando duas primitivas: canais perfeitos e disseminação alcançável bizantina. A camada intermédia, a Vizinhança, descobre os participantes em redor. Por fim, a camada de topo, a Filiação, efetua a descoberta de participantes na rede e determina o grupo de participantes capazes de executar o protocolo de acordo bizantina.

A arquitetura da pilha é semelhante à solução modular de Alchieri et al. [4], cuja implementação é, em muitos dos casos, composta por versões otimizadas dos protocolos originais que melhoram significativamente o desempenho da pilha.

A pilha protocolar contém as seguintes propriedades:

- A implementação da pilha protocolar não assume quaisquer pressupostos temporais. É um facto importante para prevenir ataques que exploram assunções temporais, que é um problema conhecido em alguns protocolos tolerantes a intrusões propostos no passado (Moniz et al. [34]);
- Não necessita de assinaturas digitais, ou seja não usa criptografia assimétrica. A criptografia assimétrica requer poder computacional considerável que se traduz num gasto significativo de energia, sendo uma das preocupações das redes MANET;
- Não obriga ao conhecimento global da rede, que por outras palavras, significa que não requer o conhecimento de todos os participantes *a priori*, que é uma propriedade interessante e adequada às redes MANET.

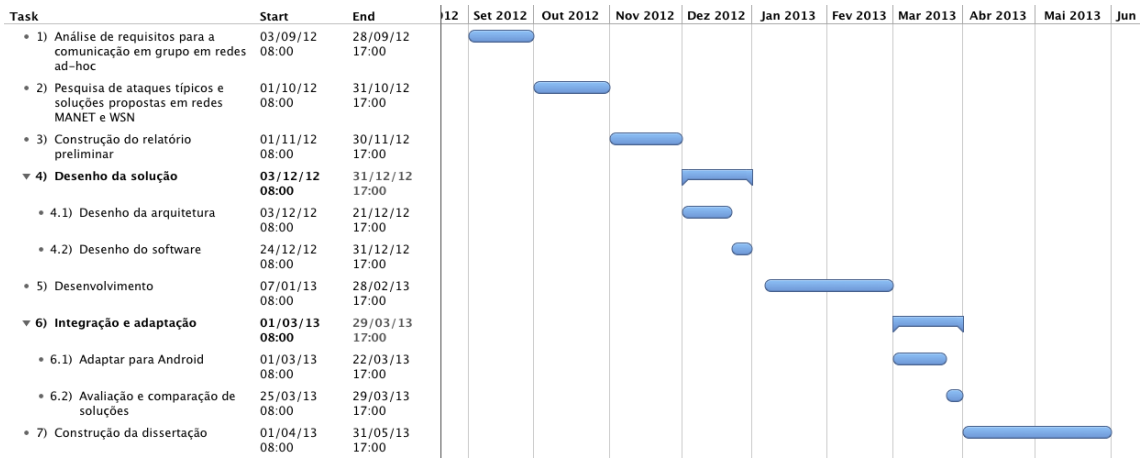
A tese tem duas contribuições principais: descreve a implementação e avaliação da pilha protocolar sobre várias condições de rede (e.g., o aumento de participantes e a introdução de faltas maliciosas e a mobilidade dos participantes). Parte deste trabalho foi publicado recentemente no quinto simpósio do INForum 2013:

- D. Matos, E. Alves, N. Neves, A. Bessani, "MiCCS4Mobile: Middleware para Comunicação e Coordenação Segura em Redes Ad-hoc com Participantes Desconhecidos" em Actas do INForum 2013, Simpósio de Informática, Lisboa, Portugal, Sep. 2013.

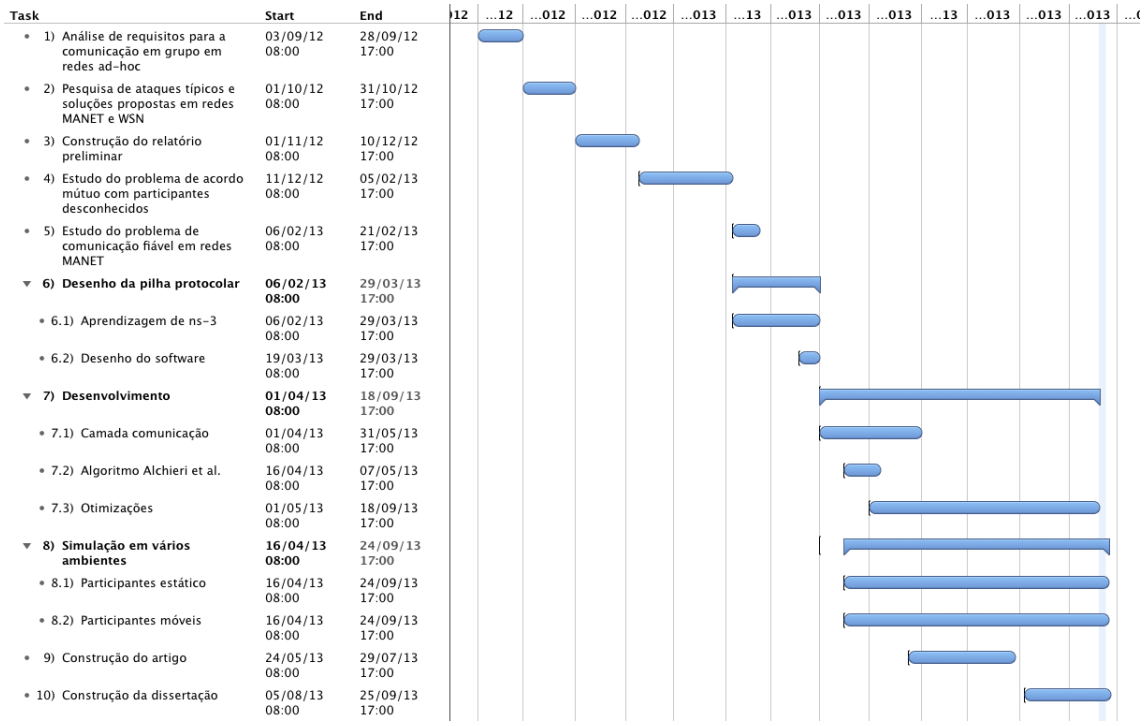
1.3 Planeamento

Os prazos previstos no planeamento do relatório preliminar não foram cumpridos porque houve atrasos na terminação de algumas tarefas propostas, como é possível averiguar comparando os mapas de Gantt na Figura 1.1. Alguns dos atrasos ocorreram devido à

dificuldade imprevista na programação de aplicações e módulos sobre o simulador *ns-3* e dos problemas de comunicação em redes MANET. Outros ocorreram porque surgiu a proposta de otimização dos algoritmos originais de Alchieri et al. [4] e da submissão do artigo "MiCCS4Mobile".



(a) Planeamento previsto no relatório preliminar.



(b) Atual.

Figura 1.1: Ilustração dos mapas de Gantt.

1.4 Estrutura do documento

A tese está organizada da seguinte forma: o Capítulo 2 enquadra e fornece algum contexto ao trabalho desenvolvido, identificando sucintamente algumas caraterísticas fundamentais das redes MANET e apresenta alguns ataques descobertos nestas redes. Fornece também uma explicação do problema acordo em sistemas clássicos e o problema acordo com participantes desconhecidos. O Capítulo 3 descreve a pilha protocolar concebida: a arquitetura, os algoritmos envolvidos e a proposta de modificações aos algoritmos originais. Os detalhes de implementação, problemas encontrados e algumas decisões são apresentadas e debatidas no Capítulo 4. Os resultados obtidos e a avaliação da pilha são descritas no Capítulo 5. Por fim, o Capítulo 6 oferece as conclusões finais da tese.

Capítulo 2

Trabalho relacionado

Este capítulo apresenta o contexto em que a tese está inserida. Como este trabalho propõe uma pilha de protocolos para aplicações de MANET que requerem acordo mútuo, iniciamos com uma breve introdução às redes MANET, que inclui uma discussão de alguns dos protocolos de encaminhamentos e ataques descobertos. Posteriormente é referido um problema clássico de sistemas distribuídos: o acordo. Finalmente, é introduzido o problema do acordo com participantes desconhecidos, i.e., o acordo num ambiente com as características semelhantes observadas às redes MANET, que requer para sua solução, protocolos similares aos proporcionados pela pilha descrita na tese.

2.1 Redes móveis sem fios

Uma *rede ad-hoc* é uma rede local sem fios (normalmente ligada), em que dispositivos móveis ou portáteis pertencem à rede enquanto estiverem próximos. Uma *rede ad-hoc sem fios móvel* (ou MANET) é uma rede autónoma de nós móveis conectados por ligações sem fios, onde a união forma uma rede, um grafo de comunicação. Estes movem-se a qualquer velocidade e em qualquer direção, e são dispostos na rede autonomamente. Consequentemente, a topologia da rede pode sofrer alterações de forma dinâmica e imprevisível. As MANET não têm qualquer infraestrutura fixa e os pacotes de informação são encaminhados por protocolos de multi-salto (i.e., os pacotes são transmitidos do remetente até ao destinatário por nós intermédios, da forma *"store-and-forward"* [24]). Os nós estão a par da dinâmica da topologia da rede para que possam encaminhar os pacotes. Os protocolos de encaminhamento para MANET avaliam e atualizam a informação sobre a topologia de rede. A próxima seção explica o funcionamento destes protocolos de encaminhamento.

2.1.1 Protocolos de encaminhamento

Um protocolo de encaminhamento pode avaliar a rede proativa ou reativamente. Os protocolos proactivos avaliam continuamente a rede, permitindo a atribuição (quase) imediata

de rotas aos pacotes de dados [2]. Por outro lado, os protocolos reativos avaliam a rede apenas quando é necessário, i.e., quando surge um pacote que precisa de ser encaminhado. A vantagem dos protocolos proactivos é a sua rapidez na atribuição de rotas. Em contrapartida, os protocolos proactivos consomem continuamente parte da largura de banda [2].

A taxonomia de protocolos de encaminhamento para MANET é composta pelas seguintes categorias: *table-driven*, *source-initiated on-demand* e híbridos. Um protocolo designado por *table-driven* contém tabelas de encaminhamento para todos os nós da rede [2]. Enquanto um protocolo *source-initiated on-demand* apenas constroí rotas para determinados nós quando requerido [2].

Os protocolos *table-driven* são, na prática, proativos e os protocolos *source-initiated on-demand* são reativos [2]. Os protocolos de encaminhamento híbridos misturam os conceitos anteriores [2]. As próximas secções referem alguns exemplos conhecidos de protocolos *table-driven* e *source-initiated on-demand* relevantes para a tese.

Destination-Sequence Distance Vector Routing Algorithm

O Destination-Sequence Distance Vector Routing Algorithm (DSDV) [41] é um protocolo *table-driven* baseado no algoritmo Bellman-Ford para evitar a construção de rotas cíclicas, onde cada nó da rede cria uma tabela de encaminhamento para os restantes, incluindo informações sobre a distância (em saltos) entre nós e a versão da rota (e.g.um número sequencial). Os nós confirmam periodicamente a presença dos seus vizinhos; caso não haja uma confirmação atempada de algum vizinho, o nó remetente assume que o vizinho está fora do seu alcance e elimina a informação do vizinho contida na tabela. Os nós enviam periodicamente atualizações da topologia de rede. Quando um nó encontra mudanças significativas na topologia de rede, este envia a sua tabela de encaminhamento, ou seja, uma atualização completa; caso contrário, apenas envia pequenas alterações a fazer na tabela de encaminhamento, ou seja, uma atualização incremental. Este conceito permite minimizar o impacto das atualizações em redes estáveis. Quando um nó recebe duas propostas de rota verifica a versão de ambas as rotas. Se as versões das rotas são iguais, é escolhida a rota com menor custo, i.e., com menos saltos de comunicação; caso contrário é escolhida a versão mais recente.

Optimized Link State Routing Protocol

O Optimized Link State Routing Protocol (OLSR) [12] é um protocolo *table-driven* que partilha algumas características do DSDV. Este protocolo seleciona um subconjunto de nós da rede, chamados nós de reencaminhamento, Multipoint relay (MPR), para disseminar mensagens por toda a rede. O propósito é reduzir o número de difusões locais desnecessárias para disseminar mensagens na rede (ilustrado na Figura 2.1). O OLSR tem vários mecanismos de manutenção da rede, que utilizam mensagens de presença e de controlo de topologia. As mensagens de presença permitem obter informação sobre a

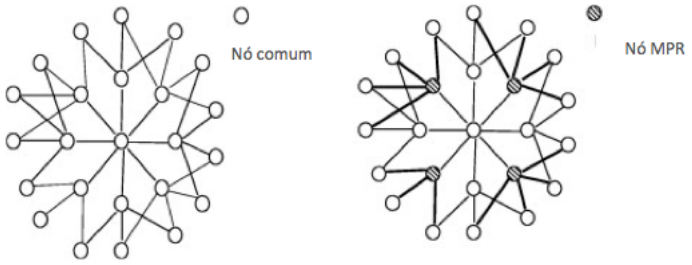


Figura 2.1: Exemplo de execução do protocolo OLSR (adaptado de [43]).

conectividade da vizinhança para criar o conjunto de nós MPR. As mensagens de controlo de topologia propagam as atualizações da topologia de rede e o novo conjunto de MPR.

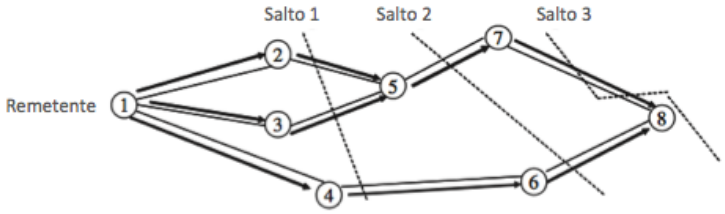
Ad hoc On-demand Distance Vector Protocol

O An On-demand Distance Vector (AODV) [40] é um protocolo *source-initiated on-demand* construído com intuito de melhorar o protocolo DSDV. A topologia de rede é avaliada quando um pacote precisa de ser encaminhado com o procedimento de descoberta de rota (ilustrado na Figura 2.2). A descoberta de rota começa pela disseminação da mensagem de pedido de rota Route Request Packet (RREQ) (ilustrado na Figura 2.2a). A disseminação de RREQ termina se a mensagem chegou a um nó intermédio com uma rota até ao destinatário ou então ao próprio destinatário. Os nós intermédios guardam um par de identificadores: o nó que disseminou a mensagem RREQ no salto de comunicação (nó antecessor) e o identificador do destinatário. Quando a disseminação de RREQ termina, cria-se uma mensagem de resposta Route Reply Packet (RREP). A mensagem RREP é encaminhada para o nó antecessor que disseminou a mensagem RREQ até chegar ao nó que executou o procedimento de descoberta de rota, percorrendo o caminho inverso ao da mensagem RREQ (ilustrado na Figura 2.2b). A disseminação pode originar múltiplas mensagens de resposta. Caso o nó que executou o procedimento de descoberta de rota receba várias respostas, é escolhida a de menor custo, i.e., com menos saltos de comunicação.

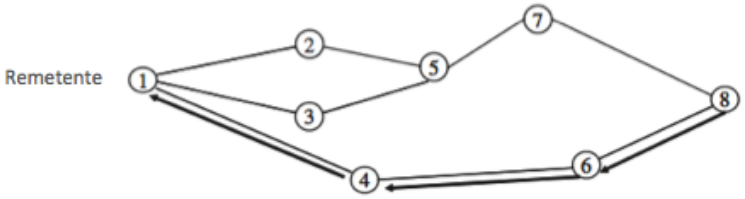
Sempre que um nó intermédio de uma rota fique inacessível, a vizinhança que detetou a ocorrência envia uma mensagem Route Error Message (RERR) para o nó que pediu a rota. Mas quando o nó que pediu a rota se desloca, este reinicia o procedimento de descoberta de rota.

2.1.2 Ataques descobertos aos protocolos de encaminhamento

As redes ad-hoc sem fios são alvos fáceis devido ao meio de transmissão sem fios, para além de que os protocolos de encaminhamento não foram inicialmente concebidos para suportar ação maliciosa. Têm vindo a ser descobertos ataques que manipulam o fun-



(a) Disseminação da mensagem RREQ.



(b) Transmissão da mensagem RREP.

Figura 2.2: Exemplo de execução do protocolo AODV (adaptado de [2]).

cionamento dos protocolos de encaminhamento. Alguns dos ataques descobertos são classificados por ataques ativos ou passivos; os ataques ativos danificam a correção da comunicação entre nós da rede, enquanto os ataques passivos são ataques ocultos para observar e analisar a rede e, por norma, são os primeiros passos para a elaboração de um ataque ativo [11]. De seguida, são apresentados alguns dos ataques ativos relevantes.

Sybil attacks. Este ataque [18] tem o objetivo de personificar ou criar identidades de outros nós da rede. Um *Sybil attack* é avaliado conforme os três parâmetros seguintes: comunicação direta ou indireta, chaves fabricadas ou roubadas e simultaneidade. Num ataque de comunicação direta o nó comunica diretamente com os nós corretos; por outro lado, os ataques de comunicação indireta requerem auxílio de outras entidades maliciosas. As chaves criptográficas são utilizadas em esquemas criptográficos utilizadas, normalmente, para provar a identidade da entidade. Uma chave pode ser roubada de outro participante correto (e.g., obtendo-a diretamente no armazenamento de dados do nó) ou então pode ser fabricada. As chaves podem ser utilizadas em simultâneo para simular todos os participantes ou apenas um a um.

Wormhole attack. Um *wormhole attack* [11] cria túneis de comunicação não visíveis na rede para transmitir pacotes de dados para zonas longínquas da rede, ultrapassando o percurso multi-salto correto que teria de ser feito (ilustrado na Figura 2.3). Um do propósitos do ataque é explorar vulnerabilidades nos protocolos de encaminhamento que utilizam métricas para seleção de rotas, para provocar a negação de serviço. O ataque

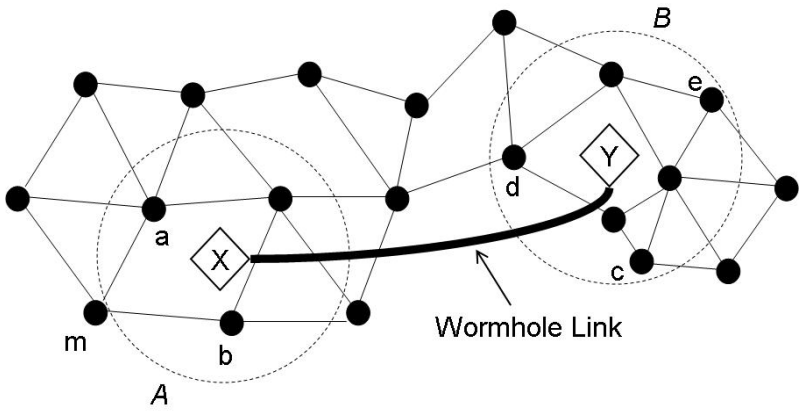


Figura 2.3: Ilustração de um *Wormhole attack* (adaptado de [33]).

poderia ser feito da seguinte forma: enviar um pedido de rota válida (i.e., nó correto) no túnel e difundi-la localmente no final do túnel que situa-se numa região próxima do destinatário. O pedido ultrapassa o conjunto de nós intermédios que são necessários para transmitir uma mensagem de nó em nó em circunstâncias normais. O ataque aumenta a probabilidade de criar uma rota inválida para o destinatário com menor custo do que outras rotas válidas propostas. O ataque é classificado consoante o vetor de ataque e a visibilidade dos participantes maliciosos envolvidos. Observou-se três vetores de ataque[1] possíveis [26]:

- utilizando canais da rede - um nó malicioso atrai as transmissões de pacotes vizinhos e reencaminha-os através do túnel (composto por nó da rede) para outro nó malicioso que transmitirá a mensagem localmente.

- sem canais da rede - semelhante ao procedimento anterior, mas com um túnel construído fora da rede ad-hoc (e.g., uma ligação física entre os nós maliciosos).

- explorando protocolos - tipicamente os nós contêm mecanismos para evitar colisões ou poupança de energia que os obriga a atrasar as transmissões ou até mesmo "adormecer". Os nós maliciosos exploram esta janela de vulnerabilidade para transmitir pacotes diretamente.

Selective forwarding attack. Um *selective forwarding attack* [11] descarta um subconjunto de pacotes transmitidos pela rede (ilustrado na Figura 2.4), interna ou externamente. Internamente, um nó malicioso pode descartar aleatoriamente subconjunto de pacotes que foram encaminhados para este. Por outro lado, um dispositivo externo à rede pode anular os sinais de rádio das transmissões vizinhas, i.e., *jamming attack* (ilustrado na Figura 2.5).

[1] Conjunto de recursos ou procedimentos utilizados para realizar o ataque

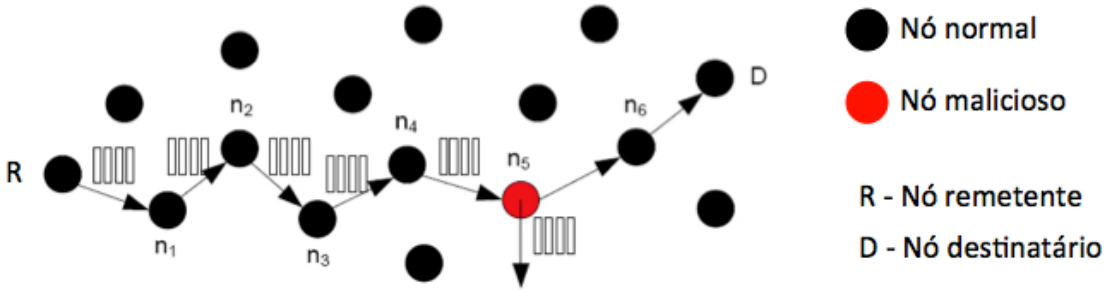


Figura 2.4: Ilustração de um *Selective forwarding attack* (adaptado de [37]).

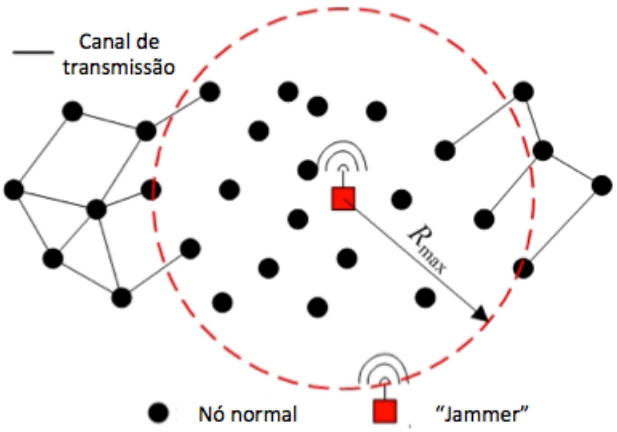


Figura 2.5: Ilustração de um *Jamming attack* (adaptado de [37]).

Sinkhole attack. O objetivo do ataque é atrair as transmissões de pacotes próximas, indicando que é o nó mais indicado para o próximo encaminhamento do pacote através de ataques que exploram a métrica do protocolo de encaminhamento [11]. Um dos ataques que permitem explorar as métricas é um *Wormhole attack* (ilustrado na Figura 2.6).

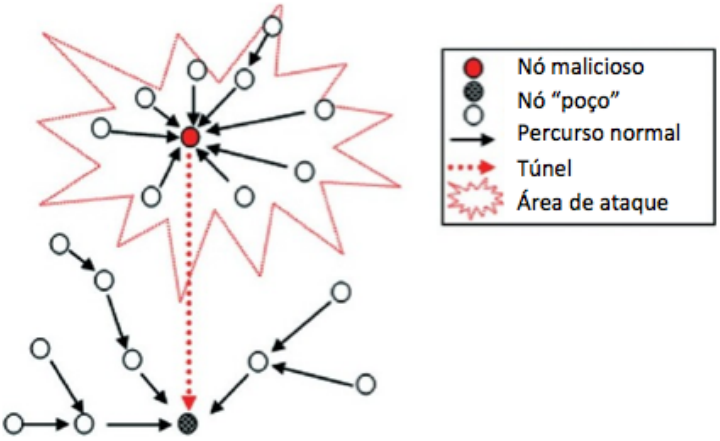


Figura 2.6: Ilustração de um *Sinkhole attack* (adaptado de [21]).

2.2 Acordo mútuo

O problema do acordo mútuo [5] é um problema clássico introduzido pela primeira vez por Pease et al. (1980), com ambos os interesses acadêmicos e práticos (Lynch, 1996; Guerraoui and Schiper, 1997; Guerraoui et al., 2000) [14]. O problema é exposto informalmente da seguinte forma: dado um conjunto de processos onde cada um propõe um valor, todos deverão acordar num desses valores. Outros têm demonstrado que diferentes problemas interessantes de sistemas distribuídos se reduzem ao problema de acordo entre processos. Por exemplo, os seguintes problemas são equivalentes: as máquinas de estado replicadas (em Lamport et al. [30]) e a disseminação atómica (em Correia et al. [13]).

O problema do acordo mútuo pode ser desenvolvido em vários modelos de sistema. Um modelo de sistema é visto como um conjunto de parâmetros que abstraem e ditam o funcionamento do sistema em questão. As duas assunções principais são o sincronismo do sistema e o modelo de faltas. Diz-se que um sistema é síncrono se existe limite máximo para o atraso da comunicação e computação dos participantes do sistema. Por outro lado, um sistema é assíncrono se não existe qualquer limite máximo de atraso.

O modelo de faltas define o tipo de faltas que podem ocorrer no sistema. As classes de faltas normalmente assumidas (ilustrado na Figura 2.7) são: por paragem, omissões e bizantinas (ou arbitrárias). O modelo de faltas por paragem dita que um nó falha parando a sua execução. O modelo de faltas omissas assume que há a possibilidade de ocorrer falhas ocasionais de processos corretos, e.g., uma mensagem que não foi entregue atem-

padamente. Por fim, o modelo de faltas bizantinas [31] considera que podem ocorrer problemas arbitrários durante o funcionamento do sistema, tanto na comunicação como na execução de protocolos, incluindo problemas causados por ações maliciosas.

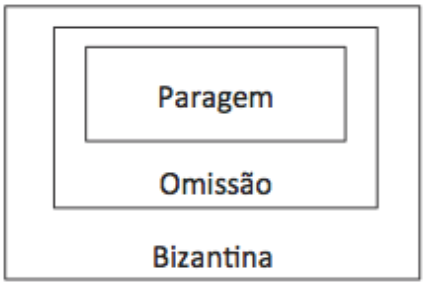


Figura 2.7: Classes de modelos de faltas.

Normalmente, os sistemas são construídos de maneira a continuarem a operar corretamente, considerando que existe um limite máximo f de participantes que possam falhar. O parâmetro é referido como a resiliência do sistema. Por além do parâmetro f, existem outros parâmetros importantes, que definem o poder do adversário sobre o sistema [34] como o escalonamento, os recursos e a sua adaptação. A reordenação da entrega de mensagens é um exemplo de poder de escalonamento do adversário. O poder computacional é um recurso importante a especificar, pois é utilizado para indicar que os adversários não têm poder computacional suficiente para quebrar esquemas criptográficos em tempo útil. Os adversários não podem prever os resultados de protocolos aleatórios mas podem "adaptarem-se"durante os seus passos de execução [34].

As soluções para este problema estão limitadas pelo resultado de impossibilidade, que prova a impossibilidade de resolver, deterministicamente, o problema num sistema assíncrono onde um processo pode falhar, que é frequentemente chamado pelo resultado de Fischer-Lynch-Paterson (Fischer et al. 1985) [20]).

Esta impossibilidade despoletou o interesse em soluções que a contornam. Muitas das soluções requerem assunções temporais explicitamente (e.g. modelos parcialmente síncronos de Dwork et al. [19]) ou implicitamente (e.g. detetores de faltas de Chandra et al. [10] ou "wormholes" de Correia et al. [13]). A única técnica que não requer qualquer assunção temporal é a aleatoriedade - um método probabilista que resolve o problema do acordo (Moniz et al. [34]).

2.3 Acordo mútuo com participantes desconhecidos

A autonomia e a mobilidade dos participantes (ou nós) das redes MANET podem impossibilitar execução de protocolos de acordo referidos anteriormente porque assumem que o conjunto de participantes é estático e conhecido por todos. Desta forma, torna-se inte-

ressante considerar características específicas no modelo do sistema semelhante às redes MANET para resolver o problema do acordo. Existem dois parâmetros para modelos de sistema a considerar como o conhecimento inicial dos participantes e a imprevisibilidade do número total de participantes na rede.

A maioria das soluções existentes para o acordo não suportam estes parâmetros porque assumem que o conjunto de participantes é conhecido por todos. Algumas exceções aparecem nos trabalhos de Cavin et al. [8, 9] e Greve et al. [22]. Estes trabalhos definem os requisitos necessários e suficientes para resolver o problema do acordo em modelos de sistema para redes MANET. Estes autores definem pela primeira vez o problema FT-CUP - o problema do acordo num modelo de sistema com faltas por paragem e participantes desconhecidos, i.e., os participantes apenas conhecem subconjuntos dos outros participantes do sistema. Cavin et al. [8, 9] provam que num cenário com fraca conetividade entre participantes, o requisito suficiente de sincronismo é um detetor de faltas da classe $\mathscr{P}$ [10]. Greve et al. [22] provam que no mesmo cenário, o requisito necessário de sincronismo é um detetor de faltas da classe $\mathscr{S}$ [10]. O trabalho de Greve et al. [22] inclui uma solução modular para resolver o problema: descobrir os participantes na rede que contêm o mesmo conhecimento da rede e executar um protocolo de acordo clássico sobre estes.

Alchieri et al. [4] estende o problema de maneira a que se considerem faltas bizantinas, BFT-CUP. Uma vez que os protocolos descritos por Alchieri et al. serviram de inspiração para o trabalho desta tese, estes serão apresentados brevemente ao longo da próximas secções.

2.3.1 Protocolos de Alchieri et al.

A solução consiste numa pilha de protocolos composta pelos seguintes elementos[2].

Participant Discover. É um oráculo distribuído (entre participantes) que disponibiliza um conhecimento inicial da rede. Nos participantes corretos, este conhecimento está sempre correto e nunca é decrementado.

Reachable Reliable Broadcast. É uma primitiva de disseminação fiável (descrito no Algoritmo 1) que entrega uma mensagem recebida de pelo menos $f + 1$ caminhos disjuntos, assegurando a integridade da disseminação. Assume que um participante conhece $k > f$ participantes para que todos os participantes corretos recebem a mensagem disseminada [4].

Discovery Protocol. O primeiro passo para resolver o acordo em sistemas com participantes desconhecidos é oferecer o máximo conhecimento possível sobre o sistema. A

[2]Os algoritmos dos protocolos encontram-se no Apêndice A

ideia principal do algoritmo (apresentado no Algoritmo 2) é que um participante efetue uma procura em profundidade sobre o conhecimento dado por *Participant Discover*. O algoritmo utiliza a primitiva de disseminação anterior para transmitir o pedido de recolha de conhecimentos iniciais (também denominado por vizinhança de participantes). Um participante é adicionado ao conjunto de participantes conhecidos se estiver presente em mais de f vizinhanças de participantes diferentes, assegurando a correção da sua presença. O algoritmo apenas assegura que todos os participantes que contenham o mesmo conhecimento da rede (considerados como participantes do conjunto poço) terminam a execução do algoritmo, havendo a possibilidade de apenas um subconjunto de participantes não pertencentes ao poço terminarem a execução do algoritmo.

Sink Protocol. Este passo intermédio (apresentado no Algoritmo 3) determina finalmente quais os participantes que pertencem ao conjunto, o poço. O algoritmo questiona a cada participante conhecido se partilham o conhecimento $\mathcal{S}$. Se $|\mathcal{S}| - f$ participantes do conjunto dão respostas positivas, então decide que pertence à filiação porque f poderão nunca responder; caso receba $|\mathcal{S}| + 1$ respostas negativas de participantes do subconjunto, decide que não pertence à filiação, assegurando que pelos um participante correto respondeu.

Consensus. Este último passo (apresentado no Algoritmo 4) consiste em executar um protocolo de acordo bizantino clássico. Um participante que pertence ao conjunto poço, executa o protocolo de acordo e envia para todos os participantes que requeriam o valor da decisão (obtido em *Discovery Protocol*). Enquanto que um participante que não pertence ao poço, aguarda até receber do valor de decisão por mais de f caminhos disjuntos e envia esse mesmo valor para todos os participantes que requisitam o valor da decisão. É assumido que existem pelo menos $2f + 1$ participantes corretos no conjunto poço, sendo o número mínimo necessário para executar protocolo de acordo bizantino como o de Correia et al. [13].

2.4 Sumário

Neste capítulo foram explicados alguns protocolos (proativos e reativos) de encaminhamento para redes MANET e os ataques descobertos. Apresentámos o problema do acordo mútuo e do acordo com participantes desconhecidos. Sucintamente, explicou-se o trabalho de Alchieri et al. [4] que tem contribuição direta para a tese, que foi base para construção da pilha protocolar.

Capítulo 3

Pilha protocolar

Este capítulo descreve a pilha protocolar implementada neste trabalho. O capítulo está divido em três seções: Modelo do sistema, Pilha protocolar e Discussão. A Seção Modelo de sistema apresenta as principais propriedades e características do funcionamento do sistema. A Seção Pilha protocolar descreve os objetivos e os protocolos que compõem a pilha. Finalmente, a Seção Discussão explica a motivação para as modificações efetuadas aos algoritmos originais de Alchieri et al. [4].

3.1 Modelo de sistema

O sistema distribuído é composto por um conjunto finito $\Pi = \{p_1, p_2, p_3, ..., p_n\}$ de n participantes no sistema, em que $n \geq 3f + 1$. O conjunto Π_i corresponde aos participantes conhecidos pelo participante i. Diz-se que um sistema contém apenas participantes conhecidos se $\forall i \in \Pi : \Pi_i = \Pi$; caso contrário, diz-se que contém participantes desconhecidos, $\exists i \in \Pi : \Pi_i \subset \Pi$.

A comunicação entre participantes é realizada através de trocas de mensagens sobre canais autenticados não fiáveis. Os participantes são identificados por números sequenciais únicos em Π (e.g., endereço IP) e não podem fabricar ou roubar identificadores para personificar outros. Por outras palavras, os participantes não podem lançar *Sybil attacks*. Supõe-se a possibilidade de substanciar esta assunção com um protocolo de encaminhamento bizantinos multi-salto e multi-caminho (e.g. Yu et al. [48]).

Os participantes que seguem o algoritmo são chamados participantes corretos; caso contrário são chamados participantes bizantinos [31]. Apenas pode existir um subconjunto $F \subset \Pi$, um número $|F| \leq f$ de participantes que podem falhar durante a execução de protocolos. Os participantes possuem padrões de mobilidade aleatórios que permitem o protocolo de encaminhamento progredir. Devido à dificuldade de distinção de faltas provocadas por falhas de comunicação ou de participantes bizantinos, ambos são considerados como faltas bizantinas. Assume-se que todos os participantes têm conectividade bidirecional com k participantes, em que $k \geq f$.

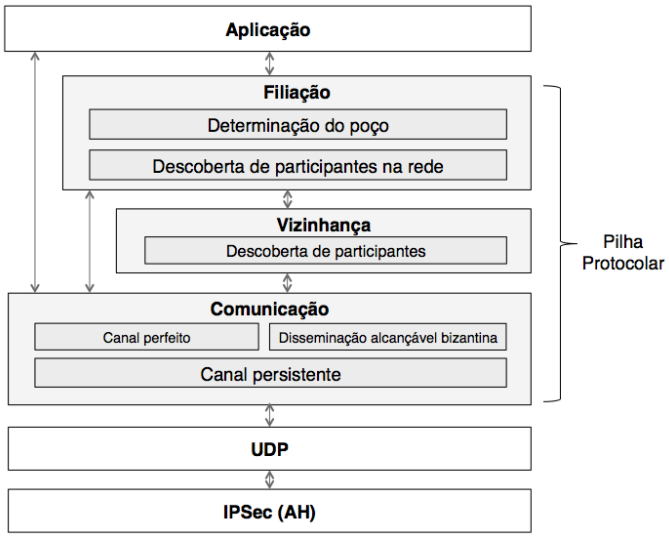


Figura 3.1: Arquitetura da pilha protocolar.

Dois caminhos de mensagem m para o mesmo receptor são disjuntos se não existir um identificador em comum (exceto o identificador do remetente), i.e., $m_1.path \cap m_2.path = \{i\}$, sendo i o remetente de m_1 e de m_2 com o mesmo conteúdo de m.

Um participante pode apenas comunicar com os participantes conhecidos (pertencentes a Π_i). O conhecimento da rede deve ser expandindo da seguinte forma. Um participante i comunica com j se o conhecer, i.e., $j \in \Pi_i$. Se o participante j recebe uma mensagem de i e $i \notin \Pi_j$, então i é adicionado ao conjunto Π_j.

Não existe qualquer assunção sobre a velocidade de processamento dos participantes ou sobre o atraso das mensagens no sistema, ou seja assumimos um modelo assíncrono.

3.2 Pilha protocolar

A pilha protocolar, ilustrada na Figura 3.1, oferece um conjunto de serviços úteis de comunicação e filiação para aplicações distribuídas e coordenadas de redes MANET. A solução modular foi apresentada pela primeira vez por Greve et al. [22], e depois estendida para o modelo de faltas bizantinas por Alchieri et al. [4]. A pilha está sobreposta em dois protocolos padrão da Internet: UDP e IPSec Authentication Header protocol (AH). Estes dois protocolos criam canais de comunicação autenticados não fiáveis para os restantes protocolos da pilha, que são representados pela abstração *Canal não fiável*. A camada de Comunicação fornece os seguintes serviços: *Canal persistente*, *Canal de comunicação* e *Difusão alcançável bizantina*. A camada de Vizinhança permite encontrar participantes dentro do alcance de comunicação com o serviço *Descoberta de participantes*. Finalmente, a camada de Filiação fornece os serviços de *Descoberta de participantes na rede* e *Determinação do poço*. Cada um destes serviços é descrito em mais detalhe nas próximas subseções (exceto os algoritmos que são apresentados no Apêndice B).

3.2.1 Interação entre os serviços da pilha e a rede

A interação entre serviços da pilha é especificada com a troca de mensagens de forma assíncrona. Os serviços são inicializados quando instanciados e depois troca mensagens em cascata (como na Figura 3.2). Os participantes são representados como um conjunto de processos que comunicam num meio de comunicação partilhado (ilustrado na Figura 3.3). As mensagens que passam por este meio de comunicação poderão ser duplicadas, reordenadas ou simplesmente perderem-se por qualquer motivo.

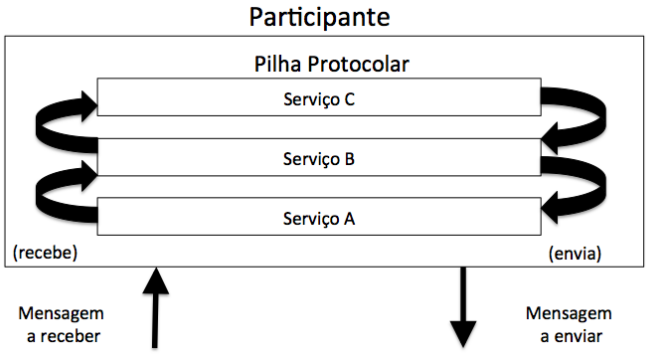


Figura 3.2: Interação entre serviços e a rede.

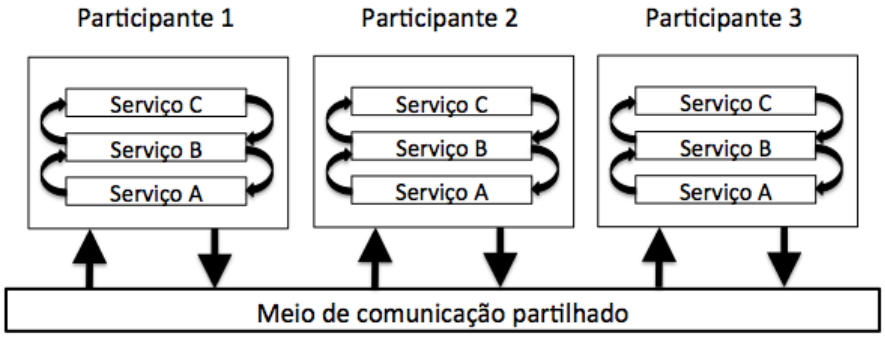


Figura 3.3: Abstração do meio de comunicação partilhado.

3.2.2 Canal não fiável

Os protocolos padrão da Internet de comunicação entre aplicações são: UDP e Transmission Control Protocol (TCP). Ambos os protocolos não podem ser considerados canais fiáveis pelos seguintes motivos. O protocolo UDP efetua um esforço para entregar pacotes de informação, sem garantias de recepção. Por outro lado, o protocolo TCP não faz retransmissão automática se conexão falhar. Estes factos levou-nos a criar uma abstração de comunicação que garante o necessário para execução dos protocolos.

O *Canal não fiável* é a abstração mais fraca de comunicação que interage com o meio de comunicação partilhado. O canal faz um esforço para entregar mensagens ao

destinatário. Contudo as mensagens podem ser reordenadas, duplicadas ou perderem-se na rede. Apenas garante que se for enviado um conjunto infinito de mensagens, então o destinatário entrega um conjunto infinito de mensagens. A assunção que propomos é necessária para resolver problemas em sistemas distribuídos (Guerraoui et al. 1996) [23], porque refere explicitamente que, algures no tempo, as partições de rede são reparadas.

3.2.3 Canal persistente

O *Canal persistente* (descrito no Algoritmo 5) é uma implementação dos *Persistent links* de Cachin et al. [7], apresentada originalmente por Guerraoui et al. [23]. A abstração retransmite continuamente uma mensagem utilizando canais não fiáveis. É necessário enviar continuamente as mensagens porque o canal não fiável não oferece garantias de entrega. A abstração contém um temporizador que inicia o procedimento de retransmissão do conjunto de mensagens a enviar (situado entre as Linhas 5 a 12). O algoritmo contém um conjunto de mensagens entregues, $messages$, e um outro conjunto de mensagens para remover após o primeiro envio, $delivered\_messages$.

3.2.4 Canal perfeito

O *Canal perfeito* (descrito no Algoritmo 6) implementa o mecanismo de confirmação da recepção de mensagens e de não duplicação de entrega de mensagens. O mecanismo de confirmação procede da seguinte forma: **1)** o emissor envia a mensagem para o receptor (na Linha 2); **2)** o receptor envia a confirmação da recepção da mensagem (na Linha 11); **3)** o emissor repete o passo **1)** até receber a confirmação (nas Linhas 4 a 9). A não duplicação de mensagens entregues é garantido verificando se a mensagem recebida pertence ao conjunto de mensagens entregues (descrito nas Linhas 5 e 6).

3.2.5 Disseminação alcançável bizantina

A primitiva de *Reachable Reliable Broadcast* de Alchieri et al. [4] entrega duplicações de mensagens (concluído a partir de [4]) e transmite mensagens desnecessárias para o progresso do algoritmo. Propomos um conjunto de modificações que obrigam que uma mensagem seja entregue apenas uma vez e que descarte as restantes recebidas (entre as Linhas 14 e 25).

A *Disseminação alcançável bizantina* (descrito no Algoritmo 7) é uma versão do algoritmo *Reachable Reliable Broadcast* de Alchieri et al. [4]. Este algoritmo é tolerante a faltas bizantinas e utiliza um mecanismo de flooding. Para reduzir o número de mensagens transmititidas, o algoritmo emprega a difusão local baseado no UDP (na Linha 5). O algoritmo baseia-se no esquema de contadores de Ni et al. [36] para reduzir a redundância e conflito de mensagens. O participante i difunde a mensagem localmente até $|\Pi_i| - f$ participantes refletirem-na (descrito nas Linhas 26 a 30), na presença de um subconjunto

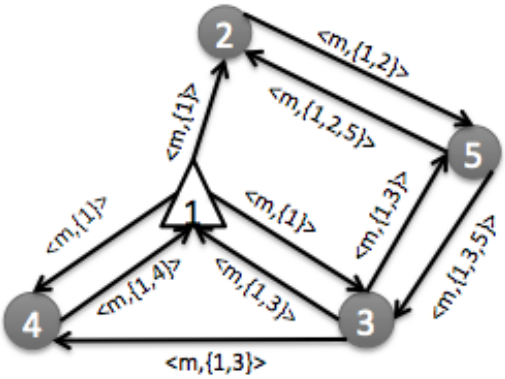


Figura 3.4: Ilustração da execução do algoritmo Disseminação alcançável bizantina.

$F \subset \Pi$, onde $|F| \leq f$ participantes bizantinos podem nunca colaborar (como ilustrado na Figura 3.4).

Na Figura 3.4, o participante 1 inicia o procedimento de disseminação da mensagem m, enquanto os restantes participantes colaboram na disseminação e terminam, apesar do participante 2 malicioso não colaborar.

A integridade da disseminação é garantida porque apenas entrega uma mensagem quando a recebe por mais de f caminhos disjuntos, ou seja, recebeu-a de pelo menos um caminho composto por participantes corretos.

3.2.6 Descoberta de participantes

A *Descoberta de participantes* (descrito no Algoritmo 8) é uma concretização de *Participant Discover* de Alchieri et al. [4]. O algoritmo difunde o identificador do próprio participante localmente em instantes aleatórios (na Linha 2) e ao mesmo tempo escuta anúncios de outros durante um intervalo de tempo definido (nas Linhas 6 e 9). O conjunto obtido é o conhecimento inicial do participante.

3.2.7 Descoberta de participantes na rede

A *Descoberta de participantes na rede* (o Algoritmo 9) expande o conhecimento dado por *Descoberta de participantes* o máximo possível para resolver o problema BFT-CUP. Sucintamente, o algoritmo reúne o conhecimento de todos os participantes *f-alcançáveis* contando com os f participantes bizantinos que podem nunca colaborar (descrito nas Linhas 19 a 33). Um participante é *f-alcançável* de outro se existe mais de f caminhos de comunicação possíveis para o envio de mensagens. Um participante é adicionado ao conjunto de participantes conhecidos apenas, se estiver presente em mais de f conhecimentos obtidos, garantindo que existe pelo menos um participante possa confirmar esta presença (descrito nas Linhas 23 a 26).

O algoritmo *Discovery* de Alchieri et al. [4] assume que o algoritmo de disseminação

entrega a mensagem do próprio emissor, i.e., *loopback*. O problema reside na condição do algoritmo *Reachable Reliable Broadcast* de Alchieri et al. [4] que descarta mensagens que já contêm identificador do participante (na Linha 6) e na primitiva `Send` de *Reachable Reliable Broadcast* de Alchieri et al. [4] adiciona o identificador do participante emissor à mensagem (na Linha 3) . Logo o algoritmo de disseminação nunca efetua o *loopback* esperado. Esta cadeia de acontecimentos prejudicam o *Discovery Protocol* de Alchieri et al. [4] pois pressupõe a sua recepção da sua mensagem para está presente em mais de f conhecimentos obtidos. A solução sugerida passa por atualizar as variáveis para que não seja necessário o *loopback* (descrito nas Linhas 6 a 9 do algoritmo *Disseminação alcançável bizantina*).

3.2.8 Determinação do poço

A *Determinação do poço* (descrito no Algoritmo 10) determina os participantes que pertencem ao conjunto poço, i.e., a filiação. O algoritmo ordena o conjunto de participantes conhecidos e guarda um subconjunto dos primeiros $3f + 1$ participantes do conjunto ordenado. Estas operações reduz o tamanho da filiação para o mínimo de uma filiação para executar o protocolo de acordo bizantino. É válido pois os participantes pertencentes ao poço contêm o mesmo conhecimento da rede, e como todos irão ordenar e retirar da mesma forma, irão obter mesmo conjunto de participantes.

Um participante que não pertença ao subconjunto decide de imediato que não pertence à filiação. Os restantes participantes verificam se todos participantes que conhecem o mesmo subconjunto $\mathcal{S}$. Se $|\mathcal{S}| - f$ participantes do conjunto dão respostas positivas, então decide que pertence à filiação porque f poderão nunca responder (nas Linhas 29 e 30); caso receba $|\mathcal{S}| + 1$ respostas negativas de participantes do subconjunto, decide que não pertence à filiação (nas Linhas 25 e 26), assegurando que pelos menos um participante respondeu corretamente.

3.2.9 Execução do protocolo de acordo

Os passos de execução do protocolo de acordo (descrito no Algoritmo 11) são iguais ao algoritmo de Alchieri et al. [4] (descrito no Apêndice A), exceto que neste algoritmo são utilizadas as primitivas da comunicação da pilha.

3.3 Sumário

Neste capítulo foram explicados protocolos de comunicação utilizados e propostas algumas modificações e otimizações aos algoritmos de Alchieri et al. [4].

Capítulo 4

Concretização da Pilha de Protocolos

A comunicação e filiação em redes ad-hoc móveis com participantes desconhecidos concretiza a pilha protocolar descrita no Capítulo 3. Este capítulo foca-se na descrição dos casos de uso, as classes concretizadas, as sequências de interações entre objetos e também inclui a justificação de decisões tomadas durante o desenho e concretização da pilha.

4.1 Casos de Uso

As motivações da pilha protocolar são o suporte da comunicação e da concordância de filiações em redes MANET para aplicações distribuídas e cooperativas. Existem portanto três casos de uso: criar uma filiação, enviar de uma mensagem para um participante e disseminar uma mensagem na rede. As próximas subseções descrevem cada caso de uso.

4.1.1 Criação de filiação

Atores:

Aplicação cliente;
Protocolo de acordo tolerante a faltas bizantinas.

Pré-condições:

A aplicação cliente num dispositivo é considerada como um participante do sistema.
Existem no máximo f participantes bizantinos.
Um participante tem conectividade com mais de f outros participantes.
Existe um subconjunto de participantes que têm o mesmo conhecimento da rede.
Durante a execução do sistema, apenas podem ocorrer f faltas bizantinas.

Fluxo de Eventos:

O caso de uso começa quando várias da aplicações cliente localizados em diversos dispositivos requerem o acordo de um valor.

1. A aplicação cliente efetua a chamada de criação de filiação ao sistema;
   (a) O sistema inicia o procedimento de descoberta de participantes na rede;
   (b) A pilha protocolar executa o procedimento de determinação do poço na rede, se o procedimento de descoberta de participantes na rede terminar;
   (c) A pilha protocolar retorna a filiação criada e um conjunto de participantes que também requisitaram o valor da decisão do valor;
2. A aplicação cliente verifica se pertence à filiação criada:
   (a) Se aplicação cliente pertencer à filiação criada:
      i. A aplicação cliente executa o protocolo de acordo tolerante a faltas bizantina entre os participantes da filiação;
      ii. O protocolo de acordo tolerante a faltas bizantina retorna o valor decidido;
      iii. A aplicação cliente dissemina o valor decidido ao conjunto de participantes;
      iv. A aplicação cliente recolhe o valor decidido da filiação;
   (b) Se aplicação cliente não pertencer à filiação criada:
      i. A pilha protocolar termina;
      ii. A aplicação cliente recolhe o valor decidido quando a recebe por mais de f caminhos.

Pós-condição:

Se a pilha protocolar propõe uma filiação à aplicação cliente, então é proposta apenas a todas aplicações cliente pertencentes à filiação.

4.1.2 Enviar mensagem para um participante

Ator:

Aplicação cliente;

Pré-condição:

A aplicação cliente é considerada, como participante pela pilha protocolar.
O remetente e o destinatário são participantes corretos.
Existe no máximo f participantes bizantinos.
O emissor e o receptor da mensagem têm canais bidirecionais diretos de comunicação.
Não ocorrem faltas bizantinas que afetam a comunicação ao longo do caminho.

Fluxo de Eventos:

O caso de uso começa quando a aplicação cliente requer o envio de uma mensagem.

1. A aplicação cliente executa a chamada de envio de uma mensagem para o destinatário;

2. Enquanto a pilha protocolar não recebe uma confirmação de recepção da mensagem do destinatário:

   (a) A pilha protocolar executa o procedimento de descoberta da rota do protocolo de encaminhamento;

   (b) A pilha protocolar espera um período de tempo aleatório;

   (c) A pilha protocolar transmite a mensagem para destinatário através do protocolo de encaminhamento;

   (d) A pilha protocolar do lado do destinatário verifica se entregou esta mensagem à aplicação;

   (e) Se pilha protocolar do lado do destinatário ainda não entregou a mensagem, entregue-a;

   (f) A pilha protocolar destinatário envia a confirmação de recepção para o remetente da mensagem.

Pós-condição:

Algures no tempo, o destinatário recebe a mensagem enviada pelo remetente.

4.1.3 Disseminar mensagem a todos os participantes

Ator:

Aplicação cliente;

Pré-condição:

A aplicação cliente é considerada, como participante pela pilha protocolar.

O remetente e alguns receptores são participantes corretos.

Existe no máximo f participantes bizantinos.

O remetente e os receptores da mensagem têm canais bidirecionais diretos de comunicação.

Não ocorrem faltas bizantinas que afetam a comunicação ao longo do caminho.

Fluxo de Eventos:

O caso de uso está divido em duas partes uma vez que implica dois eventos distintos: o início da disseminação da mensagem e a propagação da mensagem. Seguindo a ordenação da ocorrência de eventos, primeiro é apresentado o evento de início da disseminação da mensagem e depois o evento de propagação da mensagem.

Início da disseminação:

1. A aplicação cliente executa a disseminação da mensagem;

2. Enquanto a pilha protocolar não recebe a confirmação de recepção de todos participantes vizinhos (excepto f), a pilha difunde localmente a mensagem.

Propagação da mensagem por vários participantes:

1. A pilha protocolar recebe a mensagem disseminada e verifica se já a recebeu;

2. A pilha protocolar verifica se o caminho percorrido da mensagem é válido;

   (a) Se o caminho da mensagem for válido:

      i. A pilha protocolar guarda a mensagem juntamente com o caminho percorrido;

      ii. Caso tenha recebido a mensagem por pelo menos $f + 1$ caminhos diferentes, a pilha protocolar entrega a mensagem à aplicação cliente;

      iii. Enquanto a pilha protocolar não recebe a confirmação de recepção de todos participantes vizinhos (excepto f), a pilha protocolar difunde localmente a mensagem.

Pós-condição:

Algures no tempo, a pilha protocolar entrega a mensagem a alguns participantes correctos.

4.2 Diagramas de classes

O diagrama de classes está divido em duas partes, onde uma delas contém todas as classes da camada Comunicação e outra que contém as classes das camadas Vizinhança e Filiação. Cada uma delas é ilustrada nas próximas subseções.

4.2.1 Camada Comunicação

A classe mais complexa é a *StubbornSocket* pois reúne: a concretização do canal persistente (descrito no Capítulo 3), o mecanismo de gestão de mensagens a enviar e por fim, a gestão de "callbacks". *ReliableSocket* implementa o algoritmo *Canal perfeito* (do Capítulo 3). As classes *Packet* e *Header* pertencem ao simulador *ns-3* que representam a troca de informação entre participantes. A classe de pacote, *Packet*, é um agregador de cabeçalhos de mensagens, *Header*. Estas classes são abstrações disponibilizadas pelo simulador que representam dados vindos de *"sockets"*.

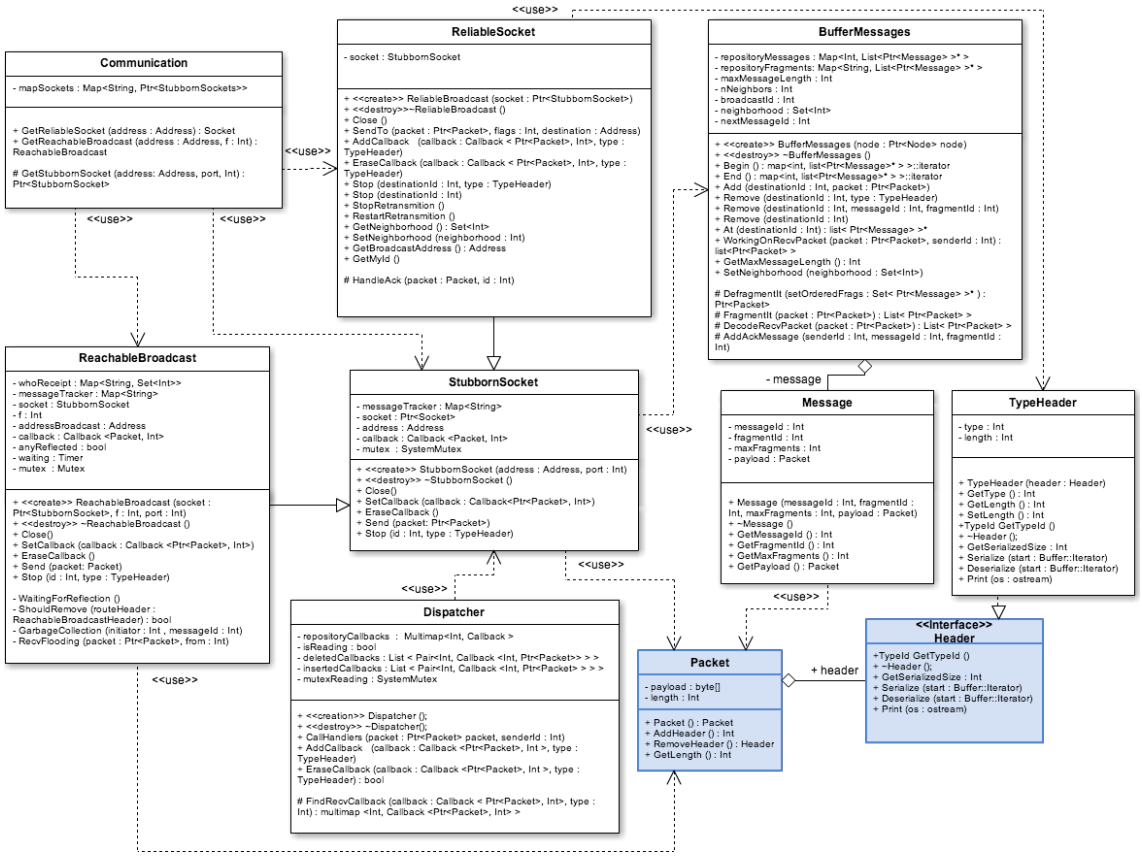


Figura 4.1: Diagrama de classes da camada Comunicação.

4.2.2 Camadas Vizinhança e Filiação

Estas camadas contêm as seguintes classes: *ParticipantDiscover*, *DiscoveryProtocol*, *SinkProtocol* e *GroupMembership*. A *DiscoveryProtocol* implementa o algoritmo de *Descoberta de participantes na rede* e o *SinkProtocol* implementa o algoritmo *Determinação do poço*. A classe *GroupMembership* é apenas uma interface que abstrai a interação entre as classes anteriores.

4.3 Diagramas de sequência

A execução da pilha protocolar envolve uma série de interações complexas e extensas entre entidades que dificultam a construção e a percepção de todas operações envolvidas na construção de uma filiação. Logo os diagramas de sequência foram organizados em quatro eventos: a inicialização da pilha, a execução do protocolo de descoberta de participantes na rede, a execução do protocolo de determinação do poço na rede e por fim a terminação da execução da pilha. As próximas subseções apresentam estes diagramas.

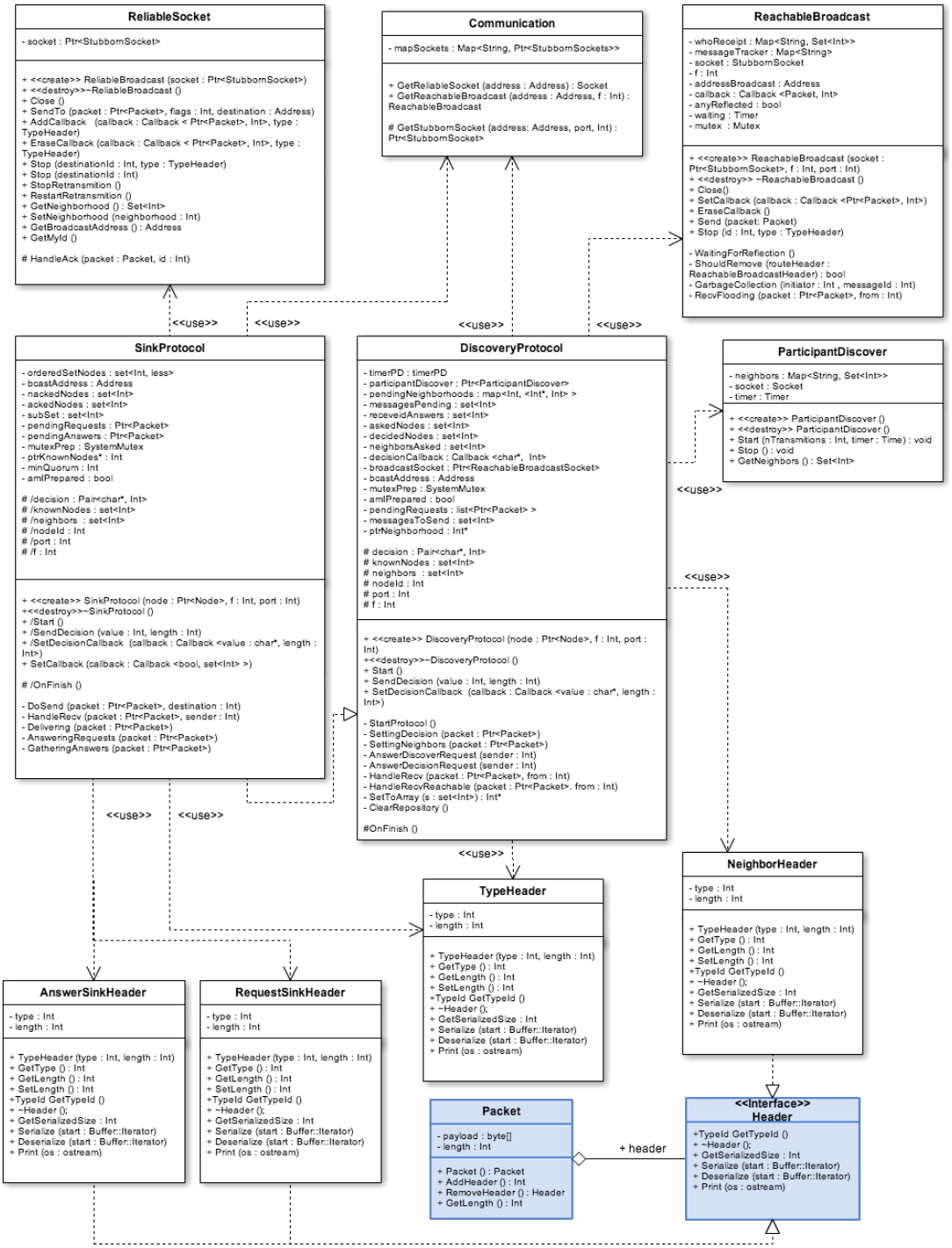


Figura 4.2: Diagrama de classes da camada Vizinhança e Filiação.

4.3.1 Inicialização da pilha protocolar

O diagrama de inicialização da pilha protocolar (ilustrado na Figura 4.3) expõe todas as interações necessárias durante a construção de objetos da pilha. As sequências de interações importantes a reter são: o funcionamento da fábrica de abstrações de canais - *ComunicationFactory* - e o registo de chamadas de conclusão de eventos assíncronos, i.e.,

registo de *Callbacks*.

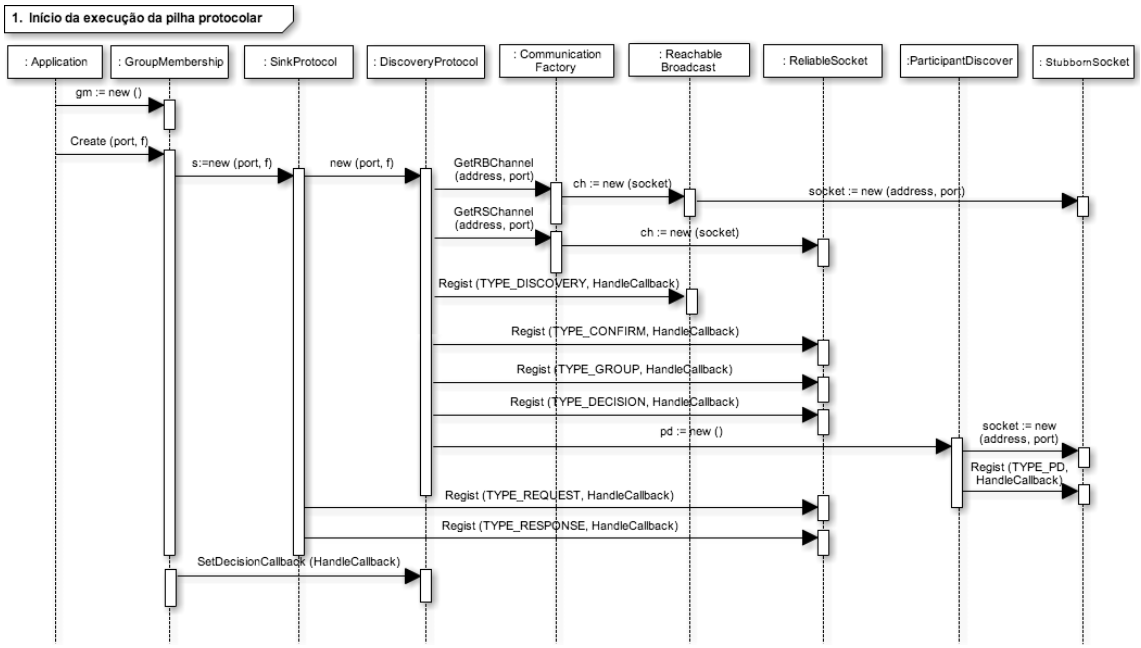


Figura 4.3: Ilustração do diagrama de sequência da inicialização da pilha protocolar.

4.3.2 Execução do protocolo de descoberta de participantes na rede

O diagrama de execução do protocolo de descoberta de participantes na rede (ilustrado na Figura 4.4) exemplifica a sequência de interações realizados para expandir o conhecimento sobre a rede: a obtenção do conhecimento inicial da rede através do descobridor de participantes *ParticipantDiscover*, a expansão do conhecimento inicial e por fim a chamada de execução do protocolo de determinação do poço *SinkProtocol*.

4.3.3 Execução do protocolo de determinação do poço

O diagrama de execução do protocolo de determinação de participantes na rede (ilustrado na Figura 4.5) exemplifica uma vantagem do desenho da arquitetura da pilha protocolar - mitigar a escalagem de eventos entre camadas protocolares. A escalagem de eventos entre camadas provoca uma perda de desempenho numa arquitetura de software desenhada em camadas. O diagrama expõe um exemplo da mitigação da escalagem. O resultado do protocolo de determinação de participantes na rede é enviado diretamente para a aplicação cliente, evitando a passagem do resultado para o objeto interface da pilha protocolar *Groupmembership* desnecessariamente, visto que o resultado do protocolo não necessita de ser processado.

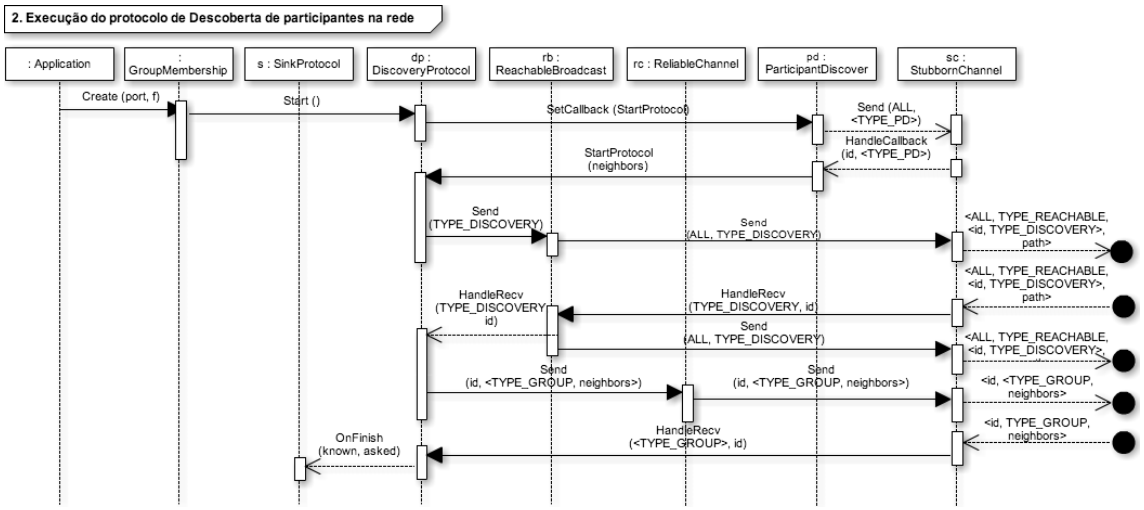


Figura 4.4: Ilustração do diagrama de sequência da execução do protocolo de descoberta de participantes na rede.

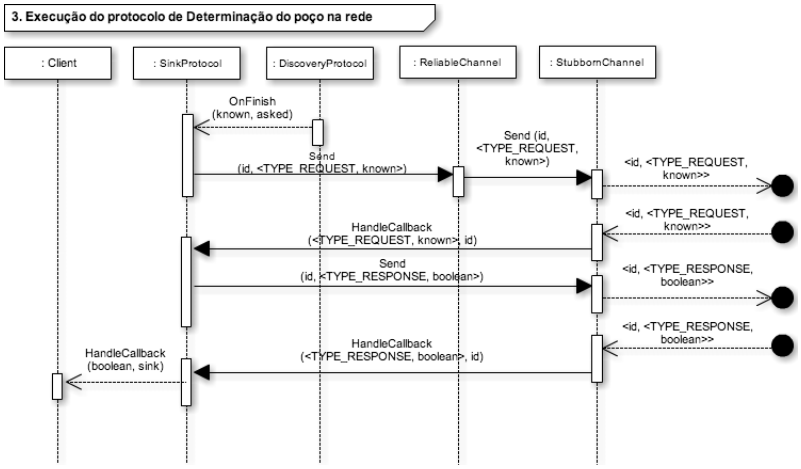


Figura 4.5: Arquitetura da pilha protocolar.

4.3.4 Terminação da execução da pilha protocolar

O diagrama de terminação da execução da pilha protocolar (ilustrado na Figura 4.6) destaca um aspecto essencial da pilha. A execução da pilha deve terminar apenas quando a aplicação cliente obtiver o valor decidido e transmiti-lo para todos os participantes que a requisitarem. Caso esta sequência de interações não seja cumprida, não é possível garantir que todos participantes corretos colaborem no progresso da pilha.

4.4 Concretização da pilha protocolar

A implementação da pilhar protocolar foi executada sobre um simulador de redes chamado *ns-3*, utilizado maioritariamente na simulação de redes ad-hoc sem fios. O *ns-3* [39] é um software *opensource* concebido para fins educacionais e de investigação, reco-

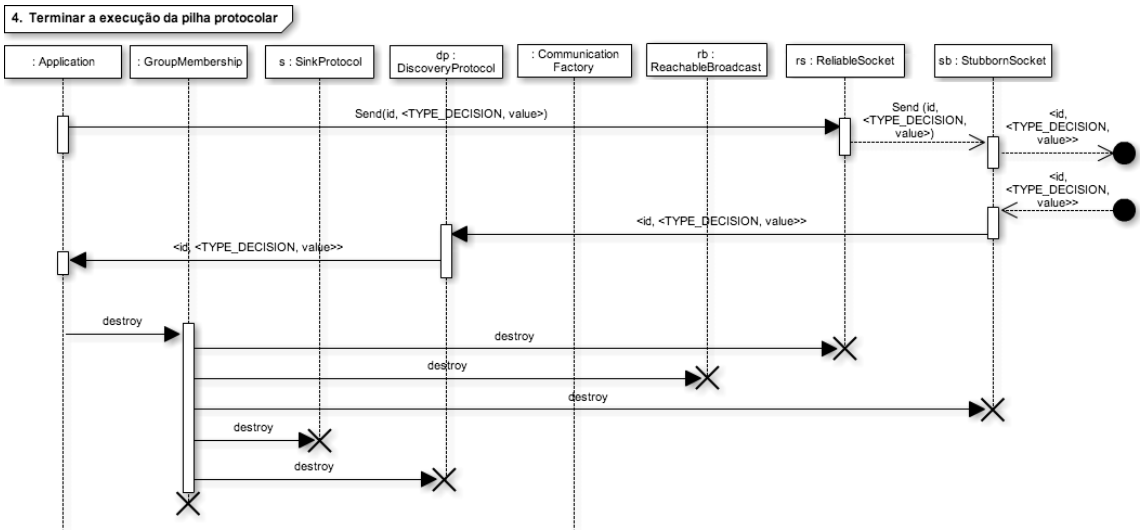


Figura 4.6: Diagrama de sequência da terminação da execução da pilha protocolar.

mendado e aceite pela comunidade científica. O simulador cria um conjunto de objetos que representam nós da rede com a aplicação instalada. Os objetos agregam a configuração desejada como a mobilidade dos nós, a propagação de sinal rádio, a configuração da interface de comunicação ou posicionamento do nó. As aplicações concebidas por vezes utilizam módulos suportados pelo *ns-3* que facilitam a programação de tarefas típicas como a gestão de memória. As simulações deste trabalho utilizam uma aplicação cliente simples que recorre a um módulo que corresponde à implementação da pilhar protocolar (ilustrado na Figura 4.7).

A execução da pilha sobre um simulador permite manipular as variáveis de cenários de teste com maior facilidade do que executá-la numa aplicação móvel dentro de um cenário real, possibilitando por exemplo o uso de um número maior de dispositivos móveis.

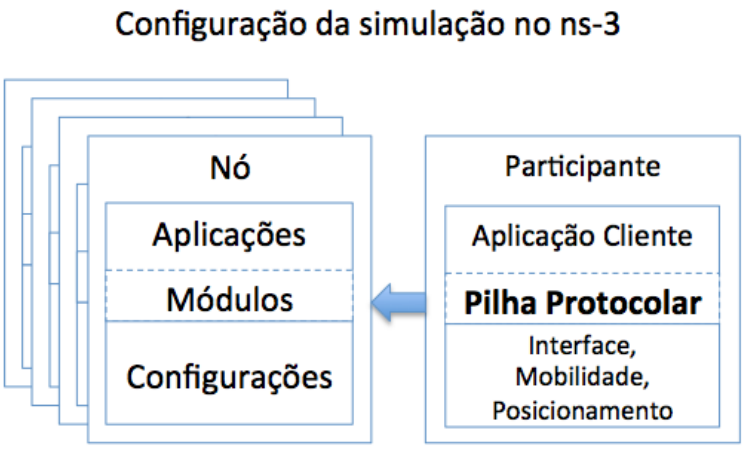


Figura 4.7: Execução da pilha protocolar sobre *ns-3*.

4.4.1 Detalhes na concretização da pilha

Durante a concretização da pilha, surgiram problemas e questões relevantes como a escolha do protocolo de transporte a utilizar. Esta seção tem como objetivo apresentar as soluções e decisões que foram tomadas.

A pilha protocolar como módulo de *ns-3*. O simulador de redes *ns-3* permite estudar os comportamentos das aplicações ou módulos do próprio simulador. A preferência foi o desenvolvimento como módulo do simulador, pois vai de encontro com alguns dos objetivos da tese: construir uma pilha protocolar como suporte para o estudo de novas aplicações em redes MANET. Se a pilha fosse implementada como uma aplicação do simulador causaria provavelmente um esforço não desprezável ao se tentar reutilizar o código fonte, e.g., a separar a lógica, a reestruturar o código ou definir novas "callbacks" internamente.

UDP versus TCP. O protocolo UDP oferece o serviço de melhor esforço na entrega de pacotes de informação, criando uma abstração de canais simples e minimalistas. O protocolo TCP é um protocolo mais robusto do que UDP, que oferece serviços como conexão, transmissão fiável em condições normais, ordenação de entrega de pacotes e gestão de congestão. Apesar da robustez dos canais de TCP, estes não garantem fiabilidade na transmissão, na situação de falha da conexão entre emissor e receptor (não é estabelecida uma nova conexão automaticamente) e não tratam bem da existência de faltas bizantinas.

Kamal et al. [27] mostram resultados de aplicações que utilizam ambos os protocolos de transporte, UDP e TCP, em redes MANET. Estes resultados demonstram uma instabilidade no desempenho do protocolo TCP que, na maioria das situações, não é sentido com o protocolo UDP. Estes autores explicam que o desempenho inferior do TCP em relação ao UDP, deve-se ao facto do TCP acionar os mecanismos de congestão de rede, quando apenas houve uma mudança de topologia de rede que origina a perda de pacotes. Estes mecanismos causam um decréscimo na taxa de transmissão do TCP, reduzindo o seu desempenho.

Estes argumentos motivaram à construção da pilha protocolar sobre os canais de comunicação do protocolo UDP.

Canais de comunicação. A criação de canais de comunicação é uma tarefa complexa que requer a gestão de vários mecanismos em simultâneo: retransmissão (aleatória) de mensagens, confirmação de recepção de mensagens, junção de mensagens para o mesmo destinatário, etc. A pilha protocolar utilizava inicialmente várias instâncias da primitiva *Canal persistente*. Após a adoção da utilização de uma só instância de canal persistente, verificou-se um decréscimo na colisão de transmissões. A conclusão foi que a gestão do mecanismo de retransmissão é problemática devido às seguintes razões: se existirem

várias instâncias do Canal persistente em bastantes participantes, é possível que estas instâncias retransmitam em instantes diferentes, encurtando os períodos de tempo livres para a transmissão de mensagens sem colisões.

A solução mais conveniente para colmatar este problema foi a concepção de uma fábrica de instâncias de canal perfeito e de difusão alcançável bizantina - uma componente lógica arquitetural que garante a utilização da mesma instância de canal persistente. Desta forma o problema da utilização da mesma instância de canal persistente torna-se transparente para os protocolos dependentes da comunicação.

Utilização da difusão local. A *Difusão local* explora o meio de transmissão sem fios para enviar pacotes de informação para participantes, dentro do raio de alcance de transmissão, com apenas uma transmissão. O conceito é atrativo para um algoritmo de disseminação, pois possibilita a propagação de um pacote numa vizinhança de n participantes com apenas uma transmissão, no melhor caso, em vez de n transmissões. A implementação do algoritmo *Difusão alcançável bizantina* faz várias difusões locais para aumentar o desempenho do algoritmo e minimizar o consumo de largura de banda.

Comunicação entre camadas através de "callbacks" A independência entre classes é um problema que dificulta a extensão ou modificação de funcionalidades no sistema. O *ns-3* expõe o problema de uma forma percetível. Neste exemplo, o protocolo TCP tem conhecimento explícito do protocolo Internet Protocol (IP) e assim invoca os seus métodos (e vice versa). A interposição do protocolo IPSec implica quebrar esta dependência entre TCP e IP, i.e., modificar código de TCP e adaptá-lo ao protocolo IPSec. A solução adotada pela simulador *ns-3* passa por utilizar "callbacks" - chamadas de funções executadas quando um evento ocorre. Esta metodologia agilizou as experiências de novas classes e funcionalidades da pilha, como por exemplo, a introdução da classe *ReliableSocket* (ilustrado na Figura 4.8).

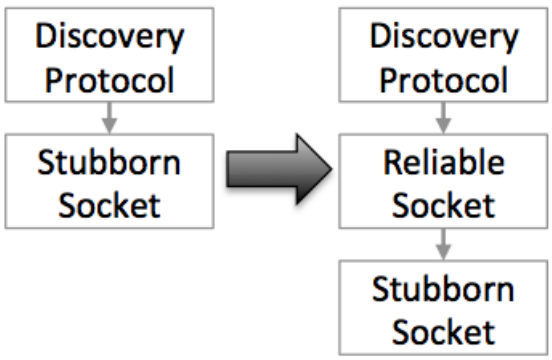


Figura 4.8: Ilustração da introdução da classe *ReliableSocket*.

Mitigação da perda de mensagens. Uma das preocupações durante o desenvolvimento da pilha foi a redução da perda de mensagens dos nós da rede, uma vez que adiam (ou impedem no pior caso) a recepção de mensagens essenciais para a progressão de protocolos colaborativos. A perda de pacotes pode ocorrer pelas seguintes razões fundamentais: a colisão de transmissões ou o consumo exagerado de largura de banda. As próximas subseções referem os mecanismos utilizados na pilha para minimizar o impacto destas duas causas.

Aleatoriedade como arma para a mitigação de colisões de transmissões. A colisão de transmissões prejudica o desempenho dos canais de comunicação porque impede temporariamente a recepção de mensagens. A probabilidade de colisão de transmissões aumenta quanto maior for a densidade e a proximidade de nós das redes MANET, uma vez que os sinais de rádio sobrepõem-se (ilustrado na Figura 4.9). Este problema foi estudado por Ni et al. [36], tendo proposto a espera de períodos de tempo aleatórios para a transmissão de pacotes de informação para evitar colisão de transmissões. Esta técnica é explorada no nosso trabalho no algoritmo do *Canal persistente* (o Algoritmo 5).

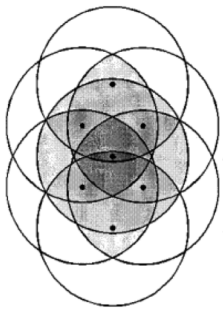


Figura 4.9: Ilustração da sobreposição de sinais de rádio de nós da rede (adaptado de [36]).

Ni et al. [36] também apresentaram várias soluções de disseminação de mensagens, sendo uma delas baseada em contadores, designado por *counter-based broadcast scheme*. Sucintamente, a solução consiste em retransmitir em tempos aleatórios uma mensagem m até receber k retransmissões de m por outros participantes, com intuito de reduzir a redundância de mensagens e diferenciar os tempos de retransmissão de mensagens. O algoritmo de *Disseminação alcançável bizantina* (o Algoritmo 9) inspira-se neste mecanismo, sendo $k = |Pi\_i| - f$ em que Pi_i a vizinhança de um participante i, seguindo as recomendações de Ni et al. [36] para mitigar o problema *broadcast storms*. O algoritmo espera apenas por $|Pi\_i| - f$ respostas, pois existe sempre a possibilidade de f participantes nunca retransmitirem a mensagem m para impedir a terminação do algoritmo.

Limitação do tamanho das mensagens. O IP contém mecanismos de fragmentação e desfragmentação para enviar pacotes na rede. Quando o IP recebe um pacote de tamanho

superior ao aceite pela rede, é obrigado a fragmentá-lo em pacotes menores. O mecanismo de desfragmentação aguarda pela recepção de todos fragmentos de um pacote durante um período de tempo. Quando um dos fragmentos não é entregue atempadamente, os restantes são descartados, i.e., o datagrama não é entregue à camada de transporte. Este evento acontecia frequentemente nas versões iniciais da pilha.

Surgiu a ideia de estabelecer o limite de tamanho das mensagens enviadas pelo canal persistente igual ao limite permitido pela rede, o que reduziu substancialmente o número de mensagens descartadas, resultando num aumento do desempenho da pilha protocolar.

Junção de mensagens. Ao longo da execução dos protocolos da pilha é necessário transmitir múltiplas mensagens relativamente curtas para o mesmo destinatário. Trata-se de um facto importante porque numa implementação ingénua do algoritmo *Canal persistente* (o Algoritmo 5) é desperdiçado tempo e poder computacional, uma vez que estas mensagens são transmitidas individualmente. A pilha protocolar minimiza o problema contendo uma entidade responsável por concatenar mensagens para o mesmo destinatário, até ao limite estabelecido pela rede (referido na subseção anterior).

Mensagens fora de contexto. A assincronia da pilha protocolar origina normalmente a situação em que um participante recebe mensagens corretas para as quais não está à espera. Isto sucede por exemplo quando um participante inicia um acordo, enquanto os restantes iniciam-no mais tarde. Moniz et al. [34] designa estas mensagens de "mensagens fora do contexto" porque ainda não existe contexto para processar tais mensagens. Estas mensagens não são descartadas na pilha protocolar, mas sim guardadas para posteriormente serem processadas pelos respetivos protocolos.

A escolha do tamanho dos identificadores de participantes. As primeiras versões da pilha protocolar usavam o endereço IPv4 das máquinas simuladas para identificar os participantes. Surgiu a ideia de utilizar apenas um subconjunto (os dezasseis "bits" menos significativos) do endereço IPv4 para analisar o impacto do tamanho do identificador no desempenho da pilha. Verificou-se que a redução do tamanho do identificador aumentou drasticamente a escabilidade do algoritmo. Um exemplo representativo deste facto é que numa simulação de uma rede densa de vinte e nove nós, os participantes não terminavam após um minuto de execução da simulação. Após a redução do tamanho de identificador, na mesma simulação, a execução da pilha termina a sua execução em menos de vinte segundos. Isto deve-se à redução de colisões de transmissão, como pela redução do tamanho médio das mensagens e pela redução do número de transmissões necessárias.

Optar por não disseminar as vizinhanças de participantes. Uma possível otimização para reduzir o número de mensagens necessárias a transmitir seria disseminar o conheci-

mento inicial da rede de imediato, substituindo a troca de mensagens de pedido e envio de vizinhanças, no início da execução de *Descoberta de participantes da rede* (o Algoritmo 9). Após a sua implementação, verificou-se uma perda de pacotes que consecutivamente originou uma degradação de desempenho e na escabilidade dos protocolos. O problema residia no protocolo de *Disseminação alcançável bizantina* (o Algoritmo 7) que agrega o identificador do participante ao campo “caminho"das suas mensagens, um campo cujo o comprimento pode chegar ao número de nós da rede vezes o tamanho de um identificador. Em redes densas com participantes próximos, o conhecimento inicial da maioria dos participantes abrange toda a rede, i.e., o comprimento pode chegar ao número de nós da rede vezes o tamanho de um identificador inclusive. Logo o tempo de transmissão destas mensagens era maior do que a versão da pilha sem esta otimização provocando conflito com outras transmissões de outras mensagens. Assim, acabamos por não utilizar esta otimização.

4.5 Sumário

Neste capítulo foram descritos os raciocínios que levaram ao desenho da arquitetura da pilha. Também foram explicadas as modificações relevantes que tornaram o algoritmo de Alchieri et al. [4] prático em cenários de redes MANET testados e avaliados no próximo capítulo.

Capítulo 5

Resultados

Este capítulo apresenta os resultados e a avaliação realizada à concretização da pilha em vários cenários de redes móveis sem fios simuladas pelo *ns-3*.

5.1 Configurações das simulações

Os nós foram configurados com o conjunto de parâmetros (especificado na Tabela 5.1) em todas as experiências, como por exemplo: o protocolo da camada de rede, o limiar do mecanismo de mitigação de colisões designado por RTS/CTS e os modelos de propagação do sinal da interface de rede.

Cenários de avaliação. As experiências foram realizadas em dois cenários especificados: "Grelha" e "Aleatório". Num cenário designado por "Grelha", os participantes são estáticos e dispostos em grelha numa área quadrada. Enquanto que num cenário"Aleatório", os participantes possuem mobilidade reduzida, e estão dispostos aleatoriamente num espaço físico limitado sem obstáculos. As caraterísticas destes cenários estão especificadas na Tabela 5.2.

Recolha de amostras. A análise baseia-se em amostras recolhidas pelo simulador nos primeiros trinta segundos após a execução da pilha. A escolha do tempo de amostra surge após o balanço feito entre a quantidade retirada de informação das experiências e o tempo que o simulador demora a computar as experiências mais interessantes de estudar.

Nas experiências variou-se o número de participantes n entre quatro e trinta. Também se testaram diferentes valores de f, entre um e nove. Uma vez que é preciso garantir que foram estudados um número total de participantes é superior ou igual a $n \geq 3f + 1$, em algumas combinações de n e f não foi possível recolher resultados (e.g., $n = 6$ e $f = 3$).

Um participante bizantino prejudica o progresso da pilha protocolar se não colaborar com outros participantes. Um vetor de ataque possível passa por anunciar sua presença e posteriormente não colaborar no resto da computação. No entanto, como os protocolos

Parâmetro	*Valor*
Versão do simulador	3.18
Unicast	Desativado
Limiar do RTS/CTS	2200 (bits)
Correção dos pacotes	Ativado
Protocolo da camada de rede	802.11g
Largura de banda	11 (Mbps)
Alcance máximo do sinal	38 (m)
Modelo de perda de sinal rádio	RangePropagationLossModel
Modelo de velocidade de sinal rádio	ConstantSpeedPropagationDelayModel
Quality-of-service	Desativado

Tabela 5.1: Configuração padrão utilizado em todas as experiências.

Cenário	Disposição		Mobilidade		
	Horizontal (m)	Vertical (m)	Velocidade (m/s)	Área (m^2)	Pausa (s)
Grelha	3	5			
Aleatório			$0.7 - 0.9$	70×70	2

Tabela 5.2: Configuração dos cenários Grelha e Aleatório.

foram concebidos contando com a possibilidade de f participantes não colaboram, estes mesmo assim deverão terminar.

Foram descartadas as experiências que não contêm a confirmação dos $3f + 1$ participantes pertencentes à filiação porque não são relevantes para os estudos e análises discutidas posteriormente (como o posicionamento inicial dos participantes é aleatório, sucedem em alguns casos a existência de participantes completamente isolados). Foram calculados a média de tempos de execução e o desvio padrão, a taxa de transmissão e de recepção de pacotes, a média da latência das transmissões, e por fim, a média da percentagem de perda de pacotes em todos os participantes.

5.2 Avaliação

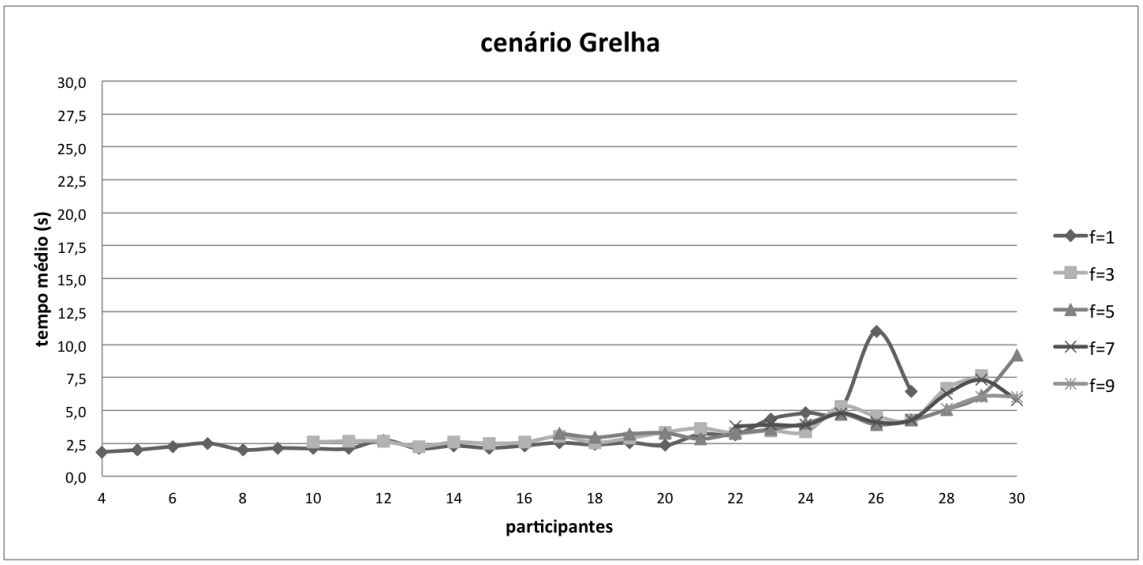
Neste seção são apresentados alguns resultados da pilha sobre diferentes cenários e condições da rede.

Algoritmos originais de Alchieri et al. versus algoritmos modificados. No Capítulo 3 foram propostos um conjunto de modificações e otimizações ao algoritmo de Alchieri et al. [4], sendo relevante efetuar uma análise comparativa entre os algoritmos originais e os modificados. A análise baseou-se na execução da pilha sobre os protocolos de encaminhamento, e na variação das faltas bizantinas e do conjunto de participantes.

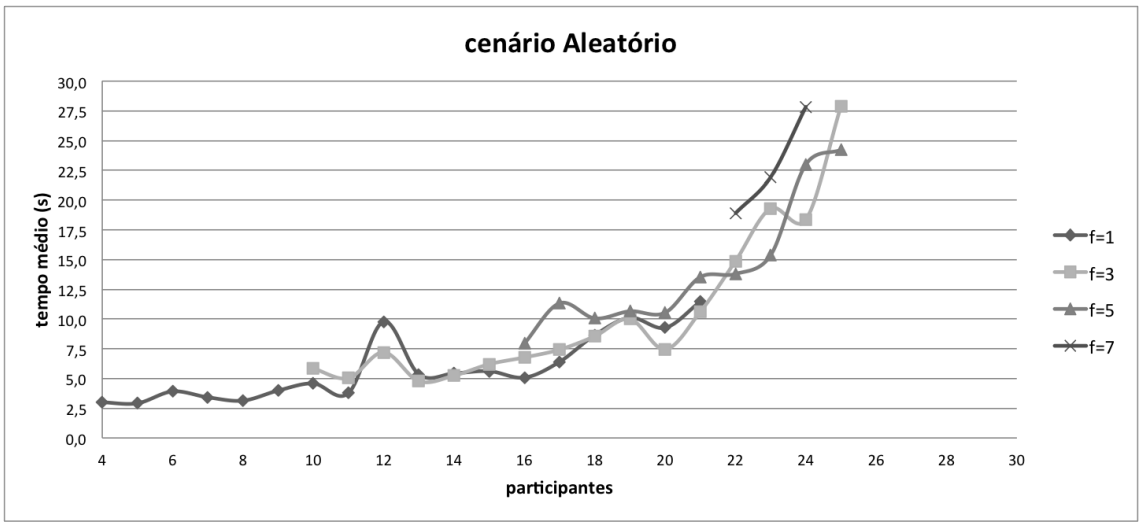
A pilha com os algoritmos originais produziu resultados apenas numa experiência com quatro participantes, uma falta bizantina e executada sobre DSDV. As restantes experiências sofreram de problemas de escalabilidade, o que motivou as nossas otimizações.

Aumento de participantes e de faltas bizantinas. As experiências variaram o número de participantes da rede e o número de faltas bizantinas assumidas, exceto a utilização do protocolo de encaminhamento, DSDV. Na Figura 5.1 observa-se num cenário Grelha (sem mobilidade) o tempo médio do retorno da filiação onde observou-se um crescimento linear dos tempos quando conjunto de participantes aumentou. Num cenário Aleatório (com mobilidade), o tempo médio do retorno da filiação tende a crescer exponencialmente até à experiência com vinte e cinco participantes.

Avaliação de protocolos de encaminhamento. No Capítulo 2 referiu-se duas grandes famílias de protocolos de encaminhamento para MANET: *table-driven* e *source-initiated on-demand*. *Table-driven* é uma família de protocolos proativos que avaliam continuamente a rede para obter uma tabela de rotas; enquanto *Source-initiated on-demand* é uma família de protocolos reativos que efetuam a descoberta de rotas apenas quando encaminham pacotes de informação. Esta subseção compara o impacto de diferentes protocolos de encaminhamento *multi-hop* na camada de suporte, sendo eles: o AODV e o DSDV.



(a) Resultados das experiências em cenários Grelha



(b) Resultados das experiências em cenários Aleatório

Figura 5.1: Tempo médio da criação da filiação face ao aumento de participantes e de faltas bizantinas.

Foram selecionados as experiências com $f = 3$ num cenário Aleatório, para estender a análise para o máximo de número de participantes móveis e de faltas bizantinas que possam ser computadas em tempo aceitável para concretização da tese.

O desvio padrão é uma métrica a não desprezar porque fornece alguma indicação da janela temporal em que os participantes poderão gerar mensagens fora de contexto (referido no Capítulo 4), implicando um consumo de largura de banda. Portanto quanto menor for desvio padrão do tempo de execução, menor será o consumo de largura de banda desnecessário (se executar o protocolo acordo que não guarde mensagens fora de contexto).

Na Figura 5.2 observa-se um comportamento irregular no desempenho da pilha quando

executada sobre o AODV. Contudo, a média e o desvio de padrão tendem a ser inferiores ao do DSDV em experiências com mais de dezoito participantes. Apesar da mobilidade reduzida, ocorrem mudanças na topologia da rede que aumentam o número de atualizações que o DSDV tem de transmitir e aplicar nas suas tabelas de encaminhamento, o que consequentemente, atrasam o processo de encaminhamento dos pacotes de dados.

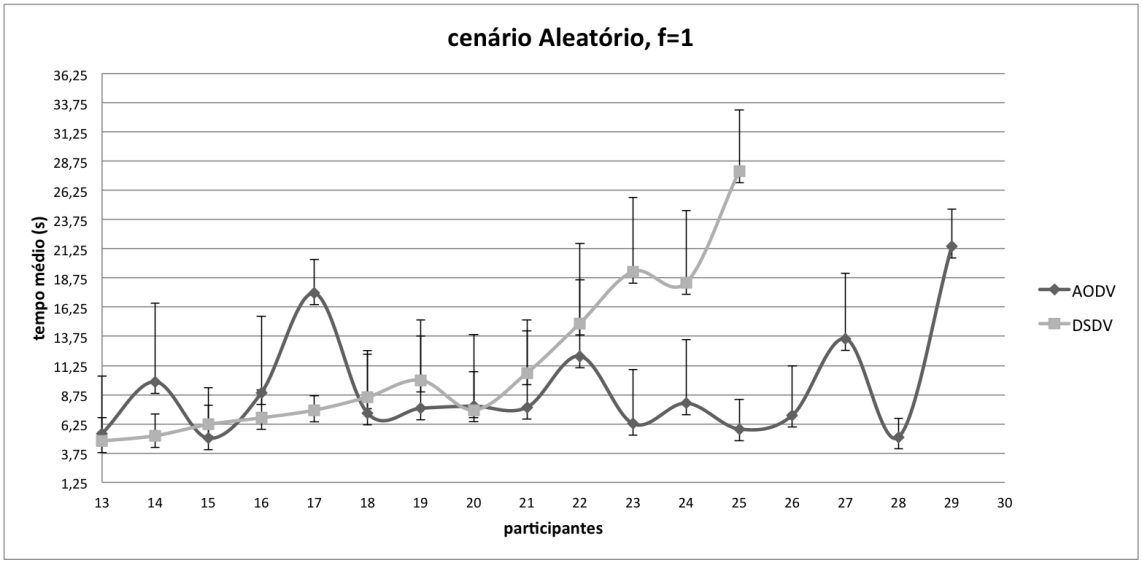


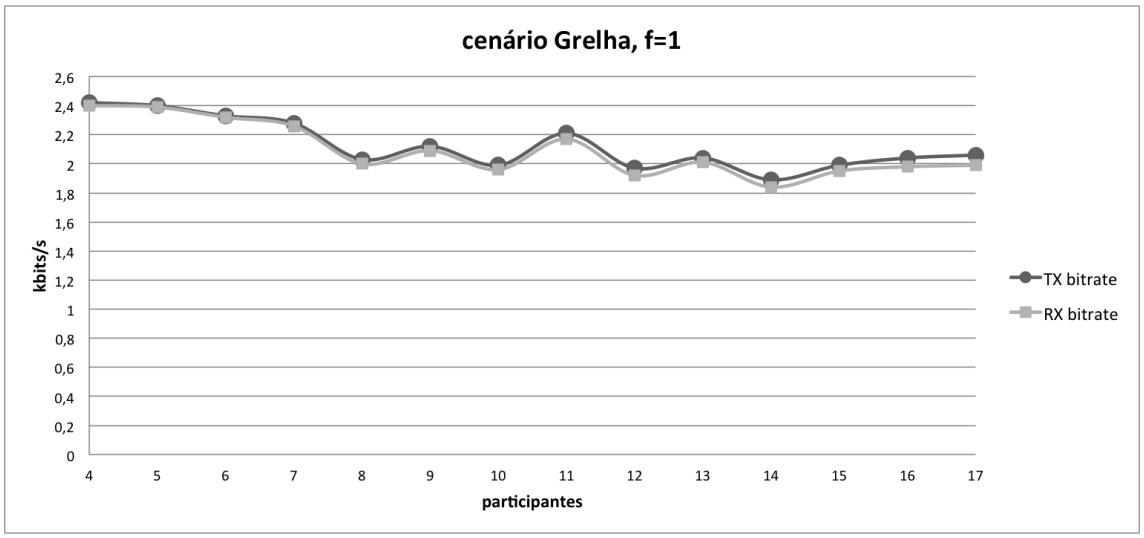
Figura 5.2: Tempo médio da criação da filiação num cenário Aleatório e $f = 1$, utilizando AODV e DSDV.

Impacto na rede. Esta análise pretende responder a questões sobre a sobrecarga e os padrões de tráfego da pilha na rede. O conjunto das experiências têm o valor de $f = 1$ fixado e utilizam o DSDV.

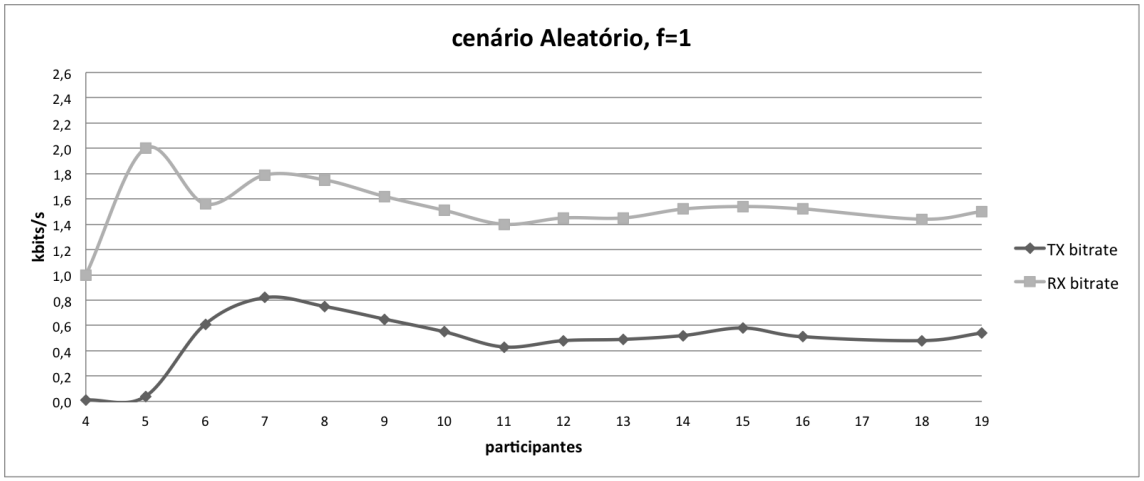
As figuras 5.3a e 5.3b mostram que o consumo de largura de banda (a taxa de transmissão somada da taxa de recepção) da pilha em cada participante não é superior a 5kbps, que é aproximadamente 0.00045% dos 11mbps da largura de banda, independemente da mobilidade e disposição dos participantes.

Num cenário Grelha, estas taxas calculadas sobrepõem-se e descem conforme o aumento dos conjuntos dos participantes. Enquanto que num cenário Aleatório verifica-se um desfasamento entre as taxas, onde a taxa de recepção de pacotes é superior ao de transmissão. Este é um efeito interessante provocado por um conjunto de variáveis da rede. Os algoritmos *Disseminação alcançável bizantina* e *Determinação do poço* fazem difusões locais. Através de uma difusão local é possível entregar um pacote de dados a vários participantes numa só transmissão, reduzindo as transmissões necessárias para o progresso dos serviços da pilha. Num cenário Aleatório, os participantes tendem a ficar mais próximos devido à mobilidade e à disposição inicial, havendo mais participantes possivelmente à escuta destas difusões locais. O padrão 802.11 emite uma confirmação, i.e.,

acknowlegment data (ACK), por cada trama entregue. Após uma análise das experiências com aplicação, NetAnim [38], verificamos que um participante que difundiu localmente uma mensagem, recebe dos outros várias mensagens ACK de protocolo 802.11g. Por isso, existe aproximadamente uma taxa de recepção do dobro da taxa de transmissão de pacotes quando os participantes têm mobilidade.



(a) Num cenário Grelha.



(b) Num cenário Aleatório.

Figura 5.3: Resultados das taxas de transmissão e recepção médios de cada participante.

A Figura 5.4 mostra a média da latência de comunicação da camada Suporte entre participantes. Esta cresce linearmente face ao aumento de participantes na rede. Yidong et al. [47] provaram matematicamente que o atraso de comunicação tem uma relação linear com o número de participantes, a mobilidade e a disposição destes na rede. As experiências de Yidong et al. [47] são semelhantes às realizadas, excepto a dimensões das áreas testadas (que atingem os mil metros).

Por fim, na Figura 5.5 observou-se uma oscilação irregular de perda de pacotes num

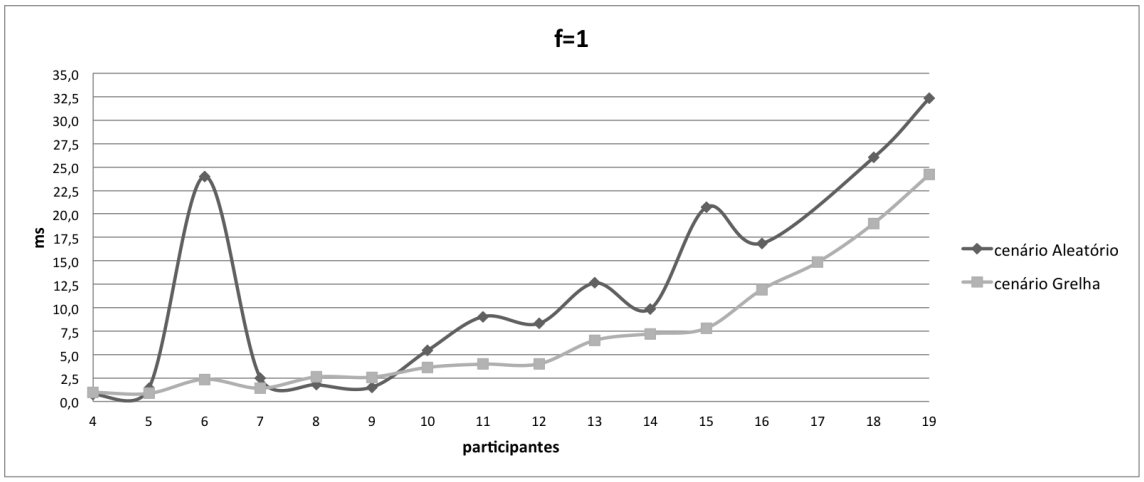


Figura 5.4: Resultados dos testes de latência num cenário Aleatório e $f = 1$.

cenário Aleatório (com mobilidade). Enquanto que num cenário Grelha (sem mobilidade) a perda de pacotes tende-se a estabilizar. Portanto a mobilidade de participantes influencia a perda de pacotes por participante.

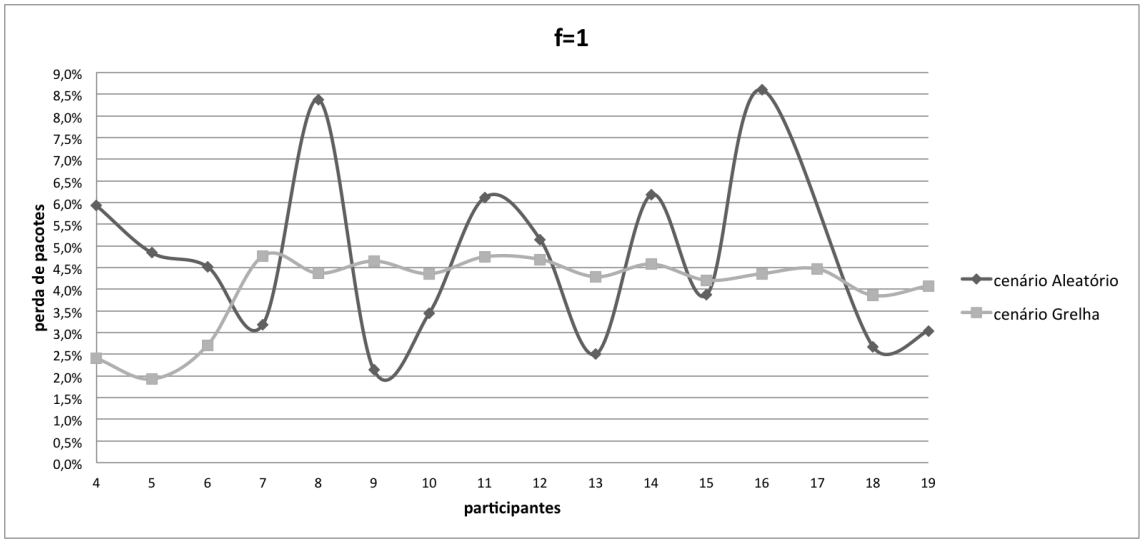


Figura 5.5: Resultados dos testes de perda de pacotes por participante com $f = 1$.

5.3 Sumário

Sucintamente, algumas das conclusões da avaliações experimentais retiradas são:

- A proposta de modificações e optimizações aumentam a escabilidade dos algoritmos originais de Alchieri et al [4];
- A mobilidade tem um grande impacto no desempenho da pilha protocolar, aumentando o tempo médio de execução exponencialmente;

- A difusão e a proximidade dos participantes reduz a taxa de transmissão e aumenta a taxa de recepção;
- A latência é linearmente proporcional ao número de participantes e à velocidade destes;
- O DSDV proporciona um melhor desempenho à pilha até experiências com dezoito participantes móveis;
- A pilha protocolar não provocam o estrangulamento em redes com 11mbps de largura de banda, pois não ultrapassam o consumo de 5kbps em ambos os cenários.

Capítulo 6

Conclusão

O objetivo desta tese foi a realização e o estudo de um "middleware" de suporte à comunicação e coordenação para aplicações móveis distribuídas e cooperativas de redes MANET.

Foi apresentada a realização de uma pilha protocolar que concretiza as modificações e otimizações propostas aos algoritmos de Alchieri et al. [4]. Os algoritmos criam uma filiação para execução de protocolos de acordo tolerante a faltas bizantinas.

Foi realizada uma análise comparativa entre a pilha com os algoritmos de Alchieri et al. e a pilha com os algoritmos modificados. Retirou-se os resultados das simulações de ambas as pilhas aumentando o conjunto de participantes e o número de faltas bizantinas, que mostraram que a pilha com algoritmos originais tinha dificuldades em escalar, havendo a necessidade de se introduzirem as otimizações.

As métricas usadas nos testes de desempenho da pilha foram essencialmente o tempo médio de execução da pilha, o consumo de largura de banda e a média da latência das transmissões da pilha. Os resultados baseiem-se em amostras obtidas em simulações de trinta segundos em cenários de participantes com ou sem mobilidade. Os testes demonstraram um aumento linear do tempo médio de execução da pilha num cenário de participantes sem mobilidade, enquanto num cenário composto por participantes móveis observou-se um aumento exponencial a partir de vinte e cinco participantes.

Uma avaliação igualmente importante realizada, foi a análise comparativa do desempenho da pilha em redes MANET quando é executada sobre dois protocolos de encaminhamento de MANET: o AODV e o DSDV. Esta análise demonstrou que o DSDV proporciona um maior desempenho do que o AODV até vinte participantes, devido à sua proatividade na construção de rotas e consumo de largura de banda contínuo.

6.1 Trabalho futuro

A pilha protocolar apresentada demonstrou um bom desempenho em redes ad hoc sem fios móveis em cenários com participantes de mobilidade a velocidades reduzidas, i.e.,

entre $0.7m/s$ a $0.9m/s$. Este tipo de cenário assemalha-se à mobilidade de pessoas com dispositivos móveis como "smartphones". Seria interessante efetuar testes reais da pilha em tais dispositivos, e o que justificaria a sua concretização em sistemas operativos como o *Android*. Primeiro é necessário concretizar um protocolo de encaminhamento de redes MANET, uma vez que, tanto quanto sabemos, não existe nenhum protocolo destes em Android. Baseando-nos na análise comparativa do desempenho de protocolos de encaminhamento, seria interessante uma investigação aprofundada na construção de protocolhos proativos de encaminhamento resilientes a faltas.

Apêndice A

Protocolos de Alchieri et al.

A.1 Reachable Reliable Broadcast

Algoritmo 1 *Reachable Reliable Broadcast* executado pelo participante i

Implements:
Reachable Reliable Broadcast, **instance** rrb
Uses:
Reliable channel, **instance** rc

f *: upper limit for number of failures*

1: **upon event** ⟨ `rrb, Send` | m ⟩ **do**
2: **for all** $j \in i.PD$ **do**
3: **trigger** ⟨ `rc, Send` | $j, < FLOODING, \{i\}, m$ ⟩
4:
5: **upon event** ⟨ `rc, Receive` | $j, < FLOODING, s, m >$ ⟩ **do**
6: **if** $getLastElement(s) = j \wedge i \notin s$ **then**
7: $appendRoute(s, i)$
8: $received\_msgs \leftarrow received\_msgs \cup \{m, s\}$
9: $routes \leftarrow computeRoutes(m, received\_msgs)$
10: **if** $routes \geq f + 1$ **then**
11: $initiator \leftarrow getFirstElement(s)$
12: **trigger** ⟨ `rrb, Receive` | initiator, m ⟩
13: $received\_msgs \leftarrow received\_msgs \setminus \{m, *\}$
14: **for all** $\langle j \rangle \in i.PD \setminus \{j\}$ **do**
15: **trigger** ⟨ `rl, Send` | $j, < FLOODING, \{i\}, m$ ⟩
16:

A.2 Discovery Protocol

Algoritmo 2 *Discovery Protocol* executado pelo participante i

Implements:
Discovery Protocol, **instance** dis
Uses:
Reliable link, **instancia** rl
Reachable Reliable Broadcast, **instancia** rrb

f : upper limit for number of failures

1: **upon event** $\langle$ `dis, Init` $\mid f \rangle$ **do**
2: $known, msg\_pend \leftarrow \{i\} \cup i.PD$
3: $nei\_pend, received, asked \leftarrow \emptyset$
4: **trigger** $\langle$ `rrb, Send` $\mid < GET\_NEIGHBOR > \rangle$
5:
6: **upon event** $\langle$ `rrb, Receive` $\mid j, < GET\_NEIGHBOR > \rangle$ **do**
7: **if** $decision = \perp$ **then**
8: $asked \leftarrow asked \cup \{j\}$
9: **else**
10: **trigger** $\langle$ `rc, Send` $\mid j, < DECISION, decision > \rangle$
11: **trigger** $\langle$ `rc, Send` $\mid < SET\_NEIGHBOR, i.PD \rangle$
12:
13: **upon event** $\langle$ `rc, Receive` $\mid j, < SET\_NEIGBOR, neighbor > \rangle$ **do**
14: $received \leftarrow received \cup \{js\}$
15: $msg\_pend \leftarrow msg\_pend \setminus \{j\}$
16: $nei\_pend \leftarrow nei\_pend \cup \{\langle j, neighbor \rangle\}$
17: **for all** $j \in s \wedge | < *, s > \in nei\_pend| > f \wedge j \notin known$ **do**
18: $known \leftarrow known \cup \{j\}$
19: **if** $j \notin received$ **then**
20: $asked \leftarrow asked \cup \{j\}$
21: **for all** $\langle j, neighbor \rangle \in nei\_pend$ **do**
22: **if** $\forall z \in neighbor : z \in known$ **then**
23: $nei\_pend \leftarrow nei\_pend \setminus \{\langle j, neighbor \rangle\}$
24: **if** $|nei\_pend| + |msg\_pend| \leq f$ **then**
25: **trigger** $\langle$ `dis, Finish` $\mid known \rangle$
26:

A.3 Sink Protocol

Algoritmo 3 *Sink Protocol* executado pelo participante i

Implements:
Sink Protocol, **instance** sin;
Uses:
Reliable channel, **instancia** rl;
Discovery Protocol, **instancia** dis;

f : upper limit for number of failures

1: **upon event** ⟨ `sin, Init` | f ⟩ **do**
2: **trigger** ⟨ `dis, Init` | f ⟩
3:
4: **upon event** ⟨ `dis, Finish` | k ⟩ **do**
5: $known \leftarrow k$
6: $acked \leftarrow \{i\}$
7: $nacked \leftarrow \emptyset$
8: **for all** $j \in known \wedge j \neq i$ **do**
9: **trigger** ⟨ `rc, Send` | $j, < REQUEST, known >$ ⟩
10:
11: **upon event** ⟨ `rl, Receive` | j, <REQUEST, k> ⟩ **do**
12: **if** (**then**)$known = k$
13: **trigger** ⟨ `rc, Send` | $j, < RESPONSE, ACK >$ ⟩
14: **else**
15: **trigger** ⟨ `rc, Send` | $j, < RESPONSE, NACK >$ ⟩
16:
17: **upon event** ⟨ `rc, Receive` | j, <RESPONSE, value> ⟩ **do**
18: **if** $value = NACK$ **then**
19: $nacked \leftarrow nacked \cup \{j\}$
20: **if** $|nacked| > f$ **then**
21: **trigger** ⟨ `sin, Finish` | false, known, asked ⟩
22: **else**
23: $acked \leftarrow acked \cup \{j\}$
24: **if** $|acked| \geq |known| - f$ **then**
25: **trigger** ⟨ `sin, Finish` | true, known, asked ⟩
26:

A.4 Consensus

Algoritmo 4 *Consensus* executado pelo participante i

Implements:

Consensus, **instance** con

Uses:

Sink Protocol, **instances** sin;
Byzantine fault-tolerant consensus protocol, **instances** conp;
Reliable channel, **instances** rc.

f : upper limit for number of failures

1: **upon event** $\langle$ `epa, Init` | f $\rangle$ **do**
2: $decision \leftarrow \perp$
3: $values, known, asked \leftarrow \emptyset$
4: **trigger** $\langle$ `sin, Init` | f $\rangle$
5:
6: **upon event** $\langle$ `sin, Get` | $true, sink, asked$ $\rangle$ **do**
7: $asked \leftarrow asked$
8: **trigger** $\langle$ `conp, Propose` | initial, sink $\rangle$
9:
10: **upon event** $\langle$ `conp, Decide` | $value$ $\rangle$ **do**
11: $decision \leftarrow value$
12: **for all** $j \in asked$ **do**
13: **trigger** $\langle$ `rc, Send` | $j, < SET\_DECISION, decision >$ $\rangle$
14: **trigger** $\langle$ `con, Decide` | $decision$ $\rangle$
15:
16: **upon event** $\langle$ `rc, Receive` | $j, < SET\_DECISION, value >$ $\rangle$ **do**
17: **if** $decision = \perp$ **then** $values \leftarrow values \cup \{< value, j >\}$
18: **if** $|\forall < value, * > \in values| > f$ **then**
19: $decision \leftarrow value$
20: **trigger** $\langle$ `con, Decide` | decision $\rangle$
21:

Apêndice B

Algoritmos dos protocolos da pilha protocolar

B.1 Canal persistente

Algoritmo 5 Canal persistente executado pelo participante i.

Implements:

Canal persistente, **instances** cp

Uses:

Canal não fiável, **instances** cnf

1: **upon event** $\langle$ `cp, Init` $|$ $\rangle$ **do**
2: $messages \leftarrow \emptyset$
3: `starttimer(`Δ`)`
4:
5: **upon event** `Timeout` **do**
6: **for all** $\langle j, m \rangle \in messages$ **do**
7: **trigger** $\langle$ `cnp, Send` $| j, m \rangle$
8: **if** $\langle j, m \rangle \in removef\_messages$ **then**
9: $messages \leftarrow messages \setminus \{< j, m >\}$
10: $removed\_messages \leftarrow removed\_messages \setminus \{< j, m >\}$
11: `starttimer(`Δ`)`
12:
13: **upon event** $\langle$ `cp, Send` $| j, m \rangle$ **do**
14: $messages \leftarrow messages \cup \{< j, m >\}$
15:
16: **upon event** $\langle$ `cnf, Deliver` $| j, m \rangle$ **do**
17: **trigger** $\langle$ `cp, Deliver` $| j, m \rangle$
18:
19: **upon event** $\langle$ `cnp, Stop` $| j, m \rangle$ **do**
20: $removed\_messages \leftarrow removed\_messages \cup \{< j, m >\}$
21:

B.2 Canal perfeito

Algoritmo 6 Canal perfeito executado pelo participante i.

Implements:

Canal perfeito, **instances** cpe

Uses:

Canal persistente, **instances** cp
Canal não fiável, **instances** cnf

1: **upon event** $\langle$ `cpe, Send` $\mid j, m \rangle$ **do**
2: **trigger** $\langle$ `cnf, Send` $\mid j, m \rangle$
3:
4: **upon event** $\langle$ `cp, Deliver` $\mid j, m \rangle$ **do**
5: **if** $m \notin delivered\_messages$ **then**
6: $delivered\_messages \leftarrow delivered\_messages \cup \{m\}$
7: **trigger** $\langle$ `cpe, Deliver` $\mid j, m \rangle$
8: **trigger** $\langle$ `cnf, Send` $\mid j, < ACK, m > \rangle$
9:
10: **upon event** $\langle$ `cnf, Deliver` $\mid j, < ACK, m > \rangle$ **do**
11: **trigger** $\langle$ `cnf, Stop` $\mid j, m \rangle$
12:
13: **upon event** $\langle$ `cpe, Stop` $\mid j, m \rangle$ **do**
14: **trigger** $\langle$ `cnp, Stop` $\mid j, m \rangle$
15:

B.3 Disseminação alcançável bizantina

Algoritmo 7 Disseminação alcançável bizantina executado pelo participante i.

Implements:

Disseminação alcançável bizantina, **instances** dab

Uses:

Canal perfeito, **instances** cpe

Canal persistente, **instances** cp

f : upper limit for number of failures

1: **upon event** $\langle$ `dp, Init` | f $\rangle$ **do**
2: $received\_msgs \leftarrow \emptyset$
3:
4: **upon event** $\langle$ `dp, Send` | j, m $\rangle$ **do**
5: **trigger** $\langle$ `cp, Send` | $ALL, < FLOODING, m, \{i\} >$ $\rangle$
6: `starttimer`(Δ)
7:
8: **upon event** `Timeout` **do**
9: **for all** $< m, * > \in received\_msgs$ **do**
10: **trigger** $\langle$ `cp, Stop` | $ALL, < FLOODING, m, s >$ $\rangle$
11: **for all** $j \in < m, j > \notin acked$ **do**
12: **trigger** $\langle$ `cpe, Send` | $j, < FLOODING, m, path >$ $\rangle$
13:
14: **upon event** $\langle$ `cp, Receive` | j, $< FLOODING, m, path >$ $\rangle$ **do**
15: **if** $j \in neighbors$ **then**
16: $acked \leftarrow acked \cup \{< m, j >\}$
17: **if** `IsValid`$(m, j) \wedge$ `IsDisjoint`$(< m, path >, received\_msgs)$
18: $\wedge |\forall < m, * > \in received\_msgs| \leq f$ **then**
19: $received\_msgs \leftarrow received\_msgs \cup \{< m, path\}$
20: **if** $|\forall < m, * > \in received\_msgs| \geq f$ **then**
21: **trigger** $\langle$ `dp, Receive` | j, m $\rangle$
22: $path \leftarrow$ `Append`$(path, i)$
23: **trigger** $\langle$ `cp, Send` | $ALL, < FLOODING, m, path >$ $\rangle$
24: **trigger** `starttimer`(Δ)
25:
26: **upon event** $|\forall < m, * > \in acked| \geq |neighbors| - f$ **do**
27: **trigger** $\langle$ `cp, Stop` | ALL, m $\rangle$
28: **for all** $j \in neighbors$ **do**
29: **trigger** $\langle$ `cpe, Stop` | j, m $\rangle$
30:

Variável ou Função	*Descrição*
$acked$	conjunto de participantes que confirmaram a mensagem m com o caminho $path$
$neighbors$	conhecimento inicial (ou vizinhaça) do participante
$receveid\_msg$	conjunto de mensagens a confirmar
`IsValid`	se o identificador j é o último membro da lista $path$ e não contêm repetição de participantes, retorna verdadeiro; caso contrário retorna falso
`IsDisjoint`	se dois caminhos são disjuntos, retorna verdadeiro; caso contrário retorna falso

Tabela B.1: Notações usadas Algoritmo Disseminação alcançável bizantina.

B.4 Descoberta de participantes

Algoritmo 8 Descoberta de participantes executado pelo participante i.

Implements:

Descoberta de participantes, **instances** dp

Uses:

Canal persistente, **instances** cp

```
1:  upon event ⟨ dp, Init | ⟩ do
2:      trigger ⟨ cnf, Send | ALL, HELLO ⟩
3:      starttimer(Δ)
4:
5:  upon event Timeout do
6:      trigger ⟨ dp, Get | neighbors ⟩
7:
8:  upon event ⟨ cp, Receive | j, HELLO ⟩ do
9:      neighbors ← neighbors ∪ {j}
10:
```

B.5 Descoberta de participantes na rede

Algoritmo 9 Descoberta de participantes na rede.

Implements:

Descoberta de participantes na rede, **instances** des

Uses:

Canal persistente, **instances** cp
Canal perfeito, **instances** cpe
Descoberta de participantes, **instances** dp

1: **upon event** $\langle$ `des, Init` | f, t $\rangle$ **do**
2: **trigger** $\langle$ `dp, Start` | t $\rangle$
3:
4: **upon event** $\langle$ `dp, Get` | n $\rangle$ **do**
5: $neighbors \leftarrow n$
6: $known \leftarrow neighbors \cup \{i\}$
7: $msg\_pend \leftarrow neighbors$
8: $nei\_pend \leftarrow \{i, neighbors\}$
9: $received, asked \leftarrow \{i\}$
10: **trigger** $\langle$ `da, Send` | $GET\_NEIGHBOR$ $\rangle$
11:
12: **upon event** $\langle$ `da, Receive` | $j, GET\_NEIGHBOR$ $\rangle$ **do**
13: **if** $decision = \perp$ **then**
14: $asked \leftarrow asked \cup \{j\}$
15: **else**
16: **trigger** $\langle$ `cpe, Send` | $j, < DECISION, decision >$ $\rangle$
17: **trigger** $\langle$ `cpe, Send` | j, $< SET\_NEIGHBOR, neighbors >$ $\rangle$
18:
19: **upon event** $\langle$ `cpe, Receive` | $j, < SET\_NEIGHBOR, neighbors >$ $\rangle \wedge neighbors \neq \emptyset$ **do**
20: $received \leftarrow received \cup \{j\}$
21: $msgs\_pend \leftarrow msgs\_pend \setminus \{j\}$
22: $nei\_pend \leftarrow nei\_pend \cup \{j, neighbors\}$
23: **for all** $j : j \in s \wedge |\forall < *, s > \in nei\_pend| > f \wedge j \notin known$ **do**
24: $known \leftarrow known \cup j$
25: **if** $j \notin received$ **then**
26: $msgs\_pend \leftarrow msgs\_pend \cup \{i\}$
27: **for all** $j :< j, neighbors > \in nei\_pend$ **do**
28: $known \leftarrow known \cup \{j\}$
29: **if** $\forall z \in neighbors : z \in known$ **then**
30: $nei\_pend \leftarrow nei\_pend \setminus < z, neighbors >$
31: **if** $|nei\_pend| + |msg\_pend| \leq f$ **then**
32: **trigger** $\langle$ `dpr, Get` | $known$ $\rangle$
33:

Variável	*Descrição*
$neighbors$	conhecimento inicial (ou vizinhaça) do participante;
nei_pend	conjunto de participantes que ainda não foram confirmados;
msg_pend	conjunto de participantes que supostamente enviaram o seu conhecimento (ou vizinhaça);
$received$	conjunto de participantes que enviaram o seu conhecimento (ou vizinhaça);
$asked$	conjunto de participantes que requeriram a decisão;
$decision$	o valor decidido;

Tabela B.2: Notações usadas no algoritmo Descoberta de participantes na rede.

B.6 Determinação do poço

Algoritmo 10 Determinação do poço.

Implements:

Determinação do poço, **instances** poc.

Uses:

Descoberta de participantes na rede, **instancia** des;
Canal perfeito, **instancia** cpe.

```
1:  upon event ⟨ des, Get | known, asked ⟩ do
2:      known ← SubSet(OrderAsc(known), 3f + 1)
3:      if i ∈ known then
4:          acked ← {i}
5:          nacked ← ∅
6:          starttimer(Δ)
7:          trigger ⟨ cpe, Send | ALL, < REQUEST, sink > ⟩
8:      else
9:          trigger ⟨ poc, Get | j, < false, sink > ⟩
10:
11: upon event Timeout
12:     trigger ⟨ cpe, Stop | ALL, < REQUEST, sink > ⟩
13:     for all j : j ∈ sink do
14:         trigger ⟨ cpe, Send | j, < REQUEST, sink > ⟩
15:
16: upon event ⟨ cpe, Receive | j, < REQUEST, oKnown > ⟩ ∧sink ≠ ∅ do
17:     if known = oKnown then
18:         trigger ⟨ cpe, Send | j, < RESPONSE, ack > ⟩
19:     else
20:         trigger ⟨ cpe, Send | j, < RESPONSE, nack > ⟩
21:
22: upon event ⟨ cpe, Receive | j, < RESPONSE, value > ⟩ do
23:     if value = nack then
24:         nacked ← nacked ∪ j
25:         if |nacked| > f then
26:             trigger ⟨ poc, Send | false, known, asked ⟩
27:     else
28:         acked ← acked ∪ j
29:         if |acked| ≥ |known| − f then
30:             trigger ⟨ poc, Send | true, known, asked ⟩
31:
```

Variável ou Função	*Descrição*
$known$	conjunto de participantes conhecidos (e confirmados)
$nacked$	conjunto de participantes que não têm o mesmo conjunto de participantes conhecidos
$acked$	conjunto de participantes que têm o mesmo conjunto de participantes conhecidos
`OrderAsc`	retorna um conjunto ordenado (de forma ascendente)
`SubSet`	retorna um subconjunto dos primeiros $3f + 1$ elementos

Tabela B.3: Notações usadas no algoritmo Determinação do poço.

B.7 Execução do protocolo de acordo

Algoritmo 11 Execução do protocolo de acordo realizado pelo participante i.

Implements:

Execução do protocolo de acordo, **instance** acor

Uses:

Determinação do poço, **instances** poc
Protocolo de acordo tolerante a faltas bizantinas, **instances** conp
Canal perfeito, **instances** cpe

f *: upper limit for number of failures*

```
1:  upon event ⟨ acor, Init | f ⟩ do
2:      decision ← ⊥
3:      values, sink, asked ← ∅
4:      trigger ⟨ poc, Init | f ⟩
5:
6:  upon event ⟨ poc, Finish | true, sink, a ⟩ do
7:      asked ← a
8:      trigger ⟨ conp, Propose | initial, sink ⟩
9:
10: upon event ⟨ conp, Decide | value ⟩ do
11:     decision ← value
12:     for all j ∈ asked do
13:         trigger ⟨ cpe, Send | j, < SET_DECISION, decision > ⟩
14:     trigger ⟨ con, Decide | decision ⟩
15:
16: upon event ⟨ conp, Receive | j, < SET_DECISION, value > ⟩ do
17:     if decision = ⊥ then
18:         values ← values ∪ {< value, j >}
19:         if |∀ < value, * >∈ values| > f then
20:             decision ← value
21:             trigger ⟨ acor, Decide | decision ⟩
22:
```

Variável	*Descrição*
$decision$	o valor decidido
$asked$	conjunto de participantes que requeriram o valor
$sink$	a filiação acordado
$values$	conjunto de valores (supostamente) da decisão recebidos mas não confirmados

Tabela B.4: Notações usadas no algoritmo de Execução do protocolo de acordo.

Lista de Abreviaturas

AH IPSec Authentication Header protocol

AODV An On-demand Distance Vector

BFT-CUP Byzantine Fault-Tolerant Consensus with Unknown Participants

DSDV Destination-Sequence Distance Vector Routing Algorithm

FT-CUP Fault-Tolerant Consensus with Unknown Participants

IP Internet Protocol

IPSec IP Security Protocol

MANET Mobile Ad-hoc Network

MPR Multipoint relay

OLSR Optimized Link State Routing Protocol

RERR Route Error Message

RREP Route Reply Packet

RREQ Route Request Packet

TCP Transmission Control Protocol

UDP User Datagram Protocol

Bibliografia

[1] Y. Afek, H. Attiya, A. Fekete, M. Fischer, N. Lynch, Y. Mansour, D. Wang, and L. Zuck. Reliable communication over unreliable channels. *Journal of ACM*, 41(6):1267–297, November 1994.

[2] D. Agrawal and Q. Zeng. *Introduction to Wireless and Mobile Systems*. Cengage Learning, 2010.

[3] E. Alcheiri, L. Tomelin, Alysson Bessani, and J. Fraga. Aspectos práticos sobre o consenso bizantino entre participantes desconhecidos. In *Proceedings of The Workshop de Testes e Tolerância a Falhas*, pages 63–76, May 2011.

[4] E. Alchieri, A. Bessani, S. Fraga, and F. Greve. Byzantine consensus with unknown participants. In Theodore P. Baker, Alain Bui, and Sébastien Tixeuil, editors, *Principles of Distributed Systems*, volume 5401 of *Lecture Notes in Computer Science*, pages 22–40, Berlin, Heidelberg, December 2008. Springer.

[5] M. Barborak, A. Dahbura, and M. Malek. The consensus problem in fault-tolerant computing. *Journal of ACM Computing Surveys*, 25(2):171–220, June 1993.

[6] A. Basu, B. Charron-Bost, and S. Toueg. Simulating reliable links with unreliable links in the presence of process crashes. In *Proceedings of The 10th International Workshop on Distributed Algorithms*, pages 105–122, 1996.

[7] C. Cachin, R. Guerraoui, and L. Rodrigues. *Introduction to Reliable and Secure Distributed Programming*. Springer, 2011.

[8] D. Cavin, Y. Sasson, and A. Schiper. Consensus with unknown participants or fundamental self-organization. In Ioanis Nikolaidis, Michel Barbeau, and Evangelos Kranakis, editors, *Ad-Hoc, Mobile, and Wireless Networks*, volume 3158 of *Lecture Notes in Computer Science*, pages 135–148. 2004.

[9] D. Cavin, Y. Sasson, and A. Schiper. Reaching Agreement with Unknown Participants in Mobile Self-Organized Networks in Spite of Process Crashes. Technical report, School of Computer and Communication Sciences, Ecole Polytechnique Fédérale de Lausanne, 2005.

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

[11] X. Chen, K. Makki, Yen. K, and N. Pissinou. Sensor network security: a survey. *Journal of IEEE Communications Surveys Tutorials*, 11(2):52–73, 2009.

[12] T. Clausen and P. Jacquet. Optimized Link State Routing Protocol (OLSR). RFC 3626, 2003.

[13] M. Correia, N. Neves, and P. Veríssimo. From consensus to atomic broadcast: Time-free byzantine-resistant protocols without signatures. *Computing Journal*, 49(1):82–96, January 2006.

[14] M. Correia, G. Veronese, N. Neves, and P. Veríssimo. Byzantine consensus in asynchronous message-passing systems: a survey. *International Journal of Critical Computer-Based Systems*, 2(2):141–161, July 2011.

[15] S. Corson and R. Macker. Mobile Ad hoc Networking (MANET): Routing Protocol Performance Issues and Evaluation Considerations. RFC 2501, 1999.

[16] D. Dolev, C. Dwork, and L. Stockmeyer. On the minimal synchronism needed for distributed consensus. *Journal of ACM*, 34(1):77–97, January 1987.

[17] D. Dolev, R. Friedman, and D. Malkhi. Failure detectors in omission failure environments. In *Proceedings of The Sixteenth Annual ACM Symposium on Principles of Distributed Computing*, pages 286–, 1997.

[18] J. Douceur. The sybil attack. In *Proceedings of The First International Workshop on Peer-to-Peer Systems*, pages 251–260, 2002.

[19] C. Dwork, N. Lynch, and L. Stockmeyer. Consensus in the presence of partial synchrony. *Journal of ACM*, 35(2):288–323, April 1988.

[20] M. Fischer, N. Lynch, and M. Paterson. Impossibility of distributed consensus with one faulty process. *Journal of ACM*, 32(2):374–382, April 1985.

[21] I. Gler, M. Meghdadi, and S. Ozdemir. A Survey of Wormhole-based Attacks and their Countermeasures in Wireless Sensor Networks. volume 28, pages 89–102, 2011.

[22] F. Greve and S. Tixeuil. Knowledge connectivity vs. synchrony requirements for fault-tolerant agreement in unknown networks. In *Proceedings of The 37th Annual IEEE/IFIP International Conference on Dependable Systems and Networks*, pages 82–91, June 2007.

[23] R. Guerraoui, R. Olivera, and A. Schiper. Stubborn communication channels. Technical report, Département d' Informatique, Ecole Polytechnique Fédérale de Lausanne, 1996.

[24] K. Herley, A. Pietracaprina, and G. Pucci. Store-and-forward multicast routing on the mesh. *Journal of Theory of Computing Systems*, 42(4):519–535, 2008.

[25] J. Hoebeke, I. Moerman, B. Dhoedt, and P. Demeester. An Overview of Mobile Ad Hoc Networks: Applications and Challenges. *Journal of the Communications Network*, 3:60–66, July 2004.

[26] Y. Hu, A. Perrig, and D. Johnson. Wormhole attacks in wireless networks. *IEEE Journal on Selected Areas in Communications*, 24(2):370–380, 2006.

[27] J. Kamal, M. Hasan, A. Griffiths, and H. Yu. Development and verification of simulation model based on real manet experiments for transport layer protocols (udp and tcp). *Internacional Journal of Automation and Computing*, 10(1):53–63, February 2013.

[28] C. Karlof and D. Wagner. Secure routing in wireless sensor networks: attacks and countermeasures. In *Proceedings of The IEEE International Workshop on Sensor Network Protocols and Applications*, pages 113–127, 2003.

[29] S. Kent and R. Atkison. Security Architecture for the Internet Protocol. RFC 2401, 1998.

[30] L. Lamport. The part-time parliament. *Journal of ACM*, 16(2):133–169, May 1998.

[31] L. Lamport, R. Shostak, and M. Pease. The byzantine generals problem. *Journal of ACM*, 4(3):382–401, July 1982.

[32] A. Lapidoth and P. Narayan. Reliable communication under channel uncertainty. *Journal of IEEE Transactions on Information Theory*, 44(6):2148–2177, 1998.

[33] R. Maheshwari, G. Jie, and S. Das. Detecting wormhole attacks in wireless networks using connectivity information. In *Proceedings of The 26th IEEE International Conference on Computer Communications*, pages 107–115, 2007.

[34] H. Moniz, N. Neves, M. Correia, and P. Veríssimo. Ritas: Services for randomized intrusion tolerance. *IEEE Transactions on Dependable and Secure Computing*, 8(1):122–136, 2011.

[35] M. Nácher, C. Calafate, J. Cano, and P. Manzoni. Comparing tcp and udp performance in manets using multipath enhanced versions of dsr and dymo. In *Proceedings of The 4th ACM Workshop on Performance evaluation of Wireless Ad-hoc, Sensor, and Ubiquitous Networks*, pages 39–45, 2007.

[36] S. Ni, Y. Tseng, Y. Chen, and J. Sheu. The broadcast storm problem in a mobile ad hoc network. In *Proceedings of the 5th Annual ACM/IEEE International Conference on Mobile Computing and Networking*, pages 151–162, 1999.

[37] ns3. Loukas lazos home page - ese, 2013. [Online; accessed 27-September-2013].

[38] ns3. Netanim - nsnam, 2013. [Online; accessed 27-September-2013].

[39] ns3. ns-3 - nsnam, 2013. [Online; accessed 27-September-2013].

[40] C. Perkins, E. Belding-Royer, and S Das. Ad hoc On-Demand Distance Vector (AODV) Routing. RFC 3561, 2003.

[41] C. Perkins and P. Bhagwat. Highly dynamic destination-sequenced distance-vector routing (dsdv) for mobile computers. 24(4):234–244, October 1994.

[42] C. Perkins and E. Royer. Ad-hoc on-demand distance vector routing. In *Proceedings of The Second IEEE Workshop on Mobile Computing Systems and Applications*, pages 90–100, 1999.

[43] F. Rango, M. Fotino, C. Calafate, P. Manzoni, and S. Marano. Olsr vs dsr: A comparative analysis of proactive and reactive mechanisms from an energetic point of view in wireless ad hoc networks. *Computer Communications*, 31(16):3843–3854, 2008.

[44] S. Toueg. Randomized Byzantine Agreements. In *Proceedings of The Third Annual ACM Aymposium on Principles of Distributed Computing*, pages 163–178, August 1984.

[45] M. Van der Schaar and P. Chou. *Multimedia over IP and Wireless Networks: Compression, Networking, and Systems*. Elsevier Science, 2011.

[46] D. Wolinsky, P. Juste, P. Boykin, and R. Figueiredo. Addressing the p2p bootstrap problem for small overlay networks. In *The 10th IEEE International Conference on Peer-to-Peer Computing*, pages 1–10, 2010.

[47] C. Yidong and T. Yang. On connectivity and delay in manet. In *Proceedings of The Third International Conference on Communications and Networking in China*, pages 488–492, 2008.

[48] M. Yu, M. Zhou, and W. Su. A secure routing protocol against byzantine attacks for manets in adversarial environments. *IEEE Transactions on Vehicular Technology*, 58(1):449–460, 2009.